GI-Edition

Lecture Notes in Informatics

Volker Markl, Gunter Saake, Kai-Uwe Sattler,
Gregor Hackenbroich, Bernhard Mitschang,
Theo Härder, Veit Köppen (Hrsg.)

Datenbanksysteme für Business, Technologie und Web (BTW) 2013

13.–15. März 2013
Magdeburg

Proceedings

214

V. Markl et al. (Hrsg.): BTW 2013

Volker Markl, Gunter Saake, Kai-Uwe Sattler,
Gregor Hackenbroich, Bernhard Mitschang,
Theo Härder, Veit Köppen (Hrsg.)

Datenbanksysteme für Business, Technologie und Web (BTW)

15. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS)

13. – 15.03.2013 in Magdeburg, Germany

Gesellschaft für Informatik e.V. (GI)

Lecture Notes in Informatics (LNI) - Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-214

ISBN 978-3-88579-608-4
ISSN 1617-5468

Volume Editors

Volker Markl
Fachgebiet Datenbanksysteme und Informationsmanagement
Technische Universität Berlin, Einsteinufer 17
10587 Berlin, Germany
E-Mail: Volker.Markl@tu-berlin.de

Gunter Saake
Otto-von-Guericke-Universität Magdeburg
Institut für Technische und Betriebliche Informationssysteme
Universitätsplatz 2
39106 Magdeburg, Germany
E-Mail: Gunter.Saake@ovgu.de

Kai-Uwe Sattler
Technische Universität Ilmenau
FG Datenbanken und Informationssysteme
Postfach 100 565
98684 Ilmenau, Germany
E-Mail: kus@tu-ilmenau.de

Gregor Hackenbroich
SAP AG, SAP Research
Chemnitzer Str. 48
01187 Dresden, Germany
Email: Gregor.Hackenbroich@sap.com

Bernhard Mitschang
Institut für Parallele und Verteilte Systeme (IPVS)
Universität Stuttgart
70569 Stuttgart, Germany
E-Mail: Bernhard.Mitschang@ipvs.uni-stuttgart.de

Theo Härder
Fachbereich Informatik
Universität Kaiserslautern
67653 Kaiserslautern, Germany
E-Mail: Haerder@informatik.uni-kl.de

Veit Köppen
Otto-von-Guericke-Universität Magdeburg
Institut für Technische und Betriebliche Informationssysteme
Universitätsplatz 2
39106 Magdeburg, Germany
E-Mail: Veit.Koeppen@ovgu.de

printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

In den letzten Jahren hat es auf dem Gebiet des Datenmanagements große Veränderungen gegeben. Dabei muss sich die Datenbankforschungsgemeinschaft insbesondere den Herausforderungen von „Big Data“ stellen, welches die Analyse von riesigen Datenmengen unterschiedlicher Struktur mit kurzen Antwortzeiten erfordert. Neben klassisch strukturierten Daten müssen moderne Datenbanksysteme und Anwendungen ebenfalls semistrukturierte, textuelle und andere multi-modale Daten sowie Datenströme in völlig neuen Größenordnungen verwalten. Gleichzeitig müssen die Verarbeitungssysteme die Korrektheit und Konsistenz der Daten sicherstellen.

Die jüngsten Fortschritte bei Hardware und Rechnerarchitektur ermöglichen neuartige Datenmanagementtechniken, die von neuen Index- und Anfrageverarbeitungsparadigmen (In-Memory, SIMD, Multicore) bis zu neuartigen Speichertechniken (Flash, Remote Memory) reichen. Diese Entwicklungen spiegeln sich in aktuell relevanten Themen wie Informationsextraktion, Informationsintegration, Data Analytics, Web Data Management, Service-Oriented Architectures, Cloud Computing oder Virtualisierung wider.

Die 15. GI-Fachtagung „Datenbanksysteme für Business, Technologie und Web“ (BTW 2013) befasst sich mit diesen Themen vom 11. bis 15. März 2013 an der Otto-von-Guericke-Universität in Magdeburg, im Rahmen eines wissenschaftlichen Programms, eines Industrieprogramms, durch Vorstellung von Demonstratoren sowie durch begleitende Tutorien und Workshops. Gleichzeitig werden aktuelle Fragestellungen in drei Keynotes erörtert.

Als Keynote-Redner werden Divyakant Agrawal von der Universität Santa Barbara in Kalifornien, Daniel Keim von der Universität Konstanz und Paul Larson von Microsoft Research zu den Themen „Big Data Analytics“ und „Informationsvisualisierung“ sowie zu „spaltenorientierten Techniken und Anfrageverarbeitung bzw. -optimierung in Hauptspeicherdatenbanken“ sprechen. Ergänzend dazu werden in einem Panel Fragestellungen zu Technologietransfer und Innovationen im Datenbankbereich erörtert.

Für das wissenschaftliche Programm wurden acht Langbeiträge sowie zehn Kurzbeiträge aus insgesamt 54 Einreichungen angenommen, die sich sowohl mit theoretischen als auch anwendungsorientierten wissenschaftlichen Aspekten zur Weiterentwicklung von Datenbanken und Informationssystemen befassen. Dies bedeutet eine Annahmequote von 13% für Langbeiträge und 33% für alle Beiträge. Dabei fallen die wissenschaftlichen Beiträge in die Kategorien „Indexierung“, „Datenströme und Workflows“, „Parallele Systeme und Algorithmen“,

„Information Retrieval und Anwendungen“, „Join-Verarbeitung“, „Anfrageverarbeitung“, sowie „Cloud Speichersysteme“.

Im Industrieprogramm spiegelt sich die Themenbreite von Datenbanken für Business Technology und Web wider. Aus neun Einreichungen wurden vom Industrieprogrammkomitee fünf Langbeiträge und zwei Kurzbeiträge ausgewählt. Durch zwei eingeladene Beiträge zu den Themenbereichen Datenstromverarbeitung und Datenbankanwendungen wurde das Industrieproramm inhaltlich abgerundet und so die Attraktivität speziell für Tagungsteilnehmer aus der Industrie nochmals erhöht.

Zum Demoprogramm wurden 22 Beiträge eingereicht, von denen 12 Beiträge zur Live-Demonstration während der Konferenz angenommen werden konnten. Die Themen reichen dabei von Kerntechniken für Datenbanksysteme über Anfragesprachen, Nutzerinteraktion und Korrelationsanalyse bis hin zu Anwendungen wie Simulation, Prozessmodellierung und Informationsextraktion bzw. -integration.

Zum siebten Mal wurde im Rahmen der BTW ein Wettbewerb um die beste Dissertation, diesmal für den Zeitraum Oktober 2010 bis September 2012, im Bereich der Datenbank- und Informationssysteme ausgeschrieben. Die Auszeichnung erhielt Dr. Stephan Günnemann für seine von Prof. Thomas Seidl betreute Dissertation "Subspace Clustering for Complex Data".

Die Ottostadt Magdeburg ist der Austragungsort für die 15. BTW im Jahr 2013. Der erste römisch-deutsche Kaiser Otto der Große und der Erfinder und Diplomat Otto von Guericke haben die Geschichte und Geschicke Magdeburgs maßgeblich geprägt und sie weit über die Grenzen hinaus bekannt gemacht.
Die Otto-von-Guericke-Universität mit ihrem Fokus auf den Ingenieurswissenschaften ist eine der jüngsten Universitäten Deutschlands. Trotzdem ist sie eine der ersten Universitäten in Ostdeutschland, die seit 1956 aktiv Forschung und Lehre auf dem Gebiet der Informatik betreibt und seit 1967 ihre Informatikstudiengänge anbietet. Neben der Informatik- und Wirtschaftsinformatikausbildung gestalten das SAP University Competence Center, das Fraunhofer Institut für Fabrikbetrieb und -automatisierung IFF sowie zahlreiche Einrichtungen im Umfeld die Forschung und Lehre am Standort Magdeburg.

Die Materialien zur BTW 2013 werden auch über die Tagung hinaus unter http://www.btw-2013.de zur Verfügung stehen.

Die Organisation einer so großen Tagung wie der BTW mit ihren angeschlossenen Veranstaltungen ist nicht ohne zahlreiche Partner und Unterstützer möglich. Sie sind auf den folgenden Seiten aufgeführt. Ihnen gilt unser besonderer Dank ebenso wie den Sponsoren der Tagung und der GI-Geschäftsstelle.

Berlin, Magdeburg, Ilmenau, Dresden, Stuttgart, Kaiserslautern, im Januar 2013

Volker Markl, Vorsitzender des Programmkomitees
Gunter Saake, Tagungsleitung und Vorsitzender des Organisationskomitees
Kai-Uwe Sattler, Vorsitzender des Demo-Programms
Gregor Hackenbroich und Bernhard Mitschang, Vorsitzende des Industrieprogramms
Theo Härder, Leitung Dissertationspreiskomitee
Veit Köppen, Tagungsband und Organisationskomitee

Tagungsleitung

Gunter Saake, Otto-von-Guericke-Universität Magdeburg

Organisationskomitee

Gunter Saake
Veit Köppen
Stefan Barthel
Sebastian Breß
Alexander Grebhahn
Thomas Leich
Siba Mohammad
Viktor Sayenko
Ivonne Schröter
Anja Strube
Eike Schallehn
David Broneske
Ziqiang Diao
Katja Gündel
Andreas Lübcke
Maik Mory
Matin Schäler

Studierendenprogramm

Thomas Neumann, TU München

Koordination Workshops

Andreas Henrich, Universität Bamberg
Wolfgang Lehner, TU Dresden

Programmkomitees

Wissenschaftliches Programm

Vorsitz: Volker Markl, TU Berlin

Wolf-Tilo Balke, TU Braunschweig
Christian Böhm, LMU München
Erik Buchmann, KIT, Karlsruhe
Stefan Deßloch, TU Kaiserslautern
Markus Endres, Univ. Augsburg
Rainer Gemulla, MPI, Saarbrücken
Theo Härder, TU Kaiserslautern
Arno Jacobsen, Univ. Toronto
Alfons Kemper, TU München
Meike Klettke, Univ. Rostock
Harald Kosch, Univ. Passau
Georg Lausen, Univ. Freiburg
Frank Leymann, Univ. Stuttgart
Thomas Mandl, Univ. Hildesheim
Rainer Manthey, Univ. Bonn
Carsten Binnig, DHBW
Alex Buchmann, TU Darmstadt
Stefan Conrad, HHU Düsseldorf
Jens Dittrich, Univ. Saarland
Norbert Fuhr, Univ. Duisburg Essen
Torsten Grust, Univ. Tübingen
Melanie Herschel, INRIA, France
Daniel Keim, Univ. Konstanz
Wolfgang Klas, Univ. Vienna
Birgitta König-Ries, Univ. Jena
Klaus Küspert, Univ. Jena
Ulf Leser, HU Berlin
Volker Linnemann, Univ. Lübeck
Stefan Manegold, CWI, Amsterdam
Klaus Meyer-Wegener, Univ. Erlangen

Karin Murthy, IBM India
Daniela Nicklas, Univ. Oldenburg
Erhard Rahm, Univ. Leipzig
Stefanie Rinderle-Ma, Univ. Wien
Eike Schallehn, OVGU Magdeburg
Ingo Schmitt, BTU Cottbus
Bernhard Seeger, Univ. Marburg
Uta Störl, HS Darmstadt
Jens Teubner, ETH Zürich
Mathias Weske, Univ. Potsdam
Felix Naumann, HPI, Potsdam
Peter Peinl, FH Fulda
Manfred Reichert, Univ. Ulm
Norbert Ritter, Univ. Hamburg
Stefanie Scherzinger, HS Regensburg
Holger Schwarz, Univ. Stuttgart
Thomas Seidl, RWTH Aachen
Myra Spiliopoulou, OVGU Magdeburg
Gottfried Vossen, Univ. Münster

Industrieprogramm

Vorsitz: Gregor Hackenbroich, SAP & Bernhard Mitschang, Univ. Stuttgart

Wolfgang Käfer, Daimler
Albert Maier, IBM
Philipp Rösch, SAP
Nelson Mattos, Google
Harald Schöning, Software AG
Thomas Ruf, GfK

Demoprogramm

Vorsitz: Kai-Uwe Sattler, TU Ilmenau

Stefan Conrad, Univ. Düsseldorf
Dirk Habich, TU Dresden
Thomas Kudraß, HTW Leipzig
Stefan Manegold, CWI Amsterdam
Thomas Neumann, TU München
Michael Gertz, Univ. Heidelberg
Katja Hose, MPI Saarbrücken
Alexander Löser, TU Berlin
Holger Meyer, Univ. Rostock
Knut Stolze, IBM Böblingen

Inhaltsverzeichnis

Eingeladene Vorträge

Wissenschaftliches Programm

Join Processing

Query Processing

Indexing

Industrieprogramm

DB-Implementierung

Data in Motion

Datenanalyse und Datensicherheit

Demo-Programm

Towards the End-to-End Design for Big Data Management in the Cloud: Why, How, and When?

Divy Agrawal

Department of Computer Science
University of California at Santa Barbara
Santa Barbara, CA 93106
agrawal@cs.ucsb.edu

Abstract

With the wide-scale adoption of cloud computing and with the explosion in the number of distributed applications and end-user devices, we are witnessing insatiable desire to build bigger-and-bigger systems that can serve hundreds of millions of end-users, are highly automated, and can collect enormous amounts of data in short periods of time. Often newer systems are implemented by integrating existing sub-systems that are already in use. A consequence of such a massive-scale integration is that it is very difficult to have a complete understanding of the overall system design. In fact, recent examples indicate that the only way to debug and test newer modules is to put them in live deployments that sometimes can lead to disastrous outcomes. In this talk, I will use some of the recent events in the context of Big Data and Cloud Computing as a motivation to argue that we need better methodologies for end-to-end system design for big data management in the cloud. I will then explore some well-known abstractions from distributed computing and databases as a means towards such a design and conclude with a contemplative question whether we can achieve such a goal or shall we leave it all to an automated self-learning and self-corrective oracle.

Biography: Dr. Divyakant Agrawal is a Professor of Computer Science and the Director of Engineering Computing Infrastructure at the University of California at Santa Barbara. His research expertise is in the areas of database systems, distributed computing, data warehousing, and large-scale information systems. From January 2006 through December 2007, Dr. Agrawal served as VP of Data Solutions and Advertising Systems at the Internet Search Company ASK.com. Dr. Agrawal has also served as a Visiting Senior Research Scientist at the NEC Laboratories of America in Cupertino, CA from 1997 to 2009. During his professional career, Dr. Agrawal has served on numerous Program Committees of International Conferences, Symposia, and Workshops and served as an editor of the journal of Distributed and Parallel Databases (1993-2008), and the VLDB journal (2003-2008). He currently serves as the Editor-in-Chief of Distributed and Parallel Databases and is on the editorial boards of the ACM Transactions on Database Systems and IEEE Transactions of Knowledge and Data Engineering. He has recently been elected to the Board of Trustees of the VLDB

Endowment and elected to serve on the Executive Committee of ACM Special Interest Group SIGSPATIAL. Dr. Agrawal's research philosophy is to develop data management solutions that are theoretically sound and are relevant in practice. He has published more than 320 research manuscripts in prestigious forums (journals, conferences, symposia, and workshops) on wide range of topics related to data management and distributed systems and has advised more than 35 Doctoral students during his academic career. He received the 2011 Outstanding Graduate Mentor Award from the Academic Senate at UC Santa Barbara. Recently, Dr. Agrawal has been recognized as an Association of Computing Machinery (ACM) Distinguished Scientist in 2010 and was inducted as an ACM Fellow in 2012. He has also been inducted as a Fellow of IEEE in 2012. His current interests are in the area of scalable data management and data analysis in Cloud Computing environments, security and privacy of data in the cloud, and scalable analytics over social networks data and social media.

Solving Problems with Visual Analytics: The Role of Visualization and Analytics in Exploring Big Data

Prof. Dr. Daniel A. Keim

Department of Computer and Information Science
Konstanz University
78457 Konstanz, Germany
keim@uni-konstanz.de

Abstract

Never before in history data is generated and collected at such high volumes as it is today. As the volumes of data available to business people, scientists, and the public increase, their effective use becomes more challenging. Keeping up to date with the flood of data, using standard tools for data analysis and exploration, is fraught with difficulty. Visual analytics seeks to provide people with better and more effective ways to understand and analyze large datasets, while also enabling them to act upon their findings immediately. Visual analytics integrates the analytic capabilities of the computer and the abilities of the human analyst, allowing novel discoveries and empowering individuals to take control of the analytical process. Visual analytics enables unexpected and hidden insights, which may lead to beneficial and profitable innovation. In the visual analysis process, it is not obvious what can be done by automated analysis and what should be done by interactive visual methods. In dealing with massive data, the use of automated methods is mandatory - and for some problems it may be sufficient to only use fully automated analysis methods, but there is also a wide range of problems where the use of interactive visual methods is necessary. The talk presents the challenges of visual analytics and exemplifies them with several application examples, illustrating the exiting potential of current visual analysis techniques but also their limitations.

Biography: Daniel A. Keim is full professor and head of the Information Visualization and Data Analysis Research Group in the Computer Science Department of the University of Konstanz, Germany. He has been actively involved in data base, data analysis, and information visualization research for about 20 years and developed a number of novel visual analysis techniques for very large data sets. He has been program co-chair of the IEEE InfoVis and IEEE VAST symposia as well as the SIGKDD conference, and he is member of the IEEE InfoVis & IEEE VAST as well as EuroVis steering committees. He is an associate editor of Palgrave's Information Visualization Journal (since 2001) and has been an associate editor of the IEEE Transactions on Visualization and Computer Graphics (1999 - 2004), Datenbank-Spektrum (2011 - 2009), the IEEE Transactions on Knowledge and Data Engineering (2002 - 2007), and the Knowledge and Information System Journal (2006 - 2011). He is coordinator of

the DFG German Strategic Research Initiative (SPP) ”Scalable Visual Analytics”, the BMBF research initiative on ”Visual Analytics for Security Applications2 (VASA), and he has been the scientific coordinator of the EU Coordination Action ”Visual Analytics - Mastering the Information Age” (VisMaster). Dr. Keim got his Ph.D. and habilitation degrees in computer science from the University of Munich. Before joining the University of Konstanz, Dr. Keim was associate professor at the University of Halle, Germany and Technology Consultant at AT&T Shannon Research Labs, NJ, USA.

Evolving the Architecture of a DBMS for Modern Hardware

Paul Larson

Microsoft Research
Redmond, WA 98052-6399
Paul.Larson@microsoft.com

Abstract

The major commercial database systems were designed primarily for OLTP workloads and under the assumption that processors are slow, memory is scarce, and data lives on disk. These assumption are no longer valid: OLAP workloads are now as common as OLTP workloads, multi-core processors are the norm, large memories are affordable, and frequently accessed data lives mostly in the main memory buffer pool. So how can a vendor with a mature DBMS product exploit the opportunities offered by these changes? Rewriting from scratch is not realistic - it is way too expensive and risky. The only realistic option is to gradually evolve the architecture of the system. SQL Server has begun this journey by adding two features: column store indexes to speed up OLAP-type queries, and Hekaton, a new engine optimized for large memories and multicore processors. The talk will outline the design of these features, the main goals and constraints, and discuss the reasoning behind the design choices made.

Biography: Paul (Per-Ake) Larson has conducted research in the database filed for over 30 years. He served as a Professor in the Department of Computer Science at the University of Waterloo for 15 years and joined Microsoft Research in 1996 where he is a Principal Researcher. Paul has worked in a variety of areas: file structures, materialized views, query processing, and query optimization among others. During the last few years he has collaborated closely with the SQL Server team on how to evolve the architecture of the core database system.

A Mutual Pruning Approach for RkNN Join Processing

Tobias Emrich, Peer Kröger, Johannes Niedermayer, Matthias Renz, Andreas Züfle

Institute for Informatics, Ludwig-Maximilians-Universität München
Oettingenstr. 67, D-80538 München, Germany
{emrich,kroeger,niedermayer,renz,zuefle}@dbs.ifi.lmu.de

Abstract: A reverse k-nearest neighbour (RkNN) query determines the objects from a database that have the query as one of their k-nearest neighbors. Processing such a query has received plenty of attention in research. However, the effect of running multiple RkNN queries at once (join) or within a short time interval (bulk/group query) has, to the best of our knowledge, not been addressed so far. In this paper, we analyze RkNN joins and discuss possible solutions for solving this problem. During our performance analysis we provide evaluation results showing the IO and CPU performance of the compared algorithms for a variety of different setups.

1 Introduction

A Reverse k-Nearest Neighbor (RkNN) query retrieves all objects from a database having a given query object as one of their k nearest neighbors. Various algorithms for efficient RkNN query processing have been studied under different conditions due to the query's relevance in a wide variety of domains — applications include decision support, profile-based marketing and similarity updates in spatial and multimedia databases.

Let us now shortly recap the definition of RkNN queries. Given a finite multidimensional data set $S \subset \mathbb{R}^d$ ($s_i \in \mathbb{R}^d$), a query point $r \in \mathbb{R}^d$, and an arbitrary distance function $dist(x, y)$ (e.g. the Euclidean distance), a k-nearest neighbor (kNN) query returns the k nearest neighbors of r in S:

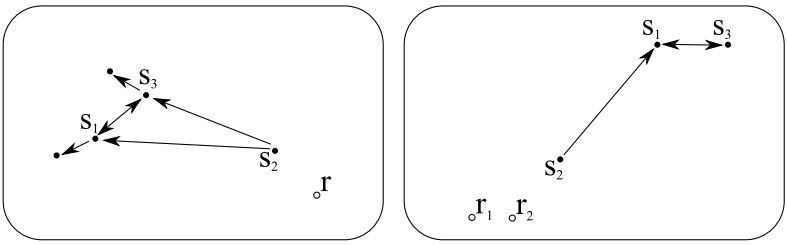
$$kNN(r, S) = \{s \in S : |\{s' \in S : dist(s', r) < dist(s, r)\}| < k\}$$

A monochromatic RkNN query, where r and $s \in S$ have the same type, can be defined by employing the kNN query:

$$RkNN(r, S) = \{s \in S | r \in (k+1)NN(s, S \cup \{r\})\}$$

Thus, an RkNN query returns all points $s_i \in S$ that would have r as one of its nearest neighbors. In Figure 1(a) an R2NN query is shown. Arrows denote a subset of the 2NN relationships between points from S. Since r is closer to s_2 than its 2NN s_1, the result set of an R2NN query with query point r is $\{s_2\}$. s_3 is not a result of the query since its 2NN s_2 is closer than r. Note that the RkNN query is not symmetric, i.e. the kNN result kNN(r,S) $\neq$ RkNN(r, S), because the 2NN of r are s_2 and s_3. Therefore the result of an RkNN(r,S) query cannot be directly inferred from the result of a kNN query kNN(r,S).

Besides the monochromatic RkNN query, research often discusses the bichromatic RkNN query. However, in this paper, we will concentrate on the monochromatic case and will



(a) R2NN Query (b) R1NN Join

Figure 1: R2NN Query and R1NN Join.

therefore just shortly introduce this second variant of the RkNN query. In the bichromatic case, two sets R_1 and R_2 are given. The goal is to compute all points in R_2 for which a query point $r \in R_1$ is one of the k closest points from R_1 [WYCT08]:

$$BRkNN(r, R_1, R_2) = \{s \in R_2 | r \in kNN(s, R_1)\}$$

An important problem in database environments is the scenario where the query does not consist of a single point but instead of a whole set of points, for each of which a RkNN query has to be performed. This setting is often referred to as *group query*, *bulk query* or simply *join* of two sets R and S. Despite the potential applications, the join operation has so far only received little attention in the context of RkNN queries. Given two sets R and S, the goal of a monochromatic RkNN join is to compute, for each point $r \in R$ its monochromatic RkNNs in S.

Definition 1 (Monochromatic RkNN join) *Given two finite sets $S \subset \mathbb{R}^d$ and $R \subset \mathbb{R}^d$, the monochromatic RkNN join $R \overset{MRkNN}{\ltimes} S$ returns a set of pairs containing for each $r \in R$ its RkNN from S:$R \overset{MRkNN}{\ltimes} S = \{(r, s) | r \in R \wedge s \in S \wedge s \in RkNN(r, S)\}$*

An example for $k = 1$ can be found in Figure 1(b). The result for both objects from R in this example is R1NN(r_1) = R1NN(r_2) = $\{s_2\}$, i.e. $R \overset{\text{MRkNN}}{\ltimes} S = \{(r_1, s_2), (r_2, s_2)\}$. Note that the elements r_1 and r_2 from R do not influence each other, i.e., r_1 cannot be a result object of r_2 and vice versa. This follows directly from the definition of the MRkNN join.

In this paper we discuss two solutions for solving RkNN joins. The first solution simply involves the iterative execution of an existing algorithm, while for the second solution we introduce an algorithm specialized for RkNN joins. The resulting algorithms are evaluated in an experimental section under a variety of different setups, including both synthetic and real data sets.

The remainder of this paper is organized as follows. Section 2 gives an insight into related work. In Section 3 we propose an RkNN join algorithm that is based on an existing mutual pruning algorithm. An extensive performance comparison of our solution follows in Section 4. Section 5 concludes this work.

2 Related Work

The problem of efficiently supporting RkNN queries has been studied extensively in the past years. Existing approaches for Euclidean RkNN search can be classified as self pruning approaches or mutual pruning approaches. *Self pruning approaches* [KM00, YL01, ABK$^+$06b, TYM06] are usually designed on top of a hierarchically organized tree-like index structure. They try to conservatively/exactly estimate the kNN distance of each index entry e. If this estimate is smaller than the distance of e to the query q, then e can be pruned. Thereby, self pruning approaches do not usually consider other entries (database points or index nodes) in order to estimate the kNN distance of an entry e, but simply precompute kNN distances of database points and propagate these distances to higher level index nodes.

Mutual pruning approaches such as [SAA00, SFT03, TPL04] use other points to prune a given index entry e. The most general and efficient approach called TPL is presented in [TPL04]. We will employ this approach as a benchmark algorithm during our performance evaluation.

The approach of combining self- and mutual pruning has been followed in [AKK$^+$09, KKR$^+$09b]. It obtains conservative and progressive distance approximations between a query point and arbitrarily approximated regions of a metric index structure.

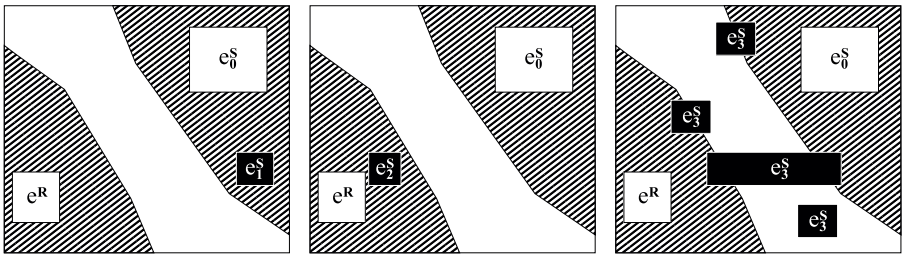
Beside solutions for Euclidean data, solutions for general metric spaces (e.g. [ABK$^+$06b, ABK$^+$06a, TYM06]) usually implement a self pruning approach.

Furthermore, there exist approximate solutions for the RkNN query problem that aim at reducing the query execution time for the cost of accuracy [SFT03, XHL$^+$05].

Besides the attention paid to single RkNN queries, the problem of performing multiple RkNN queries at a time, i.e. a RkNN join, has hardly been addressed. The authors of [YZHX10] addressed incremental bichromatic RkNN joins as a by-product of incremental kNN joins, aiming at maintaining a result set over time instead of performing bulk evaluation of large sets. Since it does not address the problem of a monochromatic join, it solves a different problem.

3 The Mutual Pruning Algorithm

Mutual pruning approaches such as TPL [TPL04] are state-of-the-art solutions for single RkNN queries. In this paper we aim at analyzing whether this assumption still holds for an RkNN join setting. Therefore, in this section, we propose an algorithm for processing RkNN joins based on a mutual pruning strategy similar to TPL. We assume that both sets R and S are indexed by an aggregated hierarchical tree-like access structure such as the aR^*-tree [PKZT01]. An aR^*-Tree is equivalent to an R^*-Tree but stores an additional integer value (often called weight) within each entry, corresponding to the number of objects contained in the subtree. The indexes are denoted by $\mathcal{R}$ and $\mathcal{S}$, respectively.



(a) Case 1 (b) Case 2 (c) Case 3

Figure 2: Mutual pruning on directory entries.

3.1 General Idea

The proposed algorithm is based on a solution for Ranking-RkNN queries, initially suggested in [KKR$^+$09a]. Unlike TPL, which can only use leaf entries (points) to prune other leaf entries and intermediate entries (MBRs), the technique of [KKR$^+$09a] further permits to use intermediate entries for pruning, thus, allowing to prune entries while traversing the tree, without having to wait for k leaf entries to be refined first. The algorithm of [KKR$^+$09a] uses the MAXDIST-MINDIST-approach as a simple method for mutual pruning using rectangles. This approach exploits that, for three rectangles R, A, B, it holds that A must be closer to R than B, if $maxDist(A, R) < minDist(B, R)$. The algorithm that we use in this work, will augment the algorithm of [KKR$^+$09a] by replacing the MAXDIST-MINDIST-approach by the spatial pruning approach proposed in [EKK$^+$10] which is known to be more selective. In the following, the base algorithm of [KKR$^+$09a], enhanced by [EKK$^+$10] will be extended to process joins.

The mutual pruning approach introduced in this section is based on an idea which is often used for efficient spatial join processing: Both indexes $\mathcal{R}$ and $\mathcal{S}$ are traversed in parallel, result candidates for points $r \in R$ of the outer set are collected and for each point $r \in R$ irrelevant subtrees of the inner index $\mathcal{S}$ are pruned; we will evaluate if this approach is also useful for RkNN joins during performance analysis. Thus, at some point of traversing both trees, we will need to identify pairs of entries ($e^R \in \mathcal{R}$, $e^S \in \mathcal{S}$) for which we can already decide, that for any pair of points ($r \in e^R$, $s \in e^S$) it must/must not hold that s is a RkNN of r. To make this decision without accessing the exact positions of children of e^R and e^S, we will use the concept of spatial domination ([EKK$^+$10]): If an entry e^R is (spatially) dominated by at least k entries in $\mathcal{S}$ with respect to e^S, then no point in e^S can possibly have any point of e^R as one of its k nearest neighbors. Due to the spatial extend of MBRs, this decision is not always definite. We have to distinct several cases, as illustrated in Figure 2. The subfigures visualize two pages e^R and e_0^S, and one of the additional pages e_1^S, e_2^S, e_3^S. The striped areas in the picture denote the set of points on which a closer decision can definitely be made. This means, no matter which points from the rectangle e^R and e_0^S are chosen, a point in the striped are is always closer to the point from e^R (or e_0^S) than to the point from e_0^S (or e^R). Therefore, in the first case, e_1^S is definitely closer to e_0^S than to e^R. In the second case, e_2^S is definitely closer to e^R than to e_0^S. In the third case, in all of the four subcases, no decision can be made.

More formally, in the first case, we can decide that an entry is (spatially) dominated by another entry. For example, in Figure 2(a), entry e^R is dominated by entry e_1^S with respect to entry e_0^S, since for all possible triples of points $(s_0 \in e_0^S, s_1 \in e_1^S, r \in e^R)$ it holds that s_1 must be closer to s_0 than r. This domination relation can be used to prune e_0^S: If the number of objects contained in e_1^S is at least k, then we can safely conclude that at least k objects must be closer to any point in e_0^S, and, thus, e_0^S and all its child entries can be pruned. To efficiently decide if an entry e_1^S dominates an entry e^R with respect to an entry e_0^S (all entries can be points or rectangles), we utilize the decision criterion $Dom(e_1^S, e^R, e_0^S)$ proposed in [EKK+10] which prevents us from doing a costly materialization of the pruning regions like the striped areas in Figure 2. Materialization here means the exact polygonal computation of the areas that allow pruning a page.

In the second case, we can decide that neither an entry, nor its children can possibly be pruned by another entry. In Figure 2(b), consider entry e_2^S. It holds that for any triple of points $(s_0 \in e_0^S, s_2 \in e_2^S, r \in e^R)$, that s_2 cannot be closer to s_0 than r. Although, in this case, we cannot prune e_0^S, we can safely avoid further domination tests of children of the tested entries. We can efficiently perform this test by evaluating the aforementioned criterion $Dom(e^R, e_2^S, e_0^S)$.

Finally, in the third case, both predicates $Dom(e_3^S, e^R, e_0^S)$ and $Dom(e^R, e_3^S, e_0^S)$ do not hold for any entry e_3^S in Figure 2(c). In this case, some points in e_3^S may be closer to some points e_0^S than some points in e^R, while other points may not. Thus, we have to refine at least some of the entries e_0^S, e_3^S or e^R. The reason for the inability to make a decision here, is that the pruning region between two rectangles is not a single line, but a whole region (called *tube* here, cf. Figure 2). For objects that fall into the tube, no decision can be made.

At any time of the execution of the algorithm only one entry e^R of the outer set is considered. For e^R, we minimize the number of domination checks that have to be performed. Therefore, we keep track of pairs of entries in $\mathcal{S}$, for which case three holds, because only in this case, refinement of entries may allow to prune further result pairs. This is achieved by managing, for each entry $e^S \in \mathcal{S}$, two lists $e^S.update1 \subset \mathcal{S}$ and $e^S.update2 \subset \mathcal{S}$: List $e^S.update1$ contains the set of entries with respect to which e^S may dominate e^R but does not dominate e^R for sure. Essentially, any entry in $e^S.update1$ may be pruned if e^S is refined. List $e^S.update2$ contains the set of entries, which may dominate e^R with respect to e^S, but which do not dominate e^R for sure. Thus, $e^S.update2$ contains the set of entries, whose children may potentially cause e^S to be pruned.

3.2 The Algorithm $joinEntry$

In order to implement these ideas, we use the recursive function shown in Algorithm 1, $joinEntry(Entry\, e^R, Queue\, Q^S)$. It receives an entry $e^R \in \mathcal{R}$ that represents the currently processed entry from the index of the outer set R, which can be a point, a leaf node containing several points, or an intermediate node. Q^S represents a set of entries from $\mathcal{S}$ sorted decreasingly in the number $|e^S.update1|$ of objects that an entry $e^S \in \mathcal{S}$ is able to prune. The reason is that resolving nodes with a large $update1$ list potentially allows pruning many other nodes.

Algorithm 1 joinEntry(Entry e^R, Queue Q^S)

1: **for all** $e_i^S \in Q^S$ **do**
2: {Update domination count (lower bound) of all e_i^S}
3: **for all** $e_j^S \in e_i^S$.update2 **do**
4: **if** Dom(e_j^S,e^R, e_i^S) **then**
5: {definite decision possible, e_j^S prunes e_i^S}
6: e_i^S.dominationCount += e_j^S.weight
7: **else if** Dom(e^R,e_j^S, e_i^S) **then**
8: {e_i^S can definitely not be pruned by e_j^S}
9: e_i^S.update2.remove(e_j^S)
10: e_j^S.update1.remove(e_i^S)
11: **end if**
12: **end for**
13: **if** e_i^S.dominationCount $\geq k$ **then**
14: {no point in e_i^S can be an RkNN of a point in e^R}
15: delete(Q^S, e_i^S))
16: **end if**
17: **end for**
18: {in the following, resolve $\mathcal{S}$}
19: Queue $Q_c^S = \emptyset$
20: **while** (e_i^S = Q^S.poll()) $\neq$ NULL **do**
21: Go to line 20 if e_i^S.dominationCount $\geq k$ {e_i^S does not contain result candidates}
22: **if** Vol(e_i^S) $>$ Vol(e^R) **then**
23: {go one level down in the subtree of e_i^S and add child pages to Q^S}
24: Q^S.add(resolve(e_i^S, e^R))
25: **else if** isLeaf(e_i^S) $\wedge$ isLeaf(e^R) **then**
26: {if no further refinement is possible, results still have to be verified}
27: **if** $e^R \in kNN(e_i^S)$ **then**
28: reportResult($< e^R, e_i^S >$)
29: **end if**
30: **else**
31: {put pages e_i^S into Q_c^S if they could neither be pruned nor reported as result}
32: Q_c^S.add(e_i^S)
33: **end if**
34: **end while**
35: {in the following, resolve e^R}
36: **if** $\neg$isLeaf(e^R) **then**
37: {finally, refine e_i^R by recursively calling $joinEntry$ with Q_c^S}
38: **for all** $e_i^R \in e^R$.children **do**
39: joinEntry(e_i^R, clone(Q_c^S))
40: **end for**
41: **end if**

Algorithm 2 resolve(Entry e^S, Entry e^R)

```
1:  LIST l
2:  {(1) check which objects the children e_i^S of e^S may affect}
3:  for all e_j^S ∈ e^S.update1 do
4:     e_j^S.update2.remove(e^S) {remove, children of e^S are now relevant instead of e^S}
5:     for all e_i^S ∈ e^S.children do
6:        if Dom(e_i^S, e^R, e_j^S) then
7:           {definite decision possible, e_i^S prunes e_j^S}
8:           e_j^S.dominationCount += e_i^S.weight
9:        else if ¬ Dom(e^R, e_i^S, e_j^S) then
10:          {no definite decision possible, e_i^S might prune e_j^S}
11:          e_j^S.update2.add(e_i^S)
12:          e_i^S.update1.add(e_j^S)
13:       end if
14:    end for
15: end for
16: {(2) check which other entries may affect a child e_i^S}
17: for all e_i^S ∈ e^S.children do
18:    for all e_j^S ∈ e^S.update2 do
19:       if Dom(e_j^S, e^R, e_i^S) then
20:          {definite decision possible, e_j^S prunes e_i^S}
21:          e_i^S.dominationCount += e_j^S.weight
22:       else if ¬ Dom(e^R, e_j^S, e_i^S) then
23:          {no definite decision possible, e_j^S might prune e_i^S}
24:          e_i^S.update2.add(e_j^S)
25:          e_j^S.update1.add(e_i^S)
26:       end if
27:    end for
28:    if e_i^S.dominationCount < k then
29:       {only return relevant entries that can not be pruned, yet}
30:       l.add(e_i^S)
31:    end if
32: end for
33: return l
```

In each call of $joinEntry()$, a lower bound of the number of objects dominating e^R with respect to e_i^S is updated for each entry $e_i^S \in Q^S$. This lower bound is denoted as *domination count*. Clearly, if for any entry e_i^S, it holds that the domination count $\geq k$, then the pair $< e^R, e_i^S >$ can be safely pruned. Note that using the notion of domination count, the list $e_i^S.update1$ can be interpreted as the list of entries e_j^S, for which the domination count of e_j^S may be increased by refinement of e_i^S. The list $e_i^S.update2$ can be interpreted as the list of entries whose refinement may increase the domination count of e_i^S. In Line 4

of Algorithm 1, the domination count of e_i^S is updated by calling $Dom(e_j^S, e_R, e_i^S)$ for each entry e_j^S in the list $e_i^S.update2$. If $Dom(e_j^S, e_R, e_i^S)$ holds, then the domination count of e_i^S is increased by the number of objects in e_j^S. The number of leaf entries is stored in each intermediate entry of the index. Otherwise, i.e., if e_j^S does not dominate e^R w.r.t. e_i^S, we check if it is still possible that any point in e_j^S dominates points in e^R with respect to any point in e_i^S. If that is not the case, then e_j^S is removed from the list of $e_i^S.update2$, and e_i^S is removed from the list of entries $e_j^S.update1$ (Lines 9-10). If these checks have increased the domination count of e_i^S to k or more, we can safely prune e_i^S in Line 15 and remove all its references from the $update1$ lists of other entries; this is achieved by the delete function.

Now that we have updated domination count values of all $e_i^S \in Q^S$, we start our refinement round in Line 20. Here, we have to decide which entry to refine. We can refine the outer entry e^R, or we can refine some, or all entries in the queue of inner entries Q^S. A heuristics that has shown good results in practice, is to try to keep, at each stage of the algorithm, both inner and outer entries at about the same volume. Using this heuristics, we first refine all inner entries $e_i^S \in Q^S$ which have a larger volume than the outer entry e^R in line 24. The corresponding algorithm is introduced in the next section.

After refining entries e_i^S, we next check in Line 25 if both inner entry e_i^S and outer entry e^R currently considered are both point entries. If that is the case, clearly, neither entry can be further refined, and we perform a kNN query using e_i^S as query object to decide whether e^R is a kNN of e_i^S, and, if so, return the pair e^R, e_i^S as a result. Finally, all entries e_i^S which could neither be pruned nor returned as a result, are stored in a new queue Q_C^S. This queue is then used to refine the outer entry e^R: For each child of e^R, the algorithm $joinEntry$ is called recursively, using Q_C^S as inner queue.

3.3 Refinement: The $resolve$-Routine

Our algorithm for refinement of an inner entry e^S is shown in Algorithm 2 and works as follows: We first consider the set of entries $e^S.update1$ of other inner entries e_j^S whose domination count may be increased by children e_i^S of e^S. For each of these entries, we first remove e^S from its list $e_j^S.update2$, since e^S will be replaced by its children later on. Although e^S does not dominate e^R w.r.t. e_j^S, the children of e^S may do. Thus, for each child e_i^S of e^S, we now test if e_i^S dominates e^R w.r.t. e_j^S in Line 6 of Algorithm 2. If this is the case, then the domination count of e_j^S is incremented according to the number of objects in e_i^S.[1] Otherwise, we check if it is possible for e_i^S to dominate e^R w.r.t. e_j^S, and, if that is the case, then e_j^S is added to the list $e_i^S.update1$ of entries which e_i^S may affect, and e_i^S is added to the list $e_j^S.update2$ of entries which may affect e_j^S. Now that we have checked which objects the children e_i^S of e^S may affect, we next check which other entries may affect a child e_i^S. Thus, we check the list $e^S.update2$ of entries which may affect the domination count of e^S. For each such entry e_j^S and for each child e_i^S, we check

[1] The check, whether the new domination count of e_j^S exceeds k will be performed in Line 21 of Algorithm 1

if e_j^S dominates e^R w.r.t. e_i^S. If that is the case, the domination count of e_i^S is adjusted accordingly. Otherwise, if e_j^S can possibly dominate e^R w.r.t. e_i^S, then we add e_j^S to the list of entries $e_i^S.update2$, and we add e_i^S to the list $e_j^S.update1$. Finally, all child entries of e^S are returned, except those child entries, for which their corresponding domination count already reaches k.

4 Experiments

We evaluate our mutual pruning approach using update lists (referred to as UL) in comparison to the state-of-the-art single RkNN query processor TPL in an RkNN join setting within the Java-based KDD-framework ELKI[AGK$^+$12] on both synthetic and real data sets. We use the synthetic data to show the behaviour of the different algorithms in a well-defined setting. Additionally, we use the real data set to show the behaviour of the different algorithms on a not normally distributed data set with dense clusters and additional noise. As performance indicators we chose the CPU time and the number of page accesses.

For measuring the number of page accesses, we assumed that a given number of pages fit into a dedicated cache. If a page has to be accessed but is not contained in the page cache, it has to be reloaded. If the cache is already full and a new page has to be loaded, an old page is kicked out in LRU manner. The page cache only manages data pages from secondary storage, remaining data structures have to be stored in main memory.

Concerning the nomenclature of the algorithms we use the following notation. UL is the mutual pruning based algorithm from Section 3. The additional subscript S (Single) means that every *single* point of R was queried on its own. With UL_G (Group), a whole set of points, a leaf page, was queried at once. UL_P (Parallel) traversed both indexes for R and S in parallel. These three versions can be easily derived from Algorithm 1 in Section 3. The algorithm expects an entry of R's index. In our performance analysis we call the algorithm with leaf entries (leading to UL_S), the entries pointing to leaf nodes (leading to UL_G) and the root entry of R's index (leading to UL_P). This is especially of interest for large data sets, since UL_G and UL_S allow to split the join up to process it on several distributed systems, increasing its applicability for distributed databases.

TPL was implemented as suggested in [TPL04], however we replaced the clipping step and instead implemented the decision criterion from [EKK$^+$10] to enable cheap pruning on intermediate levels of the indexes.

As an index structure for querying we used an aggregated R*-tree (aR*-Tree [PKZT01]). The page size was set to 1024 bytes, the cache size to 32768 bytes.

4.1 Experiments on Synthetic Data

We chose the underlying synthetic data sets from R and S, which have been created with the ELKI-internal data generator, to be normally distributed with equivalent mean and a standard deviation of 0.15. We set the default size of R to $|R| = 0.01|S|$, since the performance of both algorithms degenerates with increasing $|R|$. For each of the analyzed

algorithms we used exactly the same data set given a specific set of input variables in order to reduce skewed results.

During performance analysis, we analyzed the impact of k, the number of data points in R and S, the dimensionality d, and the mean difference $\Delta\mu$ between the data sets R and S on the performance of the evaluated algorithms keeping all but one variable at a fixed default value while varying a single independent variable. Input values for each of the analyzed independent variables can be found in Table 1. In the table, bold values denote default values that are used whenever a different variable is evaluated.

Variable	Values	Unit
k	5, **10**, 100, 500	points
$\|R\|$	10, **100**, 1000, 10000, 20000, 40000	points
$\|S\|$	10, 1000, **10000**, 20000, 40000,80000	points
$\Delta\mu$	**0.0**, 0.2, 0.4	$\|\mu_S - \mu_R\|$
d	**2**, 3, 4	dimensions

Table 1: Values for the evaluated independent variables. Default values are denoted in bold.

Varying k. In a first series of experiments, we varied the parameter k. Note that both mutual pruning approaches, TPL and our UL approach are mainly applicable to low values for k, especially concerning the execution time (cf. Figure 3 (a)). The runtime of TPL increases considerably fast. The reason for this is that not only the number of result candidates but also the number of objects which are necessary in order to confirm (or prune) these candidates increase superlinear in k. In contrast, the runtime of the UL algorithms degenerates slower compared to TPL. The main problem with this family of algorithms is their use of update lists. Each time a page is resolved, the corresponding update lists have to be partially recomputed. This leads to an increase of cost with larger k since on the one hand side, more pages have to be resolved, and on the other hand the length of the update lists of an entry increases and therefore more distance calculations are necessary. Note that UL_G and UL_P perform very similar to UL_S, which is an interesting observation, since for kNN joins parallel tree traversals usually show a higher gain in performance than in an RkNN setting. Concerning the number of page accesses, the picture is quite similar (cf. Figure 4 (a)). TPL shows a performance worse than UL.

Varying the Size of R ($|R|$). Varying $|R|$ shows a negative effect on both approaches TPL and UL — their computational time increases considerably fast (cf. Figure 3 (b)). For UL_S and TPL the increase of CPU time is linear since these algorithms perform a single RkNN query for each point in R. For larger $|R|$, the remaining approaches UL_G and UL_P show a better performance, since these algorithms traverse the tree less often. Interestingly, the number of page access (cf. Figure 4 (b)) for all UL approaches is similar, but always better than for TPL. We explain the large difference in page accesses by the different pruning approaches used by TPL and UL. TPL only employes candidate points for pruning pages, while the UL approaches can also take not yet resolved pages to prune. This can lead to a significant reduction in the number of page accesses.

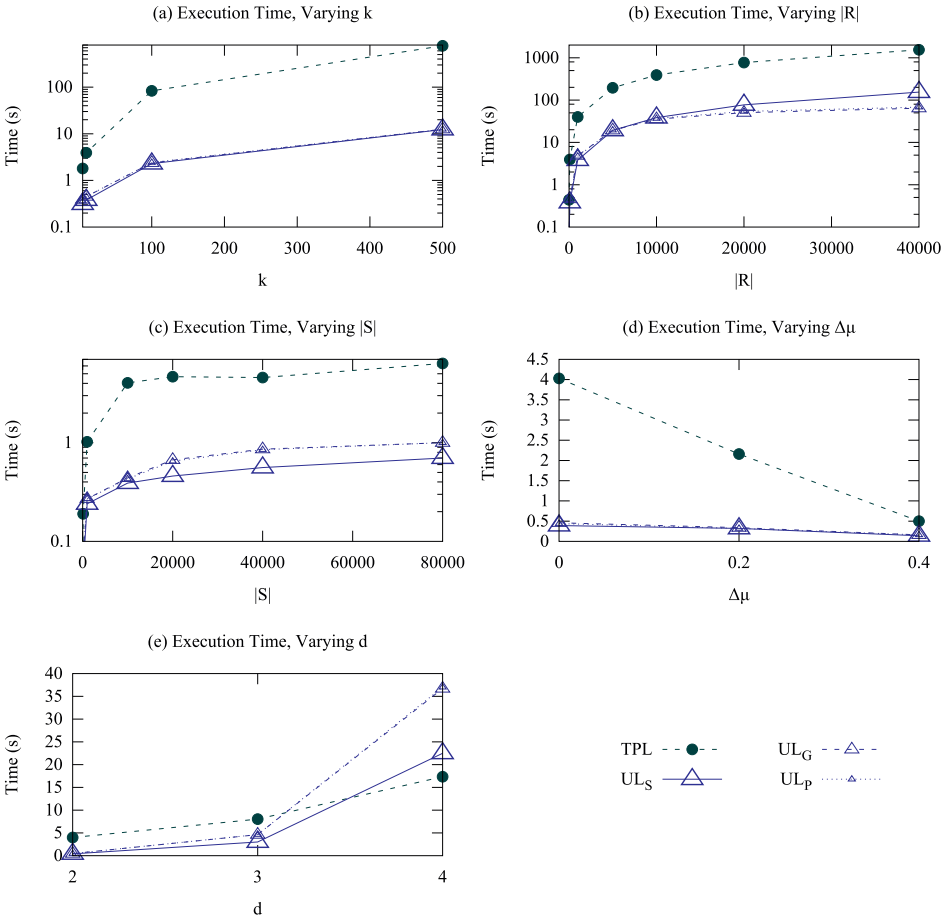


Figure 3: Performance (Execution Time), synthetic data set.

Varying the Size of S ($|S|$). Next we analyzed the effect of different values for $|S|$ regarding the CPU time (cf. Figure 3 (c)). Again, the UL approaches perform best, more precisely UL_S since this approach enables highest pruning power. Taking a look at the number of disk accesses (cf. Figure 4 (c)), the results look very similar, however the higher pruning power of UL_S does not show any effect here.

Varying the Overlap Between R and S ($\Delta\mu$). Until now we assumed that the normally distributed sets of values R and S overlap completely, i.e. both sets have the same mean. This assumption is quite intuitive for example if we assume that R and S are drawn from the same distribution. However, if for example R contains feature vectors of a set of dog pictures and S describes mostly flowers, the feature vectors from R and S should be

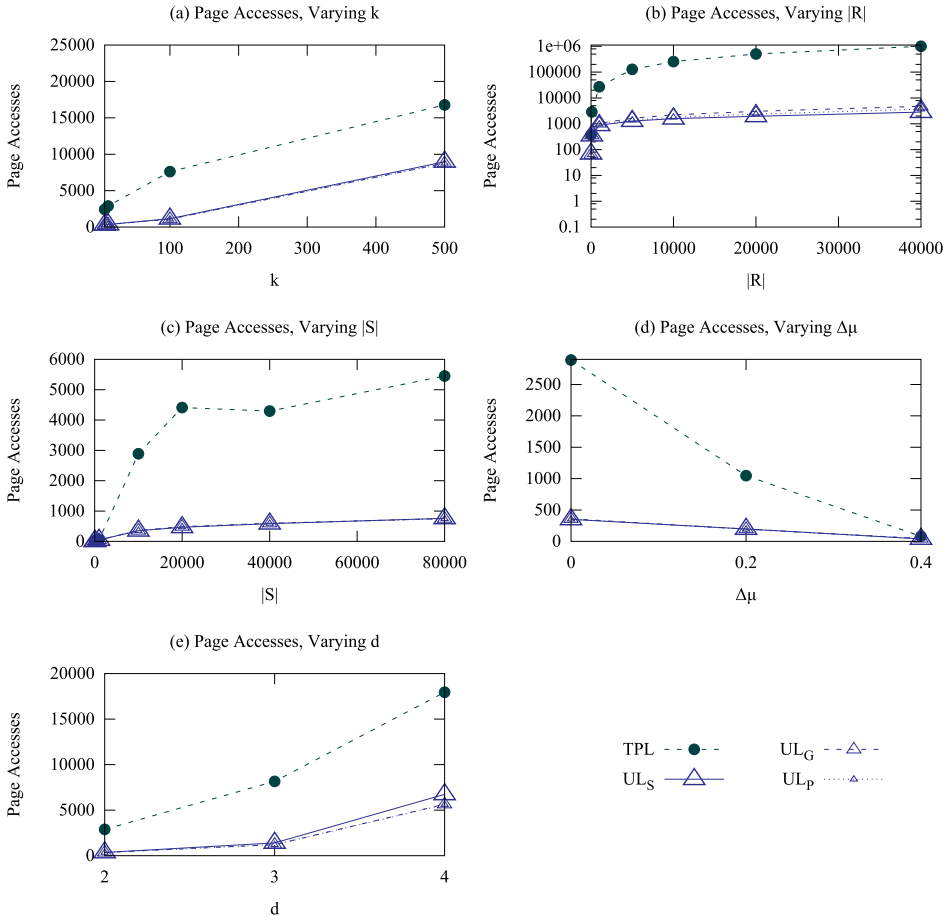


Figure 4: Performance (Page Accesses), synthetic data set.

located at different positions in feature space. We model this behaviour by decreasing the overlap of the two sets R and S and therefore increasing their mean difference ($\Delta\mu = \mu_R - \mu_S$).

Both approaches, UL and TPL can take quite some profit from lower overlap between the sets R and S. All of them employ pruning to avoid descending into subtrees that do not have to be taken into account to answer the query. If the overlap decreases, subtrees can be pruned earlier (because the MINDIST between a subtree and the query point increases), greatly reducing the CPU-time and number of page accesses (cf Figures 3 (d) and 4 (d)). Note that for TPL this gain is slightly higher, however even for a mean difference of 0.4, the UL approaches perform better than TPL.

Figure 5: A sample of 5000 points from the postoffice data set.

Varying the Dimensionality (d). Taking a look at the performance of the different algorithms with varying dimensionality offers other interesting results (cf. Figure 3 (e) and 4 (e)).

With a dimensionality of 2 and 3, the most important ones for spatial query processing, the UL approaches perform better than TPL concerning the execution time of the algorithms. For two dimensions the gain in performance reaches a factor of 8, for three dimensions still a factor of about 2.6. Beginning with a dimensionality of 4, the UL approaches scale worse than the other approaches concerning execution time, because the pruning power of index-level pruning decreases with increasing dimensionality. With increasing d, the number of entries in an update list increases exponentially. Therefore, much more entries have to be checked each time an intermediate node is resolved, leading to a significant drop in performance.

The results in terms of the number of disk accesses look very similar, therefore they shall not be further investigated. However note that the UL approaches show much better performance in terms of the number of disk accesses than TPL, since they employ pruning on an index level.

4.2 Experiments on Real Data: Postoffice Data Set

Now let us take a look at experiments driven with real data. As a real data set we employed a set of 123593 post offices in the north-eastern united states.[2] The set is clustered (and therefore correlated) in the metropolitan areas, containing further noise in the rural areas, as it can be seen in the visualization of the data set in Figure 5, containing 5000 sample points. Boths sets R and S are taken from the data set by assigning each of the 123593 points to one of the sets R or S, respectively. To take full advantage of the whole data set size of 123593 points, we decided to vary the sizes of R and S simultaneously such

[2] www.rtreeportal.org/

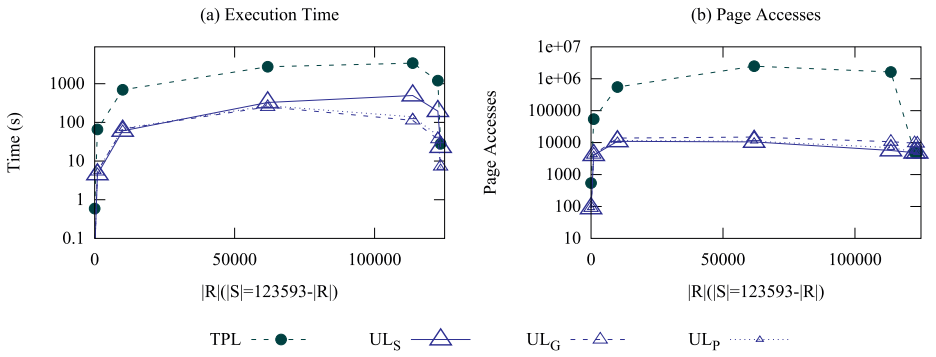


Figure 6: Performance (CPU time, page accesses), real data set (Postoffice).

that $|R| + |S| = 123593$. Clearly, the UL algorithms outperform TPL on this data set (cf. Figure 6). Note that both approaches, TPL and UL, perform better if R is small and S is large than if S is small and R is large. The explanation for this behaviour becomes most clear when taking a look at TPL: Here, the size of S has a lower influence on the performance of the algorithm, because often a larger set S just allows pruning more points. In contrast, increasing the size of R introduces more RkNN queries, which is expensive. This problem however, can be mitigated by using UL_P or UL_G, since these approaches perform index-level pruning with whole sets of points from R.

5 Conclusions

In this paper, we addressed the problem of running multiple RkNN-queries at a time, a.k.a RkNN join. For this purpose, we proposed a dedicated algorithm for RkNN join queries based on the well-known mutual pruning paradigm and evaluated it in a variety of settings including synthetic and real data sets.

However, our research is still preliminary and there is great space for improvements. For example, we would like to develop algorithms specialized for higher dimensionality, since all evaluated algorithms significantly drop in performance for a high number of dimensions. To achieve this, we would like to develop algorithms based on the self pruning paradigm and compare these to the developed mutual pruning approaches.

Acknowledgements. Part of this work was supported by the Deutsche Forschungsgemeinschaft (DFG) under grant number KR 3358/4-1.

References

[ABK+06a] E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Approximate Reverse k-Nearest Neighbor Queries in General Metric Spaces. In *Proc. CIKM*, 2006.

[ABK+06b] E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Efficient Reverse k-Nearest Neighbor Search in Arbitrary Metric Spaces. In *Proc. SIGMOD*, 2006.

[AGK+12] Elke Achtert, Sascha Goldhofer, Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. Evaluation of Clusterings - Metrics and Visual Support. In *ICDE*, pages 1285–1288, 2012.

[AKK+09] E. Achtert, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Reverse k-nearest neighbor search in dynamic and general metric databases. In *Proc. EDBT*, 2009.

[EKK+10] T. Emrich, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle. Boosting Spatial Pruning: On Optimal Pruning of MBRs. In *Proc. SIGMOD*, 2010.

[KKR+09a] H.-P. Kriegel, P. Kröger, M. Renz, A. Züfle, and A. Katzdobler. Incremental Reverse Nearest Neighbor Ranking. In *Proc. ICDE*, 2009.

[KKR+09b] H.-P. Kriegel, P. Kröger, M. Renz, A. Züfle, and A. Katzdobler. Reverse k-Nearest Neighbor Search based on Aggregate Point Access Methods. In *Proc. SSDBM*, 2009.

[KM00] F. Korn and S. Muthukrishnan. Influenced Sets Based on Reverse Nearest Neighbor Queries. In *Proc. SIGMOD*, 2000.

[PKZT01] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient OLAP Operations in Spatial Data Warehouses. In *Proc. SSTD*, pages 443–459, 2001.

[SAA00] I. Stanoi, D. Agrawal, and A. E. Abbadi. Reverse Nearest Neighbor Queries for Dynamic Databases. In *Proc. DMKD*, 2000.

[SFT03] A. Singh, H. Ferhatosmanoglu, and A. S. Tosun. High Dimensional Reverse Nearest Neighbor Queries. In *Proc. CIKM*, 2003.

[TPL04] Y. Tao, D. Papadias, and X. Lian. Reverse kNN Search in Arbitrary Dimensionality. In *Proc. VLDB*, 2004.

[TYM06] Y. Tao, M. L. Yiu, and N. Mamoulis. Reverse Nearest Neighbor Search in Metric Spaces. *IEEE TKDE*, 18(9):1239–1252, 2006.

[WYCT08] W. Wu, F. Yang, C.-Y. Chan, and K.L. Tan. FINCH: Evaluating Reverse k-Nearest-Neighbor Queries on Location Data. In *Proc. VLDB*, 2008.

[XHL+05] C. Xia, W. Hsu, M. L. Lee, J. Joxan, C. Xia, and W. Hsu. ERkNN: efficient reverse k-nearest neighbors retrieval with local knn-distance estimation. In *Proc. CIKM*, 2005.

[YL01] C. Yang and K.-I. Lin. An index structure for efficient reverse nearest neighbor queries. In *Proc. ICDE*, 2001.

[YZHX10] Cui Yu, Rui Zhang, Yaochun Huang, and Hui Xiong. High-dimensional kNN joins with incremental updates. *Geoinformatica*, 14(1):55–82, 2010.

MR-DSJ: Distance-Based Self-Join for Large-Scale Vector Data Analysis with MapReduce

Thomas Seidl Sergej Fries Brigitte Boden

RWTH Aachen University, Germany
Data Management and Data Exploration Group
{seidl,fries,boden}@cs.rwth-aachen.de

Abstract: Data analytics gets faced with huge and tremendously increasing amounts of data for which MapReduce provides a very convenient and effective distributed programming model. Various algorithms already support massive data analysis on computer clusters but, in particular, distance-based similarity self-joins lack efficient solutions for large vector data sets though they are fundamental in many data mining tasks including clustering, near-duplicate detection or outlier analysis.

Our novel distance-based self-join algorithm for MapReduce, MR-DSJ, is based on grid partitioning and delivers correct, complete, and inherently duplicate-free results in a single iteration. Additionally we propose several filter techniques which reduce the runtime and communication of the MR-DSJ algorithm. Analytical and experimental evaluations demonstrate the superiority over other join algorithms for MapReduce.

1 Introduction

As an ongoing trend, tremendously increasing amounts of data are collected in real-world applications of life science, engineering, telecommunication, business transactions and many other domains. For the management and analysis of these data, many different techniques and algorithms have been developed, ranging from basic database operations to high-level data mining approaches like clustering, classification or the detection of outliers. Processing huge data sets with millions or billions of records on a single computer exceeds the computation capabilities of single computing nodes due to limitations of disk space and/or main memory. Thus, it is indispensable to develop distributed approaches that run on clusters of several computers in parallel [RU11].

An important group of database operations are *joins*. Similarity self-joins, which are a special type of joins, play an important role in data analysis: data cleaning [CGK06, RRS00], near duplicate detection [XWLY08, Mon00], document similarity analysis [BML10] and data mining tasks like density-based clustering like DBSCAN [EKSX96, BBBK00] inherently join the input data based on similarity relationships and, therefore, will draw high benefit from efficient and scalable implementations of similarity self-joins.

In this paper, we study the distributed computation of *distance-based similarity self-joins*. A distance-based join $R \bowtie_\varepsilon S = \{r \circ s \,|\, d(r.A, s.A) \leq \varepsilon\}$ returns all pairs of objects (r, s) whose distance in attribute A does not exceed a maximum dissimilarity threshold, ε, which

is called the *range* or the *radius* of the similarity join. In our applications, the domain is a multidimensional data space $\mathbb{R}^{dim}$, and distance measures include the L_p norms like the Euclidean distance L_2, Manhattan distance L_1 or the Maximum distance L_∞.

For the development of distributed query processing algorithms, a variety of structured programming models exists. Aside classic parallel programming, the MapReduce model was proposed by Google [DG04], and its open-source implementation Hadoop found wide-spread attention and usage.

In this paper, we study the computation of distance-based self-joins for vector data using the MapReduce programming model. Our proposed grid-based approach combines the advantages of a very simple implementation and at the same time high efficiency, especially in low- to medium-dimensional domains that often occur in for example density-based clustering. However, it can also be applied to high-dimensional data by performing a dimensionality reduction first (dimensionality reduction techniques for MapReduce are implemented in the Mahout framework[1]). Overall, the main contributions of this paper are the following:

- We propose the MR-DSJ algorithm which efficiently computes the distance-based self-join on vector data using MapReduce avoiding duplicate distance computations.
- We introduce efficient filtering techniques based on mindist approximations for reducing communication and computation costs.
- We show the effectiveness of the developed approach by formal proofs and the efficiency by experiments on synthetic and real-world datasets.

The remainder of this paper is organized as follows: Section 2 describes related work. In Section 3 our join algorithm MR-DSJ is introduced. Section 4 presents the experimental evaluation of our technique. Theoretical analyses and ideas for future work are provided in Section 5, and Section 6 concludes the paper.

2 Related Work

The join operator is a fundamental database operator and is important for a large set of database queries. A general θ-join of two (or more) relations R, S is defined as $R \bowtie_\theta S = \sigma_\theta (R \times S) = \{r \circ s \,|\, \theta(r, s)\}$. Depending on the used predicate θ, different kinds of joins like equi-, spatial-, or distance-join are defined. As an alternative to our distance-based formalization, similarity functions $sim : U \times U \to \mathbb{R}_0^+$ for some object domain U can be used which indicate a high similarity of objects by high values. Prominent examples are the set intersection measure or the cosine similarity [VCL10, BML10].

The simplest solution for the computation of a join is a nested loop over both relations, which, however, has the disadvantage of quadratic complexities for computation and I/O. These problems have lead to the development of advanced approaches [Dat06] including

[1] http://mahout.apache.org/

block nested loop, sort-merge, hash- or partition-based or index-based techniques which try to alleviate one of these or both problems.

Not every approach can, however, efficiently cope with similarity joins in multidimensional vector spaces. For example there is no natural sorting of points in a multidimensional space, such that sort-merge join techniques are probably not very appropriate solutions. The similarity join of very large data sets also has often to grapple with the problem of not indexed data. Due to the size, and - in case of distributed data storage - due to the distributed data location, the building of auxiliary index structures can be too expensive. The widespread solution for this problem is the usage of partition-based schemes which often can be performed in on-line fashion [ZJ03]. Appropriate data partitioning can lead to a significant efficiency gain of the join algorithm, which is achieved by pruning unnecessary distance calculation between partitions which are located too far away from each other. A prominent partitioning scheme is a (equal-sized) grid. While it does not require any data distribution information, it provides good results for not too skewed data and different approaches make use of it [BBKK01, PD96]. If additional information about the data is available, the partitioning can also address the data skewness problem [DNSS92].

In general, the parallelization of joins leads to a higher efficiency. In this work we investigate a solution for the similarity join in MapReduce. Let us briefly recall its programming model before we describe existing join approaches based on it. In MapReduce, the data is given as a list of records that are represented as (key, value) pairs. Basically, a MapReduce program consists of two phases: In the "Map" phase, the records are arbitrarily distributed to different computing nodes (called "mappers") and each record is processed separately, independent of the other data items. The map phase then outputs intermediate (key,value) pairs. In the "Reduce" phase, records having the same key are grouped together and processed in the same computing node ("reducer"). Thus, the reducers combine information of different records having the same key and aggregate the intermediate results of the mappers. The results are stored back to the distributed file system. A simple example for a MapReduce program for the word count problem is given in [DG04].

Join processing using the MapReduce framework has already found high attention. [BPE$^+$10] provides an overview of common join strategies in MapReduce. In [PPR$^+$09], MapReduce is compared with parallel DBMS. Recent extensions of Hadoop including HadoopDB [ABPA$^+$09], Hadoop++ [DQRJ$^+$10] or PACT [ABE$^+$10] also have a specific focus on join processing. However, the vast majority of existing work about parallel joins refers to equi-joins. Afrati and Ullmann [AU10] present optimization strategies for multiway equi-joins, but they do not approach similarity joins. Broadcasting join strategies (e.g. [BPE$^+$10]) rely on the assumption that the join partners significantly differ in their size ($|R| \ll |S|$), which does not apply for self-joins.

The field of similarity joins on MapReduce also gained a high attention in the last few years. The k-NN joins for Euclidian spaces were addressed in [LSCO12] and in [ZLJ12]. In the latter the author exploits the space-filling curves and transform the kNN joins into a sequence of one-dimensional range searches. In [LSCO12] a Voronoi based partitioning allows for an effective join. A technique for similarity joins in metric spaces was presented in [SRT12]. Both, [SRT12] and [LSCO12] are data partitioning approaches which require one or multiple runs on the data before the join algorithm can start. In this work we focus

on similarity joins on vector data and exploit properties of vector spaces using a grid-based approach to obtain an efficient join. The vector space representation and the choice of equi-sized grids allows for a simple single-iteration algorithm which is particularly well suited for low- to medium-dimensional data. Though metric space joins are potentially able to handle very high-dimensional data, we believe that a simple but fast method as we propose in this work is an enrichment for MapReduce-based solutions for similarity joins.

Besides the joins for vector and metric spaces, there exist similarity join approaches that exploit characteristics of certain data types: Baraglia et al. [BML10] propose a two-step similarity self-join for textual documents based on inverted lists. Afrati et al. [ASM+12] present an approach for similarity joins on data given as strings or sets. Similarity joins for set and multisets data on MapReduce is addressed in [VCL10] and [MF12]. Zhang et al. developed a spatial join algorithm for two-dimensional complex shape objects [ZHL+09].

A general problem for parallelized similarity joins is the avoidance of duplicate results [ASM+12]. E.g., in [ASM+12] lexicographic orderings are used to solve this problem. In [BML10], the *reference tile method*, first introduced in [DS00], was used. In our work, we propose a simple yet very efficient technique for avoiding duplicates.

A very general approach that also avoids duplicates is the θ-join algorithm for MapReduce by Okcan and Riedewald [OR11]. The authors propose an effective randomized balancing strategy to distribute all possible result tuples (i.e. each pair of objects) across a given set of reducers. In the reduce phase, an arbitrary join algorithm is run in each reducer to compute the join of the data points that are assigned to this reducer. Though it shows an optimal load balancing, the original θ-join does not prune any distance computations during the run. As the general algorithm does not make any assumptions about the type of join, every pair of objects has to be processed by a reducer. The authors propose strategies to avoid some computations from the start for special join types, however no strategy for similarity joins is given. A further property of the θ-join algorithm is the dependence of its data replication factor on the cluster size. A higher number of reducers (i.e., computational nodes) leads to a higher replication of the data. In contrast to this approach, our technique makes use of pruning, and the replication of the data is independent from the cluster size.

3 Efficient Distance Self-Join

In this section, we present MR-DSJ, our algorithm for a similarity self-join of vector data in MapReduce. We first introduce a basic approach showing the idea of the algorithm in Section 3.1 and prove its correctness in Section 3.2. In Section 3.3 we introduce improved techniques to reduce the number of distance computations and the communication overhead and provide an efficient implementation for MapReduce framework in Section 3.4. We give an analytical analysis of the basic approaches, which extends the experimental evaluation from Section 4 and analyzes the worst case scenario, possible bottlenecks and load balancing properties of the approach. We use the following notations for grid cells:

(1) $NC(c)$: Set of neighboring cells of cell c

(2) $cell(p)$: Cell containing point p

	00	01	10	11
00	X	X	X	X
01		-	X	-
10			-	-
11				-

Figure 1: Bit Codes for the 2d case

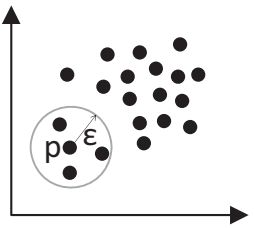


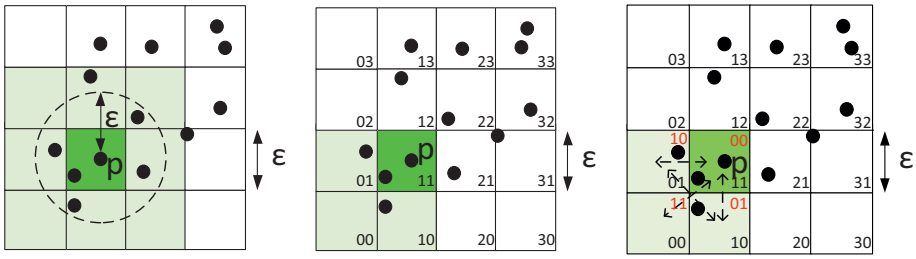
Figure 2: Small ε-neighborhood

3.1 MR-DSJ algorithm

A naive way for calculating joins is nested loop which computes the distance from each point to every other point. In the case of similarity joins where a result set can be very small (e.g. in the case of near-duplicate detection or clustering), this intuitive solution regularly produces far more distance calculations than necessary. For example, point p in Fig. 2 has only a small subset of database objects in its ε-neighborhood but nested loop would calculate the distances to all points. In order to reduce the number of distance computations we use a quite common grid-based partitioning approach with equal-sized grid cells of width ε. In such a grid all join partners of a point p are located either in the same cell as p or in the direct neighboring cells, and therefore all distance computations to objects in other cells can be pruned. This grid based discretization of the data space is depicted in Fig. 3(a) for Euclidean distance. It applies as well to other L_p norms, and weighted L_p norms are supported by scaled grid dimensions. For the point p lying in the dark green cell, each point in its ε-neighborhood is lying either in one of the adjacent (light-green) cells or in the cell of p itself. Using this knowledge, we can avoid the computation of the distances from p to the points in all other grid cells, because none of them would result in a valid result tuple.

This approach can be easily translated into a MapReduce program. Each reducer R_i is responsible for one cell c_i and its neighboring cells and computes - via nested loop - all the result tuples between all points located in c_i and $NC(c_i)$. We refer to c_i as the "home cell" of R_i. In the map phase, all points lying in a cell c_i are sent to the reducer R_i that is responsible for c_i and to the reducers of all adjacent cells, i.e. to all R_j for $c_j \in NC(c_i)$. In the reduce phase, each reducer R_i gets as input all points from c_i and $NC(c_i)$ and calculates the distances between all points in c_i and the distances of all points from c_i to all points from the neighboring cells via a nested loop. Since for each data point the distances to all objects in its neighbor cells are computed in some reducer, this method is a correct similarity self-join implementation. If ε is small compared to the distribution of values in the dataset, this simple approach can reduce the number of computations significantly.

However, this approach suffers from high communication overhead which stems from the 3^d times replicated data, since each point has to be sent to each reducer responsible for a cell neighboring to $cell(p)$. This high replication can be significantly decreased by reducing the number of neighbor cells that are taken into account by a single reducer. Namely, it is enough if each reducer R_i only considers the neighbor cells of c_i that have a



(a) Grid with cell width ε (b) Reducing the replication factor. (c) Avoiding duplicate results

Figure 3: Example grids (dark green cell: home cell, arrows indicate point replication between cells)

smaller or equal ID in every dimension, as shown in Fig. 3(b). Then the reducer performs a join on all the points from these cells. This approach still computes all valid result pairs because the same is done for *each* of the cells. For example, consider the cell c_{21}, which is the direct right neighbor of $cell(p) = c_{11}$. Although the reducer R_{11} of cell c_{11} does not compute the distance from p to the points from c_{21}, there exists another reducer R_{21} that has c_{21} as its home cell. Following the aforementioned rule, this reducer will also receive the points from c_{11} and then compute the distances between these two cells. Since the number of neighboring cells with smaller or equal ID in each dimension is equal to 2^d, this method replicates the data 2^d times, which is significantly smaller than 3^d.

Both, the 3^d and 2^d approaches though still suffer from the problem that lot of result pairs are duplicated, which occurs when two objects p and q from neighboring cells are processed in two separate reducers. This is e.g. the case when reducers of cells c_{11} and c_{21} both calculate the distances between cells c_{11} and c_{10}.

To avoid the unnecessary computation of duplicate result pairs, a reducer has to differentiate between the points from the different cells that were sent to it. Therefore we introduce a "bit code" that is sent with each data point to the reducers and identifies the relative position of the point's cell to the home cell of the reducer. The bit codes consist of d bits (for a d-dimensional grid), where each bit corresponds to one dimension. The points of the home cell itself are assigned the bit code '0^d'$= 00 \ldots 0$ (d times). For the other cells, each bit indicates if the position of this cell deviates from that of the home cell in the corresponding dimension. An example is shown in Fig. 3(c), where the bit codes for the cells that are sent to the reducer R_{11} are presented. For example, the lower left cell is assigned the bit code '11' because it differs from the home cell in both dimensions. Using these bit codes we can now decide, considering any particular reducer, which distances we have to compute in this reducer and which ones can be skipped as they are computed in other reducers. In Fig. 1 we exemplary show the decision matrices for the 2-dimensional case. For each pair of cells (represented by their bit codes) an '$\times$' indicates that the distances between the points from this cells have to be computed in the considered reducer, while a '-' indicates that the computations can be skipped as they are done in another reducer. The lower half of the matrix can be skipped due to symmetry. As a first rule, we only have to compute the distances between points from the same cell if it is the home cell of the considered reducer, because each of the other cells is the home cell of another reducer, thus

the distances between its points will be computed there. As another rule, we compute all the distances from the points in the home cell to the points in other cells. As a next step we determine all the distance computations between cells that will already be done in another reducer. Intuitively, we skip all distance computations between cells that both differ from the home cell in the same dimension, i.e. both of their bit codes contain a '1' at the same position (we refer to this rule as to *1s-rule* further in the text). In this case we know that in some other reducer, the same two cells will be processed together again and will then both contain a '0' at this position, thus the distances will be computed there. Thus, for the 2-dimensional case in Fig. 3(c), we just have to compute the distances between the cells with the bit codes '01' and '10' besides the distances including points from the home cell. A formal proof for the correctness of this step will be given in Section 3.2.

Using this approach, our join algorithm is guaranteed not to produce any duplicate result pairs. This does not only save unnecessary distance computations, but also the need to eliminate duplicates after the join.

Analysis

Now we give an analysis for uniformly distributed data in a d-dimensional space in terms of (1) number of computations per reduce-job ($comp_{red}$), (2) input size/communication per reduce-job ($input_{red}$), (3) memory footprint per reduce-job (mem_{red}), (4) overall number of computations ($comp_{overall}$) and (5) overall communication of the algorithm ($comm_{overall}$). We assume that the attribute domains are [0; 1] that means that the number of the set of all cells in the grid is equal to ε^{-d} (ε = width of a grid cell) and each cell in the grid contains $C = \varepsilon^d \cdot |DB|$ objects. Due to the 1s-rule and the resulting dependencies each reducer of MR-DSJ has to store all objects from the assigned home cell c_{home} and all neighboring cells $NC(c_{home})$. Since each cell contains C objects, the input of a single reducer (which is also the communication of a single reducer) $input_{red}$ and the memory footprint of a single reducer mem_{red} are equal to $2^d \cdot C$. Please consider that for small ε values, the value of $2^d \cdot C$ decreases very fast with growing dimensionality d. Using the bit code information the memory consumption can be halved to $2^{d-1} \cdot C$. A detailed description of this reduction technique is presented in Section 3.4.
The overall communication $comm_{overall}$ of the job is equal to $\varepsilon^{-d} \cdot input_{red} = 2^d \cdot |DB|$. Further, each reducer performs $\frac{3^d+1}{2} \cdot C^2$ computations, such that the overall computations $comp_{overall}$ is equal to $\frac{3^d+1}{2} \cdot \varepsilon^{2d} \cdot |DB|^2$. Intuitively, the points of a single cell c_i are compared to a half of all neighboring cells (there are $\frac{3^d-1}{2}$ of them) and with c_i itself, that is, c_i is compared to $\frac{3^d-1}{2} + 1 = \frac{3^d+1}{2}$ cells.

The presented analysis only holds for uniformly distributed data. Now we consider the worst case for the algorithm which occurs when all objects of the dataset are concentrated in a single cell only. In this case each reducer around the cell $cell(p)$ and the reducer of this cell itself receives all objects of the database, i.e., the overall communication $comm_{overall}$ is $2^d \cdot |DB|$. Additionally each of these reducers has to store the complete database locally, such that $input_{red}$ and mem_{red} grow to $|DB|$. The overall number of computations $comp_{overall}$ is then equal to $\frac{|DB|^2}{2}$.

3.2 Effectiveness of the MR-DSJ algorithm

In this section we show the correctness, completeness and minimality of the MR-DSJ algorithm. Therefore we prove the following lemmata that prepare Theorem 1, which states the desired properties.

Let $R \bowtie_\varepsilon R = \{(id_p, id_q) \in R \times R \,|\, d(data_p, data_q) \leq \varepsilon\}$ be the desired similarity self-join result, and $out_{DSJ} \subseteq R \times R$ denote the set of tuples reported by the MR-DSJ algorithm. id_p denotes the ID of a point p and $data_p$ represents the object coordinates. For $(p, q) \in R \times R$, let s_p^i, $s_p^i + 1$, s_q^i, $s_q^i + 1$ be the slices in dimension i to which DSJ_map assigns $data_p$ and $data_q$, respectively. Furthermore, let b_p^i, b_q^i denote the bit codes of p, q in a slice of dimension i where p, q are present.

Lemma 1 (Completeness of DSJ_map) *For each pair $(p, q) \in R \bowtie_\varepsilon R$, there is a reducer which receives both partners p, q by the partitioning of DSJ_map.*

Proof 1 *Assume that the proposition is false, i.e., there is a pair $(p, q) \in R \bowtie_\varepsilon R$ which does not meet in any reducer. This only may happen if $s_p^i + 1 < s_q^i$ or $s_p^i > s_q^i + 1$ for at least one dimension i. As the slices have width ε, it follows that $data_p^i + \varepsilon < data_q^i$ or $data_p^i > data_q^i + \varepsilon$, respectively. This eventually implies $d(data_p, data_q) > \varepsilon$ which contradicts the assumption.* □

Lemma 2 (COMPLETENESS AND MINIMALITY OF DSJ_REDUCE) *For each $(p, q) \in R \bowtie_\varepsilon R$, exactly one of the reducers performing DSJ_reduce emits (id_p, id_q).*

Proof 2 *For each dimension i, two objects $(p, q) \in R \bowtie_\varepsilon R$ are processed in exactly one slice of i since exactly the following cases may occur for their bit codes b_p^i, b_q^i:*

- *(i) $(b_p^i, b_q^i) = (1, 1)$: The pair is not processed in this slice $s_p^i + 1 = s_q^i + 1$ but will be emitted by a reducer of the neighboring slice $s_p^i = s_q^i$ where $b_p^i = b_q^i = 0$ and case (iv) applies.*
- *(ii) $(b_p^i, b_q^i) = (1, 0)$: The pair (p, q) is emitted by one of the reducers for this slice $s_p^i + 1 = s_q^i$ since q was not present in the preceding slice s_p^i, and p is not present in the subsequent slice $s_q^i + 1$.*
- *(iii) $(b_p^i, b_q^i) = (0, 1)$: Symmetric case to (ii), the pair (p, q) is emitted in this slice $s_p^i = s_q^i + 1$.*
- *(iv) $(b_p^i, b_q^i) = (0, 0)$: The pair (p, q) is emitted by a reducer for this slice $s_p^i = s_q^i$; it is not emitted in the neighboring slice $s_p^i + 1 = s_q^i + 1$ where the bits for dimension i are both set and case (i) applies. Neither in preceding slices $s < s_p^i$ nor in subsequent slices $s > s_p^i + 1$, the objects p or q are present.*

At all, the pair (p, q) is emitted by reducers of a single slice per dimension only. The intersection of these slices over all dimensions determines a single partition. As this partition is not empty, it is processed by exactly one reducer, and the proposition holds. □

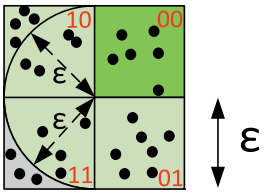


Figure 4: Pruning distance computations by $MindistCell$ (for the L_2 norm)

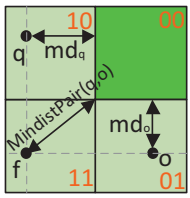


Figure 5: Example for MindistPair.

Lemma 3 (Correctness of DSJ_reduce) *MR-DSJ does not emit false positive pairs:* $out_{DSJ} \subseteq R \bowtie_\varepsilon R$.

Proof 3 *For each emitted reflexive pair* (id_n, id_n)*, the inequality* $d(data_n, data_n) = 0 \leq \varepsilon$ *trivially holds. Aside these, only pairs* (id_n, id_b) *and* (id_b, id_n) *are emitted for which* $d(data_p, data_q) \leq \varepsilon$ *was explicitly tested, and it holds that* $out_{DSJ} \subseteq R \bowtie_\varepsilon R$. □

Theorem 1 (Effectiveness of MR-DSJ) *The algorithm MR-DSJ produces complete and correct results without duplicates:* $out_{DSJ} = R \bowtie_\varepsilon R$.

Proof 4 *The completeness of MR-DSJ,* $out_{DSJ} \supseteq R \bowtie_\varepsilon R$*, follows from Lemmata 1 and 2, and the correctness of MR-DSJ,* $out_{DSJ} \subseteq R \bowtie_\varepsilon R$*, holds due to Lemma 3. The freedom of duplicates is equivalent to the minimality that any resulting pair is emitted by no more than a single reducer which was proven by Lemma 2.* □

3.3 Pruning distance computations

The basic solution provides a very efficient solution for grid-based similarity self-join on MapReduce, which decides whether a distance computation between two points is necessary by considering in which cells the points are contained. In this section we introduce two techniques to save even more distance computations and reduce replication by considering the position of points *within* a cell as will be shown in Sections 3.3.1 and 3.3.2.

3.3.1 Reducer side pruning by MindistCell

We start with an example in Fig. 4. The cell with bit code '11' contains some points that can in no case belong to a valid result pair including a point from the home cell, because their distance to *any* point in the home cell is greater than ε. For such points we do not have to compute the distances to all points from the home cell. A similar case occurs in the cell '10'. There exist some points such that none of them lies in the ε-neighborhood of *any* point from the cell '01', so we do also not have to compute the distances from those points to the points of the cell '01'. To exploit this facts for pruning distance computations, we first introduce the minimum L_p norm-based distance from a point to any point from a given cell, which is equal to the definition of the $MINDIST$ from [RKV95].

Definition 1 ($MindistCell$**)** *The distance of a point* q *to a cell* c *is defined as*

$$MindistCell(q,c) = \sqrt[p]{\sum_{i=1}^{d} \begin{cases} |lb_c[i] - q[i]|^p & q[i] < lb_c[i] \\ 0 & lb_c[i] \leq q[i] \leq ub_c[i], \textit{ where} \\ |q[i] - ub_c[i]|^p & q[i] > ub_c[i] \end{cases}}$$

$lb_c[i]$ *denotes the lower bound and* $ub_c[i]$ *the upper bound of the cell* c *in dimension* i*.*

In the reduce phase, we compute the $MindistCell$ of points q from each cell c_1 to each cell $c_2 \neq h_c$ such that the bit codes c_1 and c_2 differ in at least two dimensions. If $MindistCell(q, c_2) > \varepsilon$, the reducer does not compute the distance of q to *any* of the points in c_2.

Note that the $MindistCell$ pruning does not remove any relevant tuples such that the join result remains correct. For the reducer side pruning the proof is obvious, since for each object o and for each possible cell we test whether to prune o or not.

3.3.2 Mapper side pruning by MindistCell

The $MindistCell$ pruning is also applicable in the mapper which results in significant benefits. First, if a point is pruned in the mapper w.r.t. a certain cell c_i, it is not communicated to the reducer R_i such that the communication between mapper and reducer is reduced. This, secondly, induces that the replication factor of the overall approach decreases, which, in turn, additionally leads to a decreased runtime.

Technically, in the map phase, we detect for each home cell c_h the points q such that $MindistCell(q, c_h) > \varepsilon$. Please note that this is only possible for points from cells that differ from the home cell in at least two dimensions, thus we only have to compute the $MindistCell$ values for the points from those cells. For the mapper side pruning we show in theorem 2 that none of the pruned points occurs as a join partner in the reducer R_i. For that we briefly show which dimensions contribute to the $MindistCell(q, c)$ for a point q and a cell c in the Lemma 4.

Lemma 4 (Contribution of dimensions) *For a point* q *from cell* c_q*, and a cell* c*, only dimensions which are different in the bit codes for* c_q *and* c *contribute to* $MindistCell(q, c)$*.*

Proof 5 *According to definition of the bit code, if the values of the bit code are equal in a dimension* i*, then the range of the cells in* i *is equal. Therefore the middle rule of definition 1 applies and such a dimension does not contribute to* $MindistCell(q, c)$*.* □

Theorem 2 Completeness of map side $MindistCell$ pruning. *Let* q *be a map side pruned point, i.e., a point with* $MindistCell(q, c_{home}) > \varepsilon$ *then there is no other cell* $c_j \in NC(c_{home})$ *with* $MindistCell(q, c_j) \leq \varepsilon$*.*

Proof 6 *Let* $b^q = b_1^q, \cdots, b_d^q$ *be the bit code of* $cell(q)$ *and* $b^{c_j} = b_1^{c_j}, \cdots, b_d^{c_j}$ *the bit code of the cell* c_j*. Additionally let* I *be the set* $\{i | 1 \leq i \leq d\}$ *with* $b_i^q = 1$*. Then* $|I|$ *corresponds*

to the number of positions in which b^q differs from the bit code 0^d of the home cell and represents the dimensions which contribute to the $MindistCell$ according to Lemma 4. Lets now consider the cell c_j: due to the 1s-rule for every $b_i^{c_j}, i \in I$ must hold $b_i^{c_j} = 0$, since otherwise it will be pruned by the MR-DSJ algorithm. This in turn means that b^{c_j} has at least $(|I| + 1)$-many bits differing from b^q and therefore $MindistCell(q, c_j) > MindistCell(q, c_{home})$. □

3.3.3 Reducer side pruning by MindistPair

In the previous sections we described pruning techniques between a point and a cell. In this section we introduce *MindistPair* pruning between pairs of objects, which is defined in Definition 2.

Definition 2 *Let q, o be d-dimensional points in cells c_q, c_o, respectively, such that $bitcode(c_q)\&bitcode(c_o) = 0$, i.e., q and o are located in cells which are not pruned by the bit code pruning. Let c_{home} be the home cell, $md_q = MindistCell(q, c_{home})$ and $md_o = MindistCell(o, c_{home})$. Then $MindistPair(q, o)$ is defined as: $MindistPair(q, o) := \sqrt[p]{md_q^p + md_o^p}$, for $p \in \mathbb{N}$.*

Definition 2 states that given the $MindistCell$ to home cell for points q and o, the lower bound for the real distance between these points is L_p norm of their $MindistCell$s distances to the home cell.

Theorem 3 (Correctness of MindistPair pruning) *Let $q, o, c_q, c_o, c_{home}, md_q, md_o$ be defined as in Definition 2. Then it holds $MindistPair(q, o) \leq dist(q, o)$, where $dist(q, o)$ is a L_p or weighted L_p norm induced distance.*

At first we provide an intuitive explanation for this statement and then give a formal proof. Let points q and o in Fig. 5 be the two objects under consideration, the upper-right dark-green cell is the home cell and the arrows from q, o to the home cell represent the shortest distance to the home cell, i.e., the $MindistCell$s. The dashed lines represent the position of the point q on the x-axis and of the point o on the y-axis. Intuitively, the $MindistPair$ calculates the distance from the intersection of the dashed lines in point f to the home cell, which is always less or equal than the real distance between q and o. To be more precise, $MindistPair$ calculates the same distance for all points lying on the dashed lines in the cells c_q and c_o.

Proof 7 (MindistPair) *According to Definition 2, c_q and c_o differ in at most d dimensions and there is no dimension which contributes to md_q as well as to md_o at the same time. W.l.o.g. we assume that dimension i contributes to md_o with a value t_i, i.e., there is a hyperplane S of c_{home} to which the distance of o is t_i. If in the example in Fig. 5 i is the y-axis, then the hyperplane S would be the middle vertical line and t_i is the distance of o to this middle line. The distance $|q_i - o_i|$ is, however, equal to $t_i + x$, where $x \geq 0$ is the distance of q to the hyperplane S (in the example, x would the distance from the middle line to point q). I.e., $t_i \leq |q_i - o_i|$ for every dimension i. Therefore it holds that $\sqrt[p]{\sum_{i=1}^{d} t_i^p} \leq$*

$\sqrt[p]{\sum_{i=0}^{k} |q_i - o_i|^p}$. *If c_q and c_o differ in less than d dimensions, then according to Lemma 4, the dimensions which do not differ do not contribute to the $MindistCell$ and, therefore, the proof also holds for this case.*

3.4 Implementation of MR-DSJ algorithm

In the preceding sections, we have introduced the concepts of our MR-DSJ algorithm. Now we present its implementation, which consists of the Listings 1 and 2 for the mapper and the reducer, respectively. Both rely on a few global input parameters, namely the radius ε of the similarity join, the dimensionality of the data and the range of the data space, in terms of $bits\_per\_dimension$. Let us walk through our pseudocode. The recursion in the mapper (Listing 1) is started by the method *DSJ_map* which for each point first calculates the home slice and the distance to the upper slice boundary in each dimension. The value of the similarity radius ε is used to define the width of the slices. The recursion in *map_recursive* over the dimensions is initialized by a zero partition ID and a zero bit code. The parameter mdp aggregates the p-th power of the mindist from the point to the respective adjacent cells it is assigned to by summing up dist[dim]^p over dimensions dim where the bit code is set to 1. That means that no contributions to mindist are added in dimensions of home slices, i.e., where the respective bit equals 0. The recursive calls for adjacent slices are conditional to the test if the mindist does not exceed ε (tested by $md^p \leq \varepsilon^p$). This way, the mapper-side mindistCell test is implemented with almost no additional effort compared to the basic variant. The value of mdp is handed over to the reducer for further mindist-based pruning.

The MR-DSJ reducer (Listing 2) cascades **for** and **if** statements to realize the respective loops and pruning strategies introduced in Section 3.3. Within the loop over the value records from the reducer's input, the second loop iterates over individual buffers for each neighboring cell. This separate buffer organization allows for efficient bit code pruning and *mindistCell* filtering of cells as a whole and, this way, prevent from unnecessary iterations over the contents in a pruned cell. Only for the remaining adjacent cells, all objects are tested by the last *mindistPair* filter before the final exact distance check from the join condition, and the resulting pairs are emitted.

As an additional optimization, we use the bit codes c_n not only to prevent from duplicate distance computations and duplicate results but also for minimizing the main memory footprint in the reducers. The tuple $(c_n, id_n, data_n)$ is buffered only if $c_n < maxcode$ holds since the bitwise AND test includes the most significant bits (MSB), and all pairs with set MSBs disqualify in particular. Reading the input in increasing bit code order, thus, enables to safely exclude all tuples with set MSB from the buffer; all their potential join partners got inserted into the buffer in earlier steps but no one will arrive later. The MSB threshold for the bit codes is precomputed in advance by $maxcode = 2^{dimension-1}$, which may be implemented by $maxcode = 1 << (dimension - 1)$.
The $c_n < maxcode$ test saves up to half the main memory consumption on average over all the reducers as the mapper assigns every object to up to two partitions per dimension.

Listing 1: The MR-DSJ mapper recursively assigns objects $(id, coord)$ to neighboring partitions pid. The bit codes c reflect the local neighborhood relationship for each dimension dim, and mdp aggregates the p-th power of the minimum distance to objects in the respective partition.

```
void DSJ_map(int id, float [] coord)
  for dim = 1..dimension do
    home_slice[dim] = int(coord[dim] / ε);
    dist [dim] = (home_slice[dim]+1) * ε − coord[dim];
  map_recursive (1, 0, 0, 0.0, id, coord);

void map_recursive (int dim, long pid, int c, float mdp, int id, float[] coord)
  if (dim ≤ dimension)
    pid = (pid << bits_per_dimension) + home_slice[dim];
    map_recursive (dim+1, pid, c<<1, mdp, id, coord);

    mdp = mdp + dist[dim]^p;
    if (mdp ≤ ε^p)
      map_recursive (dim+1, pid+1, (c<<1)|1, mdp, id, coord);
  else
    emit(pid, (c, mdp, id, data));
```

Technically, the required sorting of the values in ascending bitcode order for each reducer is accomplished by the 'secondary sort' functionality of MapReduce. The key for the shuffle phase does not just comprise the partition ID, but includes the bit code in order to sort the input with respect to $(partition\_id, bitcode)$ in lexicographic order.

4 Experiments

In this section we present an experimental evaluation of our new MR-DSJ approach. In Section 4.1 we evaluate the scalability of our approach on synthetic data and compare it to the θ-join approach from [OR11]. In [OR11] the author propose the usage of specialized join algorithms inside the reducer tasks. Therefore we implemented RSJ join [BKS93] which is very well suited for low-/medium-dimensional vector data.

In Section 4.2 we evaluate the efficiency gain of the optimizations presented in Section 3.3 compared to the basic MR-DSJ algorithm. In Section 4.3 we evaluate our approach on real-world data sets.

All the experiments were performed on a cluster running Hadoop 0.20.2 and consisting of 14 nodes with 8 cores each that are connected via a 1 Gbit network. Each of the nodes has 16 Gb RAM. For each experiment the number of the performed distance computations as well as the runtime is measured. Runs of the algorithms were aborted if they did not finish within 8 hours.

Listing 2: The MR-DSJ reducer computes the join results. The bit codes prevent from both, duplicate distance calculations and duplicate results from concurrent reducers while reducing the local main memory footprint as well. The minimum distances of objects to cells avoid several distance calculations.

```
void DSJ_reduce(long partition_id , Iterator values)
  cellBuffer . clear () ;
  forall ((c_n, mdp_n, id_n, data_n) in values)
    for (c_b=0; c_b < maxcode; c_b + + )
      if (c_n & c_b == 0)     // bitcode filter
        if (mindist(data_n, c_b) ≤ ε) // mindistCell filter
          forall ((mdp_b, id_b, data_b) in cellBuffer[c_b])
            if (mdp_n + mdp_b ≤ ε^p) // mindistPair filter
              if (d(data_n, data_b) ≤ ε) // join condition
                emit(id_n, id_b);
                emit(id_b, id_n); // symmetric pair, if desired
    if (c_n < maxcode)   // relies on secondary sort
      cellBuffer [c_n].insert((mdp_n, id_n, data_n));
    if (c_n == 0)
      emit(id_n, id_n);   // reflexive pair, if desired
```

4.1 Scalability on synthetic data

In this section we evaluate our join approach on synthetic datasets of different sizes and dimensionalities. In all our synthetic datasets, the attribute values are uniform distributed between 0 and 1.

In our first experiment, the database sizes of our synthetic datasets are varied from 1 million points to approximately 10 million points. All used datasets are 2-dimensional and are processed with the parameter $\varepsilon = 0.05$. The results are shown in Fig. 6 (number of calculations) and Fig. 7 (runtime). Please note the logarithmic scale on the y-axes. As expected, the runtimes and the number of performed distance calculations of both algorithms increase for increasing database sizes. For these 2-dimensional datasets, the number of distance calculations performed by MR-DSJ is by a factor 2 to 3 higher than that of θ-join (denoted by "TJ" in the figures), as the RSJ-join in the reduce phase of TJ saves many calculations. However, for all datasets, the runtimes of MR-DSJ are significantly lower (by a factor 3 to 5) than those of TJ. This difference can be explained by the fact that in TJ *each* pair of data points is processed by a common reducer. Even if the distance computations for many pairs can be pruned by the RSJ join in the corresponding reducers, the distribution of the data and the building of the internal indexes for RSJ leads to high runtimes. In MR-DSJ, however, only pairs of points that have a certain proximity are processed by a common reducer.

In the next experiment, we vary the dimensionality of our datasets from 2 to 4 dimensions. All datasets consist of ca. 1 million points and are processed with the parameter $\varepsilon = 0.05$. The results presented in Fig. 9 show that the runtimes for both algorithms increase with higher dimensionality, while the runtimes of the MR-DSJ approach are always

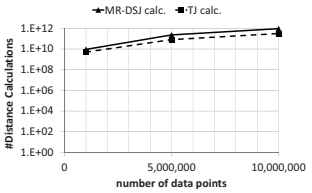


Figure 6: Database size vs. number of Distance Calculations

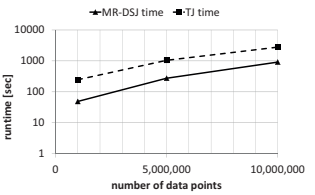


Figure 7: Database size vs. Runtime

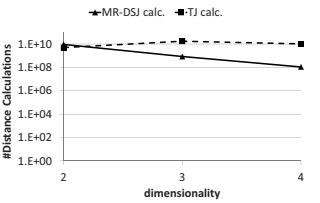


Figure 8: Dimensionality vs. number of Calculations

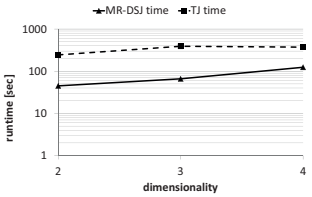


Figure 9: Dimensionality vs. Runtime

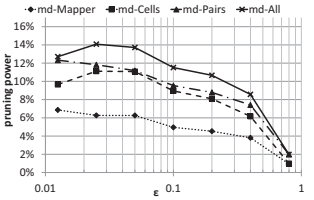


Figure 10: Efficiency gain by the optimizations on a 2d dataset

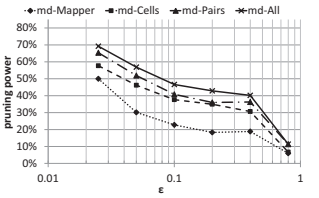


Figure 11: Efficiency gain by the optimizations on a 4d dataset

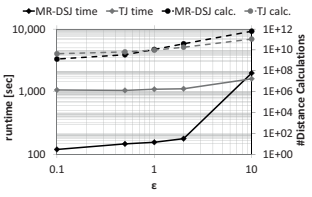


Figure 12: Varying ε on the cloud dataset

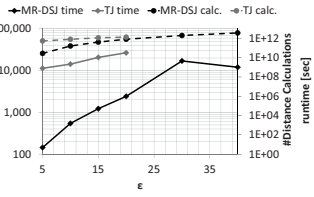


Figure 13: Varying ε on the minutiae dataset

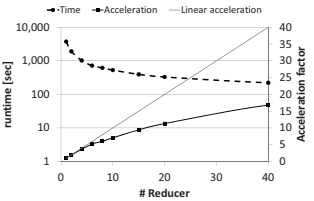


Figure 14: Scalability on the minutiae dataset

significantly lower than those of TJ. For the θ-join, the number of distance computations shown in Fig. 8 is hardly influenced by the dimensionality, as the partitioning strategy of this algorithm does not depend on the data dimensionality. In contrast, the number of distance computations for MR-DSJ strongly decreases for higher dimensionalities. This is caused by the fact that the data points are distributed among a larger number of grid cells for higher dimensionalities and thus for each point, the number of points in the same cell and the neighboring cells decreases, such that fewer distance computations have to be performed. The increasing runtime for higher dimensionalities results from the higher communication overhead between mappers and reducers which is caused by the higher replication factor.

4.2 Efficiency gain by the $MINDIST$ filters

In this section we evaluate the efficiency gain by the optimizations from Section 3.3. Therefore we use two synthetic datasets, with 2 and 4 dimensions. The experiment is repeated for different values of ε. For both datasets we measure the influence of each single filter and their combination in terms of saved distance calculations. The results are depicted in Fig. 10 (for the 2-dimensional dataset) and 11 (for the 4-dimensional dataset).

First, we evaluate the efficiency gain by using the mapper side pruning by computing the MindistCell (cf. Section 3.3.2) (denoted by "md-Mapper"). Depending on the ε value, we save up to 7% of the distance calculations that would be performed by the basic MR-DSJ algorithm on the 2-dimensional dataset and even up to 50% on the 4-dimensional dataset. The pruning power of this optimization, as well as the other optimizations, is the best for small ε values, which would be reasonable values for e.g. clustering or outlier detection. For ε values approaching 1 (please note that the synthetic data points lie in $[0, 1]$ in each dimension), naturally only a small amount of distance calculations can be pruned as the join selectivity approaches 77% (2-dimensional) and 53% (4-dimensional) and thus most distances have to be calculated. Additionally using the reducer side pruning by MindistCell (cf. 3.3.1) (denoted by "md-Cells"), leads to the pruning of up to 10% of the distance calculations for the 2-dimensional dataset and up to 58% for the 4-dimensional dataset. Using the reducer side pruning by MindistPair from Section 3.3.3 (together with the mapper side pruning), denoted by "md-Pairs", we can prune up to 12% of the distance calculations for the 2-dimensional dataset and up to 65% for the 4-dimensional dataset. Finally, the advanced MR-DSJ algorithm using all optimizations needs to perform up to 14% less distance calculations than the basic MR-DSJ algorithm for the 2-dimensional dataset and up to 70% less for the 4-dimensional dataset.

Overall, we observe that for the 4-dimensional dataset, the pruning power of the optimizations is much higher than for the 2-dimensional dataset. (This effect can also be observed in Fig. 8.) This can be explained by the fact that in higher-dimensional spaces, a larger percentage of the data points lie near the borders of their respective cell. As the optimizations mostly prune distance calculations for points near the borders, they are much more effective in a 4-dimensional space than in a 2-dimensional space.

4.3 Scalability on real world data

We evaluate our approach on two real-world datasets. The first dataset is a sample of 5 million records from a dataset of cloud observations from land stations and ships [HW99] that is available online[2]. We use a 2-dimensional dataset for which the attributes "latitude" and "longitude" are used. The second dataset "minutiae" contains extracted minutiae data from the fingerprint datasets "NIST Special Database 14" and "NIST Special Database 29[3]". It contains ca. 11 million three-dimensional entries, distributed in the ranges [0;832],

[2] http://cdiac.ornl.gov/ftp/ndp026c/

[3] http://www.nist.gov/srd/nistsd{14,29}.cfm

[0;768] and [0;100], respectively. As the efficiency of the MR-DSJ approach depends on the parameter ε which determines the size of the grid cells, we evaluated its scalability (and that of TJ) to different ε-values on both datasets. The results are shown in Fig. 12 and Fig. 13, respectively. Both experiments show the expected behavior for both algorithms - increasing ε values result in higher runtimes and number of performed calculations, which makes sense as for higher ε values we also get a larger result set.

The number of distance calculations as well as the runtime for different ε values vary significantly for the MR-DSJ approach. For $\varepsilon = 0.1$, MR-DSJ finishes on the cloud dataset (Fig. 12) in 119 seconds and performs ca. $1.3 \cdot 10^9$ distance calculations. For higher values for ε the number of needed calculations increases. For $\varepsilon = 10$ the runtime is approx. 2000 seconds and approx. $6 \cdot 10^{11}$ distance calculations are performed. Whereas the number of distance calculations of TJ behaves similar to that of MR-DSJ, the runtimes for MR-DSJ are significantly lower than those of TJ, except for the value $\varepsilon = 10$, which leads to very large grid cells in the MR-DSJ approach.

For the minutiae dataset (Fig. 13) our observations are similar the those for the cloud dataset. The runtimes and numbers of calculations increase for increasing ε values; for values larger than 20, the TJ approach did not finish within 8 hours and is thus not included in the figure. The numbers of performed distance calculations are again similar for both algorithms. However, MR-DSJ outperforms TJ in terms of runtime by a factor of 10 to 80.

In a further experiment (Fig. 14) we analyze the scalability of our approach on the minutiae w.r.t. a varying cluster size. Therefore we vary the number of used reducers from 1 to 40. In Fig. 14 we depict the runtimes of MR-DSJ as well as the acceleration factor compared to the runtime using only 1 reducer. For up to 6 reducers, the acceleration factor is nearly linear. For larger numbers of reducers, the acceleration is sub-linear which is mainly caused by data skew: As some grid cells contain more data points that other ones, the reduce tasks processing these cells have longer runtimes than those of the other reducers. However, MR-DSJ reaches a significant acceleration for increasing cluster sizes, i.e. using 40 reducers the runtime is lower than the runtime for 1 reducer by a factor of 17.

5 Further Analysis

In the previous sections we analyzed the basic MR-DSJ algorithm in terms of number of calculations. In this section we investigate its general behavior and present scenarios and use cases which benefit the most from its usage. We also identify problematic scenarios for MR-DSJ and present ideas how to tackle this problems in future work.

5.1 Data replication and data dimensionality

The effect of data replication is very common in the MapReduce framework. The only (desired) way to share information is its duplication on different computational nodes. Since each reducer in MR-DSJ relies on information from neighboring cells, the replication of

data points is unavoidable. As shown earlier, our approach produces 2^d replicas of every data point, i.e. the replication factor grows very fast with every additional dimension of the data that is used for partitioning the data space. Thus, our approach is most suitable for data with a low or medium dimensionality d. Please note that d does *not* necessarily equal the dimensionality of the original data, since we can apply dimensionality reduction techniques (e.g. the techniques from the Mahout framework) to obtain a reasonably low dimensionality.

5.2 Influence of the parameter ε

In the analysis parts of Section 3 we already mentioned the worst case scenario for our approach: all points are located in a single grid cell and thus are processed by a single reducer. This case occurs for very skewed data or when the value of ε is very large in comparison to the largest distances between data objects. On the other hand, small ε values result in a high number of grid cells with few elements. In this case the computation of the self-join becomes very efficient since most of the distance computations can be pruned. Due to these facts our algorithm is best suited for join tasks with small ε values which among other things arise in near-duplicate detection, data cleaning and clustering tasks.

One possible way to solve the problem with large ε values would be a connection of our approach with an adjusted version of the θ-join [OR11]. The join inside cells containing too many objects would then be calculated by θ-join. This connection could also be helpful in the case of highly skewed data. We will examine such a connection in our future work.

The threshold ε also directly influences the number of created reduce tasks and therefore the possible parallelization of our approach. A small number of cells, which corresponds to a small number of created reduce-jobs, can significantly deteriorate the performance of the complete task. This case will for example occur if the chosen threshold ε is very large. A possible solution for this problem, which will be investigated in our future work, is adjusting the cell width to larger or smaller values than ε.

6 Conclusion

In this work we proposed the novel distance-based similarity self-join algorithm MR-DSJ for the MapReduce framework. We presented different optimization solutions for the used grid-based approach which minimize the communication and the number of needed distance computations. We provided a theoretical analysis of the used basic techniques as well as an experimental evaluation of the efficiency of our approach. The evaluation shows that our solution often significantly outperforms the existing θ-join algorithm in terms of number of calculations and execution runtime.

Acknowledgments This work has been supported by the B-IT Research School of the Bonn-Aachen International Center for Information Technology. We also thank Roman Haag for his technical support.

References

[ABE+10] Alexander Alexandrov, Dominic Battré, Stephan Ewen, Max Heimel, Fabian Hueske, Odej Kao, Volker Markl, Erik Nijkamp, and Daniel Warneke. Massively Parallel Data Analysis with PACTs on Nephele. *PVLDB*, 3:1625–1628, 2010.

[ABPA+09] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, and Avi Silberschatz. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *PVLDB*, 2(1):922–933, 2009.

[ASM+12] F.N. Afrati, A.D. Sarma, D. Menestrina, A. Parameswaran, and J.D. Ullman. Fuzzy Joins Using MapReduce. *ICDE*, 2012.

[AU10] Foto N. Afrati and Jeffrey D. Ullman. Optimizing Joins in a Map-Reduce Environment. In *EDBT*, pages 99–110, 2010.

[BBBK00] Christian Böhm, Bernhard Braunmüller, Markus M. Breunig, and Hans-Peter Kriegel. High Performance Clustering Based on the Similarity Join. In *CIKM*, pages 298–305, 2000.

[BBKK01] C. Böhm, B. Braunmüller, F. Krebs, and H.P. Kriegel. Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data. In *SIGMOD*, pages 379–388, 2001.

[BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient Processing of Spatial Joins Using R-Trees. In *SIGMOD*, pages 237–246, 1993.

[BML10] Ranieri Baraglia, Gianmarco De Francisci Morales, and Claudio Lucchese. Document Similarity Self-Join with MapReduce. In *ICDM*, pages 731–736, 2010.

[BPE+10] Spyros Blanas, Jignesh M. Patel, Vuk Ercegovac, Jun Rao, Eugene J. Shekita, and Yuanyuan Tian. A comparison of join algorithms for log processing in MapReduce. In *SIGMOD*, pages 975–986, 2010.

[CGK06] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A Primitive Operator for Similarity Joins in Data Cleaning. In *ICDE*, page 5, 2006.

[Dat06] Chris J. Date. *The relational database dictionary - a comprehensive glossary of relational terms and concepts, with illustrative examples*. O'Reilly, 2006.

[DG04] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI*, pages 137–150, 2004.

[DNSS92] David J. DeWitt, Jeffrey F. Naughton, Donovan A. Schneider, and S. Seshadri. Practical Skew Handling in Parallel Joins. In *VLDB*, pages 27–40, 1992.

[DQRJ+10] Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Alekh Jindal, Yagiz Kargin, Vinay Setty, and Jörg Schad. Hadoop++: Making a Yellow Elephant Run Like a Cheetah. *PVLDB*, 3:518–529, 2010.

[DS00] Jens-Peter Dittrich and Bernhard Seeger. Data Redundancy and Duplicate Detection in Spatial Join Processing. In *ICDE*, pages 535–546, 2000.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *SIGKDD*, pages 226–231, 1996.

[HW99] C.J. Hahn and S.G. Warren. Extended edited synoptic cloud reports from ships and land stations over the globe, 1952–1996. *NDP026C, Carbon Dioxide Information Analysis Center*, 1999.

[LSCO12] Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. Efficient Processing of k Nearest Neighbor Joins using MapReduce. *PVLDB*, 5(10):1016–1027, 2012.

[MF12] Ahmed Metwally and Christos Faloutsos. V-SMART-Join: A Scalable MapReduce Framework for All-Pair Similarity Joins of Multisets and Vectors. *PVLDB*, 5(8):704–715, 2012.

[Mon00] Alvaro E. Monge. Matching Algorithms within a Duplicate Detection System. *IEEE Data Eng. Bull.*, 23(4):14–20, 2000.

[OR11] Alper Okcan and Mirek Riedewald. Processing theta-joins using MapReduce. In *SIGMOD*, pages 949–960, 2011.

[PD96] Jignesh M. Patel and David J. DeWitt. Partition Based Spatial-Merge Join. In *SIGMOD Conference*, pages 259–270, 1996.

[PPR+09] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD*, pages 165–178, 2009.

[RKV95] Nick Roussopoulos, Stephen Kelley, and Frédéic Vincent. Nearest Neighbor Queries. In *SIGMOD*, pages 71–79, 1995.

[RRS00] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient Algorithms for Mining Outliers from Large Data Sets. In *SIGMOD*, pages 427–438, 2000.

[RU11] A. Rajaraman and J.D. Ullman. *Mining of massive datasets*. Cambridge Univ Pr, 2011.

[SRT12] Yasin N. Silva, Jason M. Reed, and Lisa M. Tsosie. MapReduce-based similarity join for metric spaces. In *Proceedings of the 1st International Workshop on Cloud Intelligence*, Cloud-I '12, pages 3:1–3:8, New York, NY, USA, 2012. ACM.

[VCL10] R. Vernica, M.J. Carey, and C. Li. Efficient parallel set-similarity joins using MapReduce. In *SIGMOD*, pages 495–506, 2010.

[XWLY08] Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140, 2008.

[ZHL+09] Shubin Zhang, Jizhong Han, Zhiyong Liu, Kai Wang, and Zhiyong Xu. SJMR: Parallelizing spatial join with MapReduce on clusters. In *CLUSTER*, pages 1–8, 2009.

[ZJ03] Yanchang Zhao and Song Junde. AGRID: An Efficient Algorithm for Clustering Large High-Dimensional Datasets. In *PAKDD*, pages 271–282, 2003.

[ZLJ12] Chi Zhang, Feifei Li, and Jeffrey Jestes. Efficient parallel kNN joins for large data in MapReduce. In *EDBT*, pages 38–49, 2012.

Extending the MPSM Join

Martina-Cezara Albutiu, Alfons Kemper, Thomas Neumann

Technische Universität München
Boltzmannstr. 3
85748 Garching, Germany
firstname.lastname@in.tum.de

Abstract:

Hardware vendors are improving their (database) servers in two main aspects: (1) increasing main memory capacities of several TB per server, mostly with non-uniform memory access (NUMA) among sockets, and (2) massively parallel multi-core processing. While there has been research on the parallelization of database operations, still many algorithmic and control techniques in current database technology were devised for disk-based systems where I/O dominated the performance. Furthermore, NUMA has only recently caught the community's attention. In [AKN12], we analyzed the challenges that modern hardware poses to database algorithms on a 32-core machine with 1 TB of main memory (four NUMA partitions) and derived three rather simple rules for NUMA-affine scalable multi-core parallelization. Based on our findings, we developed MPSM, a suite of massively parallel sort-merge join algorithms, and showed its competitive performance on large main memory databases with billions of objects. In this paper, we go one step further and investigate the effectiveness of MPSM for non-inner join variants and complex query plans. We show that for non-inner join variants, MPSM incurs no extra overhead. Further, we point out ways of exploiting the roughly sorted output of MPSM in subsequent joins. In our evaluation, we compare these ideas to the basic execution of sequential MPSM joins and find that the original MPSM performs very well in complex query plans.

1 Introduction

Hardware vendors are improving their (database) servers in two main aspects: (1) increasing main memory capacities of several TB per server, mostly with non-uniform memory access (NUMA) among sockets, and (2) massively parallel multi-core processing. These emerging hardware characteristics will shape database system technology in the near future. New database software has to be carefully targeted against the upcoming hardware developments. This is particularly true for main memory database systems that try to exploit the two main trends – increasing RAM capacity and core numbers. So far, main memory database systems were either designed for transaction processing applications, e.g., VoltDB [Vol10], or for pure OLAP query processing [BMK09]. However, industry thought leaders such as Hasso Plattner of SAP voiced the requirement for so-called real-time or operational business intelligence that demands complex query processing in "real time" on main memory resident data. SAP's Hana [FCP+11] and our hybrid OLTP&OLAP

database system HyPer [KN11], for which MPSM [AKN12] was developed, are two such databases. The query processing of in-memory DBMSs is no longer I/O bound and, therefore, it makes sense to investigate massive intra-operator parallelism in order to exploit the multi-core hardware effectively. Only massively parallel query engines will be able to meet the instantaneous response time expectations of operational business intelligence users if large main memory databases are to be explored. Single-threaded query execution is not promising to meet the high expectations of these database users as the hardware developers are no longer concerned with speeding up individual CPUs but rather concentrate on multi-core parallelization.

Merely relying on straightforward partitioning techniques to maintain cache locality and to keep all cores busy will not suffice for the modern hardware that increases main memory capacity via non-uniform memory access (NUMA). Besides the multi-core parallelization, also the RAM and cache hierarchies have to be taken into account. In particular, the NUMA division of the RAM has to be considered carefully. The whole NUMA system logically divides into multiple nodes, which can access both local and remote memory resources. However, a node can access its own local memory faster than remote memory, i.e., memory which is local to another node. Therefore, **data placement** and **data movement** such that threads/cores work mostly on local data is a key prerequisite for high performance in NUMA-friendly data processing.

Micro-benchmarks on our 1 TB, NUMA database server led us to state in [AKN12] the following three rather simple and obvious rules (called "commandments") for NUMA-affine scalable multi-core parallelization:

C1 *Thou shalt not write thy neighbor's memory randomly* – chunk the data, redistribute, and then sort/work on your data locally.

C2 *Thou shalt read thy neighbor's memory only sequentially* – let the prefetcher hide the remote access latency.

C3 *Thou shalt not wait for thy neighbors* – don't use fine-grained latching or locking and avoid synchronization points of parallel threads.

By design, MPSM obeys all three commandments. It scales almost linearly with the number of cores and outperforms state-of-the-art parallel join implementations. The IBM database research group at Almaden further improved our original MPSM algorithm with respect to an optimized data movement that avoids cross traffic between NUMA partitions [LPP+13]. In this paper, we go one step further and investigate the effectiveness of MPSM for non-inner join variants and complex query plan.

The remainder of the paper is structured as follows: In Section 2, we recap the MPSM algorithm. Then, we extend MPSM to compute non-inner join variants such as semi and outer joins in Section 3. In Section 4, we investigate the applicability of MPSM for complex query plans. We evaluate the presented MPSM variants in Section 5. Finally, we conclude our work in Section 6.

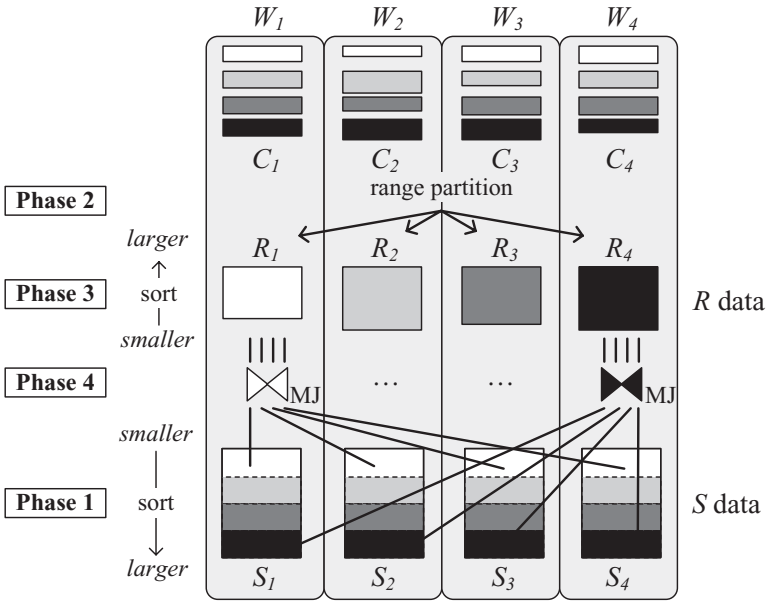


Figure 1: P-MPSM join with four workers W_i

2 The MPSM Algorithm

In [AKN12], we presented a suite of massively parallel sort-merge join algorithms for main memory and for disk-based systems. We limit the discussion of MPSM and its extensions to the range-partitioned version (P-MPSM) in this paper.

The MPSM join is designed to take NUMA architectures into account, which were not yet in the focus of prior work on parallel join processing for main memory systems. Though, we emphasize that MPSM is oblivious to specific NUMA architectures as it only assumes the locality of a RAM partition for a single core – without relying on multiple cores sharing the locality of RAM partitions or caches. As illustrated in Figure 1, each data chunk is processed, i.e., sorted, locally. Unlike traditional sort-merge joins, we refrain from merging the sorted runs to obtain a global sort order and rather join them all in a brute-force but highly parallel manner. During the subsequent join phase, data accesses across NUMA partitions are sequential, so that the prefetcher mostly hides the access overhead. We do not employ shared data structures so that no expensive synchronization is required. Therefore, MPSM obeys all three NUMA-commandments by design.

The P-MPSM processes its input in four phases as sketched in Figure 1 for a scenario with four workers. We call R the private input and S the public input. The input data is chunked into equally sized chunks among the workers, so that for instance worker W_1 is assigned a chunk R_1 of input R and another chunk S_1 of input S. In phase 1, each worker sorts its public input run locally, resulting in runs S_1 to S_4. Subsequently, in phase 2, the private input chunks C_1 to C_4 are range partitioned. Thereby, the private input data is partitioned into disjoint key ranges as indicated by the different shades in Figure 1 ranging from white

over light and dark gray to black. In phase 3, each worker then sorts its private input chunk and in phase 4, each worker merge-joins its own private run R_i with all public input runs S_j.

Besides the general algorithmic structure, several implementation details of the MSPM are essential for its good performance. This is discussed in detail in [AKN12], but in particular efficient sorting and efficient range partitioning are absolutely essential. Both steps must be executed largely branch-free, comparison-free, and synchronization-free, as otherwise they can easily dominate the join costs.

3 Outer, Semi, Anti Semi Joins

Depending on whether the private or the public input (or both) produces outer, semi, or anti semi join result tuples, additional data structures are required. As we will show, the costs for these data structures in terms of time and space, are negligible.

As each thread traverses its private input several times, we need to maintain joined-flags for private input tuples. We then only produce (additional) output tuples if

- no join partner has been found (outer and anti semi join) or if
- the respective tuple has not been joined yet (semi join).

On the contrary, each public input tuple is touched only once due to the implicit partitioning of the public input. Therefore, we can decide right away if outer, semi, or anti semi output tuples have to be produced. Before the join phase 4, outer, semi, and anti semi MPSM process their inputs the same way as inner join MPSM. If necessary, joined-flags in the form of bitmaps tracking whether private input tuples have (already) been joined are initialized when MPSM enters join phase 4.

While semi joins are very similar to inner joins (output tuples are produced if a join partner has been found), outer and anti semi joins require some attention in order to avoid missing or duplicating output tuples due to range partitioning and interpolation search. In the following, we briefly describe the implementations of outer, semi, and anti semi joins. R and S denote the private and the public input, respectively.

3.1 Outer Joins

Outer joins bring together matching tuples like inner joins, and, in addition, they produce output tuples for input tuples that didn't find a join partner. The left (private input) outer join $R⟕S$ requires joined-flags indicating whether the private input tuples already took part in the join or not. Each time a regular join output tuple is generated, the corresponding flag is set. In Figure 2, this is indicated by the "set" arrows toward the bitmap, which are shown only for worker W_1 for the sake of readability. During its last merge join, in addition to the regular merge join computation, each thread checks the joined-flags and

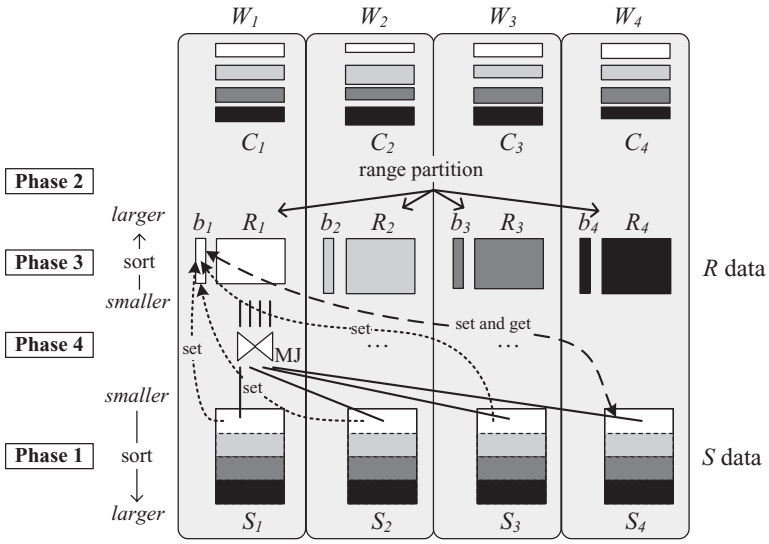
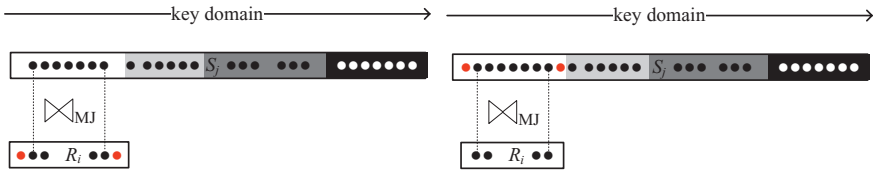


Figure 2: Outer and anti semi P-MPSM join with four workers W_i maintaining each an additional joined-bitmap b_i for their private input runs

produces output tuples for private input tuples for which the flag is not set. In the example in Figure 2, when worker W_1 conducts the last merge join of its private input run R_1 with S_4, it sets flags for joined tuples and gets flags to decide on the generation of additional output tuples ("set and get" arrow from and to bitmap b_1).

As opposed to the computation of inner joins, not only tuples finding a join partner have to be considered but also those that do not find a match. This requires special care: Due to interpolation search, the first private input tuples may be skipped. Further, the last tuples may be skipped as merge join stops early as soon as one of the inputs terminates. These two issues are illustrated in Figure 3a where the first and the last tuple of R_i are not considered. In order to not miss outer output tuples, it is therefore crucial for each thread to scan its whole private input (at least) once. Thus, when executing the last merge join, the threads omit interpolation search on their private input runs and start scanning at the first tuple in their run. This actually mainly affects R_1 as for all other R runs interpolation search is usually performed on S runs. Further, the threads scan their private input run up to its last tuple irrespective of the occurrence of matching tuples in S.

The right (public input) outer join $R \bowtie S$ is straightforward as it can be decided at the time a tuple is processed whether it found a join partner or an extra output tuple has to be returned. However, here again due to interpolation search and early stop of merge join, the first and last tuples of the considered key range may be skipped as illustrated in Figure 3b. Therefore, interpolation search on public input runs S_j is not based on tuple key values of R_i but on the splitters determined in phase 2. That way, all S tuples within a worker's key range (white to black shades in the figures) are considered in the join processing.



(a) Left outer and anti semi joins (b) Right outer and anti semi joins

Figure 3: Outer and anti semi join require attention: due to interpolation search and early stop of merge join the outer-most (red) tuples are skipped by inner-join MPSM

3.2 Semi Joins

Semi joins produce output for tuples of one of the inputs which find a join partner in the other input. In contrast to inner joins, one input tuple may produce at most one output tuple. For this purpose, the left (private input) semi join $R \ltimes S$ requires joined-flags indicating whether a private input tuple already took part in the join or not. If so, the tuple will not produce any output again. If the flag is not set and a public input tuple matches, an output tuple is produced and the corresponding flag is set. As opposed to left outer joins, the joined-flags are not only checked at the end of the join phase 4 but need to be consulted for each matching tuple during each single merge join. This is illustrated in Figure 4 using "get and set" arrows for all merge joins.

For the right (public input) semi join $R \rtimes S$, the private input is scanned for a specific public input tuple until one match is found or the key of the private input is greater than the current public input key. If a match exists, an output tuple is generated and the worker moves on to the next public input tuple as the current may not produce any further output.

3.3 Anti Semi Joins

Anti semi joins are the opposite of semi joins. Output is produced for input tuples that do not find a join partner in the other input. The left (private input) anti semi join $R \triangleright S$ requires joined-flags indicating whether the private input tuples already took part in the join or not. Each time a public input tuple matches, the flag for the corresponding private input tuple is set (without producing an output tuple). In Figure 2, this is indicated by the "set" arrows toward the joined-bitmap b_1 of worker W_1. As for outer joins, during the last merge join, in addition to setting flags for joined tuples, each thread checks the bitmap and produces an output tuple for each private input tuple, for which the flag is not set ("set and get" arrow from and to bitmap b_1). Due to interpolation search and early stop of merge join, some tuples may be skipped as illustrated in Figure 3a. As for left outer joins, by omitting interpolation search on private input runs for the last merge join and scanning the private input completely, we make sure that no anti output tuples are missed.

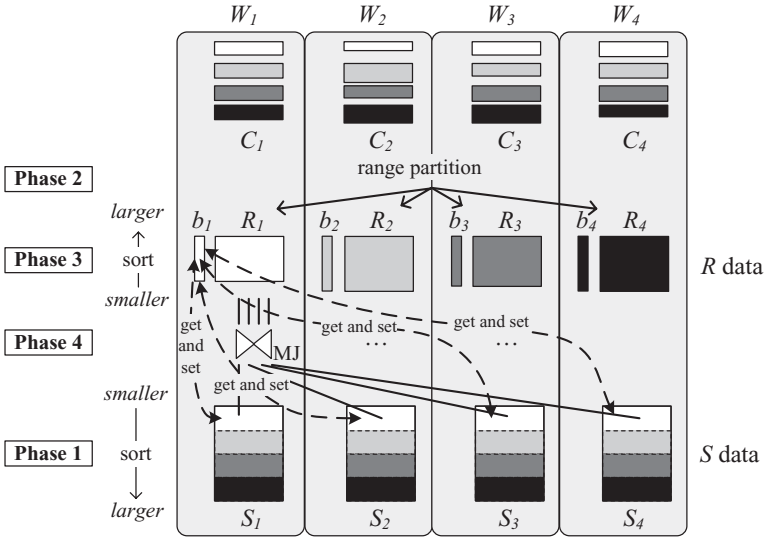


Figure 4: Semi P-MPSM join with four workers W_i maintaining each an additional joined-bitmap b_i for their private input runs

When computing the right (public input) anti semi join $R \triangleleft S$, it can be decided at the time a tuple is processed whether it found a join partner or – in case no join partner at all was found – an output tuple has to be returned. Again, we need to make sure that no output tuples are missed due to interpolation search and early stop of merge join as shown in Figure 3b. Therefore, as for right outer joins, interpolation search on public input runs S_j is based on the splitters determined in phase 2, so that all S tuples within a worker's key range are considered in the join processing.

4 Complex Query Plans: The Guy Lohman Test

After having covered MPSM for inner, outer, semi, and anti semi joins, we put MPSM to the Guy Lohman test [Gra93] stating that a join operator must not only be useful for joining two inputs but also in complex query execution. In particular, an operator is suitable in complex query processing if it does not require intermediate results to be materialized for further processing. MPSM is roughly order preserving, which can be exploited in subsequent join operations of a complex query plan as shown by [SAC$^+$79, CKKW00]. We depict different ways of how to make use of the output sort order in a sequence of two MPSM joins. Here, we consider the intermediate result to be taken as private or public input for further processing. Teams even go one step further and combine multiple operations in a single one. Thereby, teams are usually more efficient than an equivalent sequential execution of the operations by an effective preprocessing of the data.

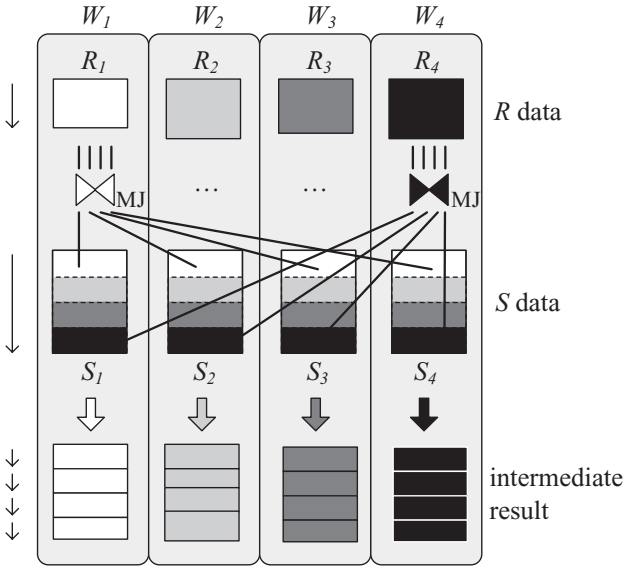


Figure 5: P-MPSM join with four workers W_i: Each worker produces four sorted (denoted by the arrows on the left) runs covering only its private input run part of the key domain and stores them locally

We present approaches for the use of MPSM in multiple join operations on the same column(s) and discuss their applicability for cases where the joins are executed on different columns.

4.1 Initial Situation

Figure 5 illustrates the situation after one MPSM join has been executed. Each of the workers produces several sorted output runs covering only part of the key domain. The intermediate result data is stored locally. A second MPSM join operator may take the intermediate result data as private or as public input depending on its size compared to the third relation to be joined. Assuming certain data distributions (in particular, similar data distributions of the inputs to the first and the second join), we can benefit from the given range partitioning of the intermediate result. When using it as private input we can omit re-partitioning the data. When using it as public input, this introduces location skew, i.e., most or all join partners of a private input run will be found in one local or remote public input run. As we showed in [AKN12], this reduces the effective number of merge joins and thus execution time.

Without any knowledge of the data distribution, however, the second MPSM join processing the intermediate result and a third relation is executed as usual. That is, the public

input is sorted, the private input is re-distributed among the workers and sorted, and the private input runs are each merge joined with all public input runs. We therefore use this scenario as the baseline and compare our approaches presented below to it.

4.2 Local Merge of Output Runs

Each worker's output consists of sorted runs within the worker's assigned key range. By merging the output runs one sorted run of the respective key range is produced. This intermediate result run can then be fed into the second join as private or as public input. Used as private input, we benefit from the given range partitioning. If there is key value skew within the inputs to the second join it is handled by computing new splitters and passing consecutive parts of the own run to other workers. As the workers' key ranges are disjoint and the data is already sorted, this only requires copying or linking run parts. When used as public input, this introduces location skew, i.e., basically only one merge join pass is required to find all join partners of a private input run as described above.

This variant requires each worker to store its complete intermediate result before it can be merged as the runs are produced subsequently. Furthermore, the sort order within one worker's output runs cannot be exploited if the second join is computed using different join column(s). It is possible, however, to sort the input chunks primarily by the first join column(s) and secondarily by the second. This requires the second join column(s) to be contained in the respective input relation (which is the case for one of the inputs) and incurs high merging overhead (potentially $n \cdot |D|$, where n is the number of workers and D is the key value dimension, runs have to be merged). In contrast to the scenario of merging in between joins on the same column(s), the merged output run then contains the complete key range, i.e., is not range partitioned.

4.3 Concatenation of Output Runs

When concatenating the workers' output runs instead of merging them, each worker obtains one sorted run covering the complete key range. This is achieved by letting each worker W_i collect the i-th output run of all workers and append those runs. In contrast to merging, concatenating theoretically does not require the intermediate result to be materialized completely in case we know the size of the intermediate result runs. However, practically this is only applicable in case of non-filtered primary key–foreign key joins. Otherwise, additional buffer might be allocated for the result runs so that they are not completely dense. As the resulting runs cover the complete key range, feeding them into the second join as private input will not be beneficial as no work can be saved. We might only exploit the given sort order to copy whole run parts during scattering instead of considering each tuple on its own. When using the intermediate result runs as public input, sorting can be omitted.

This approach cannot be adapted to work for multiple joins on different columns.

4.4 MPSM Teams

Teams prepare all inputs to be joined before starting the join phase such that both joins can then be done in one pass. For hash based join algorithms, this means partitioning all input relations, then loading the corresponding partitions of all relations and producing output tuples [GBC98]. We adapt this idea to MPSM by pre-processing the relations in the following way: we range partition the two smaller relations (i.e., treat them as private inputs) and sort chunks of the third one (public input). Then, each worker is in charge of merge-joining the two corresponding range partitioned chunks to all sorted runs of the public input.

MPSM Teams are not directly applicable for joins on different columns. In case of joins along primary key-foreign key chains with $1:N$ functionalities, it is however possible to map the join keys to new values and allow for teams processing even for different join columns. Of course, key mapping incurs extra overhead.

4.5 Pipelined Execution

The approaches above share the disadvantage that intermediate results have to be stored. This contradicts the Guy Lohman requirements for join operators. We now present a pipelined execution of two subsequent MPSM joins, which exploits the fact that each worker produces sorted output runs and that these runs can immediately be joined with the third relation. In total, each worker then executes quadratic as many merge joins as there are workers (and thus output runs), however, sorting of the third relation and the merge joins between intermediate result runs and runs of the third relation are executed in parallel with the first join processing.

The pipelined MPSM is applicable to joins on different columns. The pipelined intermediate result run parts are then sorted, thereby probably losing the range partitioning of the private input.

5 Experimental Evaluation

We implemented the MPSM join variants in C++, and compiled the query execution plans to machine code as employed in our HyPer query processor [Neu11]. In all our experiments, the input data is completely in main memory. To minimize interactions with other parts of the system, we consider the case where the input relations are scanned, a selection is applied, and then the results are joined. Thus, no indexes or referential integrity constraints (foreign keys) can be exploited during query processing. In the following, we vary the data sets regarding input sizes, join multiplicities, and data distributions to explore the application space thoroughly. Note that join multiplicities are expected multiplicities, individual tuples might have more or less join partners, or even none at all.

5.1 Platform and Benchmark Scenarios

We conduct the experiments on a Dell PowerEdge R910 Linux server (kernel 3.0.0) with 1 TB main memory and four Intel(R) Xeon(R) X7560 CPUs clocked at 2.27 GHz with 8 physical cores (16 hardware contexts) each, resulting in a total of 32 cores (and due to hyperthreading 64 hardware contexts). The machine has four NUMA regions, one for each CPU socket. Due to its large main memory of 1 TB that Dell "squeezed" into this comparatively low-cost server (ca. 40,K Euro) it has quite noticeable NUMA effects. Some micro-benchmark results with NUMA effects for this precise server are included in [AKN12].

In the experiments, each tuple consists of a 64-bit key within the domain $[0, 2^{32})$ and a 64-bit payload:

$$\{[\textit{joinkey}\text{: 64-bit, }\textit{payload}\text{: 64-bit}]\}$$

Each dataset consists of two relations R and S. R is $1600M$, the cardinality of S is scaled to be $1{\cdot}|R|$, $4{\cdot}|R|$, $8{\cdot}|R|$, and $16{\cdot}|R|$. Our datasets of cardinality $1600M\times$ (1+multiplicity) have sizes ranging from 50 GB to 425 GB, which is representative for large main memory operational BI workloads. The multiplicities between the relations R and S further cover a wide range, including not only the common cases (4, as specified for instance in TPC-H and 8 to approximate the TPC-C specification) but also extreme cases (1 and 16). The data was generated by uniformly generating 32-bit integers that were padded to 64-bit join keys. Thereby, referential integrity was not given, which results in "real" outer join result tuples. For the experiments on multi-way joins, we extended the datasets by a third relation, which is scaled in the same way.

The experiments are conducted using a parallelism of 32 (equal to the number of physical cores) if not stated otherwise.

5.2 Performance of Outer, Semi, and Anti Semi Join compared to Inner Join

We execute an equi-join on the tables:

```
SELECT count(*)
FROM R <join variant> S
WHERE R.joinkey = S.joinkey
```

This query is designed to ensure that the payload data is fed through the join while only one output tuple is generated in order to concentrate on join processing cost only. Further, we made sure that early aggregation was not used.

In Figure 6, we compare the execution time of the inner MPSM join presented in [AKN12] and the non-inner join variants for multiplicities between 1 and 16. The non-inner join variants described in Section 3 incur no (in case of right outer, semi, and anti semi joins) or only little overhead for tracking whether one tuple of the left input already found a join partner in the right input (in case of left outer, semi, and anti semi joins). The modification

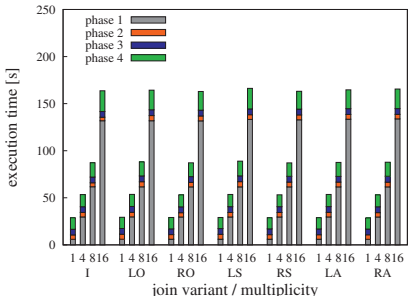


Figure 6: Inner (I), left outer (LO), right outer (RO), left semi (LS), right semi (RS), left anti semi (LA), and right anti semi (RA) join

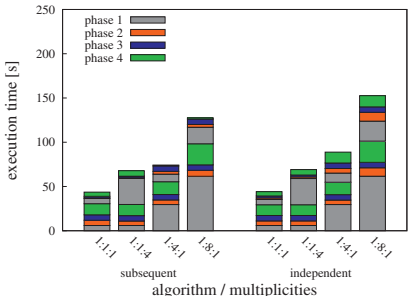


Figure 7: Comparing two subsequent MPSM join executions to two independent MPSM joins

of interpolation search required for outer and anti semi joins does not incur additional overhead. In total, we find that the performance decrease caused by the additional data structures is negligible.

5.3 Exploiting MPSM Characteristics in Complex Query Plans

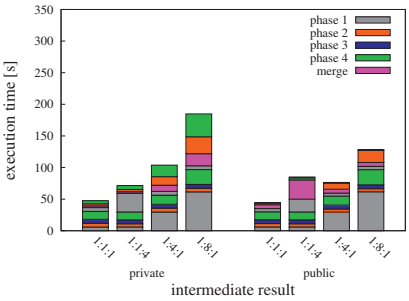
We examine the suitability of MPSM for complex query plans on the example of a three-way-join on the same join key:

```
SELECT max(R.payload + S.payload + T.payload)
FROM R, S, T
WHERE R.joinkey = S.joinkey AND S.joinkey = T.joinkey
```

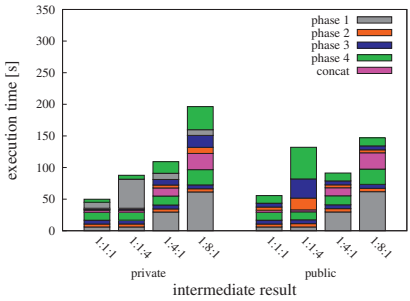
We compare the alternatives of exploiting the rough sort order of the MPSM output presented in Section 4 to the base case where two MPSM joins are executed subsequently without post-processing the intermediate result. We report experiments using the multiplicities $1:1:1$, $1:1:4$, $1:4:1$, and $1:8:1$. In a perfect scenario, an optimizer would always join smaller relations first, i.e., the third and fourth case wouldn't occur. However, we included those experiments to cover cases in which the intermediate result is smaller than the third table and in which it is larger, without modifying the key ranges or uniformity of the data distribution. In our experiments, for multiplicity $1:1:1$, the intermediate result is a little smaller than the third relation, for $1:1:4$ it is much smaller, for $1:4:1$ it is a little larger, and for $1:8:1$ it is much larger.

5.3.1 Implicit Benefits of Subsequent MPSM Joins

We first want to point out that two subsequent MPSM joins in one query plan implicitly benefit from locality of the data and range partitioning. As illustrated in Figure 5, each worker's output runs are stored locally and cover only part of the key domain. A



(a) Merging the intermediate result runs between two subsequent MPSM joins



(b) Concatenating the intermediate result runs between two subsequent MPSM joins

Figure 8: Two subsequent MPSM joins exploiting the rough sort order of the intermediate result runs

second operator (not changing the affinity of threads to cores in between) can therefore initially work on local data and (in case of a second MPSM) profits from the location skew introduced by the first operator. In Figure 7, we compare the execution times of two independent MPSM joins and two subsequent MPSM joins. The positive effect shows in the two leftmost bars in the second join's phase 2 (upper red) and phase 3 (upper blue) execution times and in the two rightmost bars in the second join's phase 1 (upper gray) and phase 4 (upper green) execution times.

5.3.2 Merge and Concatenation of Intermediate Result Runs

We applied merge and concatenation to the intermediate result runs and then fed the result into the second join, once as private input and once as public input. As shown in Figure 8, the findings in [AKN12] that the smaller relation should always be picked as the private input were confirmed. This is due to the efficient scans on local memory compared to remote memory. In the following, we therefore assume the optimizer to correctly assign private and public input roles to the smaller, respectively larger relations after the first join.

5.4 Comparison to Baseline

In Figure 9, we compare the total execution time of two MPSM joins using the approaches described in Section 4 to that of two subsequent MPSM executions without any additional processing of the intermediate result. Overall, the simple execution of two MPSM joins shows the best performance.

Merging the intermediate result runs to use them as private input to the second join allows for omitting the sort phase (in case of uniform distribution). However, merging shows approximately the same performance as the optimized sorting technique of MPSM combining Radix sort and Introsort. When using the merged runs as public input this has the same positive effect as location skew [AKN12]. Two subsequent MPSM joins benefit from location skew as well (see Section 5.3.1) and thus has the same performance.

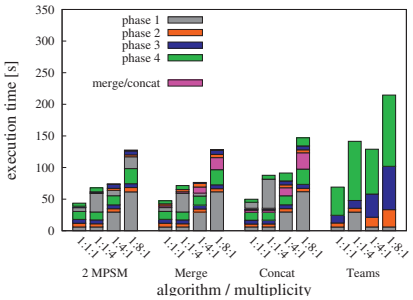


Figure 9: Performance comparison of two subsequent MPSM joins without post-processing of the intermediate result (2 MPSM), with merging each worker's runs of the intermediate result (Merge), and with concatenating the intermediate result runs (Concat)

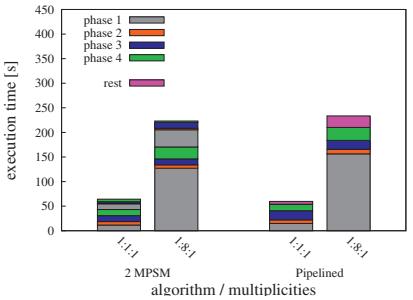


Figure 10: Performance comparison of two subsequent MPSM joins without post-processing of the intermediate result (2 MPSM) and pipelined MPSM (16 threads)

Concatenating the result runs is less beneficial when using the outcome as private input to the second join. This is because those runs cover the complete key range and thus must be range-partitioned as usual. When using the intermediate result runs as public input, the performance even degrades because concatenating the runs from multiple remote NUMA partitions is even slower than sorting the own runs within the local partition.

The MPSM Teams are not competitive at all as three-way-merge joining incurs a very high overhead. We conclude that subsequent merge joins are more efficient than three-way-merge joins.

5.5 Pipelined MPSM

For the evaluation of pipelined MPSM we instantiate 16 threads to process the first join and another 16 threads to which the intermediate results are piped. The total number of threads thus equals the number of physical cores on our server.

Figure 10 shows the comparison of two subsequent MPSM joins to pipelined MPSM. Here, "rest" denotes the time from the completion of the first join until the second join execution finishes. Due to the additional bandwidth incurred by the parallel processing of the first join and operations (sorting of the third relation and merge joining) of the second join, the performance of the first join degrades slightly. Overall, there is no significant performance difference between the two three-way-join variants.

6 Conclusions

In this work, we developed the algorithmic details of MPSM for other join variants, i.e., outer, semi, and anti semi joins. We also worked on exploiting the rough sort order that MPSM inherently generates due to its range-partitioned run processing. We compared the effect of merging and concatenating intermediate result runs to MPSM Teams execution processing a three-way-join in one operation. Furthermore, we investigated a pipelined version of MPSM. The experimental evaluation revealed that the efficient sort and merge phases of MPSM leave almost no room for improvement by replacing sort by merge or concatenation or by overlapping the merge phase with subsequent operations. Although some of the proposed optimizations for MPSM in complex query processing are applicable for multiple joins on different columns, we are confident that executing two subsequent MPSM operations results in the most robust and efficient performance.

References

[AKN12] Martina-Cezara Albutiu, Alfons Kemper, and Thomas Neumann. Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems. *PVLDB*, 5(10):1064–1075, 2012.

[BMK09] Peter A. Boncz, Stefan Manegold, and Martin L. Kersten. Database Architecture Evolution: Mammals Flourished long before Dinosaurs became Extinct. *PVLDB*, 2(2):1648–1653, 2009.

[CKKW00] Jens Claussen, Alfons Kemper, Donald Kossmann, and Christian Wiesner. Exploiting Early Sorting and Early Partitioning for Decision Support Query Processing. *The VLDB Journal*, 9:190–213, 2000.

[FCP+11] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. SAP HANA Database: Data Management for Modern Business Applications. *ACM SIGMOD Record*, 40(4):45–51, 2011.

[GBC98] Goetz Graefe, Ross Bunker, and Shaun Cooper. Hash Joins and Hash Teams in Microsoft SQL Server. In *VLDB*, pages 86–97, 1998.

[Gra93] Goetz Graefe. Query Evaluation Techniques for Large Databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.

[KN11] Alfons Kemper and Thomas Neumann. HyPer: A Hybrid OLTP&OLAP Main Memory Database System based on Virtual Memory Snapshots. In *ICDE*, pages 195–206, 2011.

[LPP+13] Yinan Li, Ippokratis Pandis, Ippokratis Pandis, Vijayshankar Raman, and Guy Lohman. NUMA-aware algorithms: the case of data shuffling. In *CIDR*, 2013.

[Neu11] Thomas Neumann. Efficiently Compiling Efficient Query Plans for Modern Hardware. In *VLDB*, 2011.

[SAC+79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access Path Selection in a Relational Database Management System. In *SIGMOD*, pages 23–34, 1979.

[Vol10] VoltDB LLC. VoltDB Technical Overview, Whitepaper, 2010. `http://voltdb.com/_pdf/VoltDBTechnicalOverviewWhitePaper.pdf`.

Taking the Edge off Cardinality Estimation Errors using Incremental Execution

Thomas Neumann
Technische Universität München
Munich, Germany
neumann@in.tum.de

Cesar Galindo-Legaria
Microsoft
Redmond, WA, USA
cesarg@microsoft.com

Abstract: Query optimization is an essential ingredient for efficient query processing, as semantically equivalent execution alternatives can have vastly different runtime behavior. The query optimizer is largely driven by cardinality estimates when selecting execution alternatives. Unfortunately these estimates are largely inaccurate, in particular for complex predicates or skewed data.

We present an incremental execution framework to make the query optimizer more resilient to cardinality estimation errors. The framework computes the sensitivity of execution plans relative to cardinality estimation errors, and if necessary executes parts of the query to remove uncertainty. This technique avoids optimization decisions based upon gross misestimation, and makes query optimization (and thus processing) much more robust. We demonstrate the effectiveness of these techniques on large real-world and synthetic data sets.

1 Introduction

In most of today's database systems data access is hidden behind declarative query interfaces. The user (or a query generator) submits a query that describes the users intent, and the database system chooses the most efficient way to produce the requested data. This query optimization step, i.e., the transformation from a declarative query into an imperative execution plan, can have drastical impact on the performance of query processing. It is largely driven by a cost model that predicts how expensive certain operations are, and as such is used to find the most efficient execution alternatives for the given query. One of the most central components of the cost model is the cardinality estimation that predicts the result sizes of intermediate results.

What is interesting about cardinality estimation is that on one hand it has a very large impact on the selected execution plan, and on the other hand it tends to produce estimates that are very far off from time to time. Some people even claim that cardinality estimation is so brittle and unreliable that one should try to avoid relying on cardinality estimates at all (see for example [Cha09] for a discussion). We are a bit more optimistic and believe that for many queries and data sets the current cardinality estimation methods work reasonably well. Estimating the result cardinalities for the well known TPC-H benchmark for example is relatively easy, as both queries and data distributions are simple. Real-world data sets are more difficult for estimation purposes, but even their estimates are usually not that bad, at least for base tables. On the other hand, bad cardinality estimates that can be orders of magnitude off are an undeniable reality, either because the data is heavily skewed or because

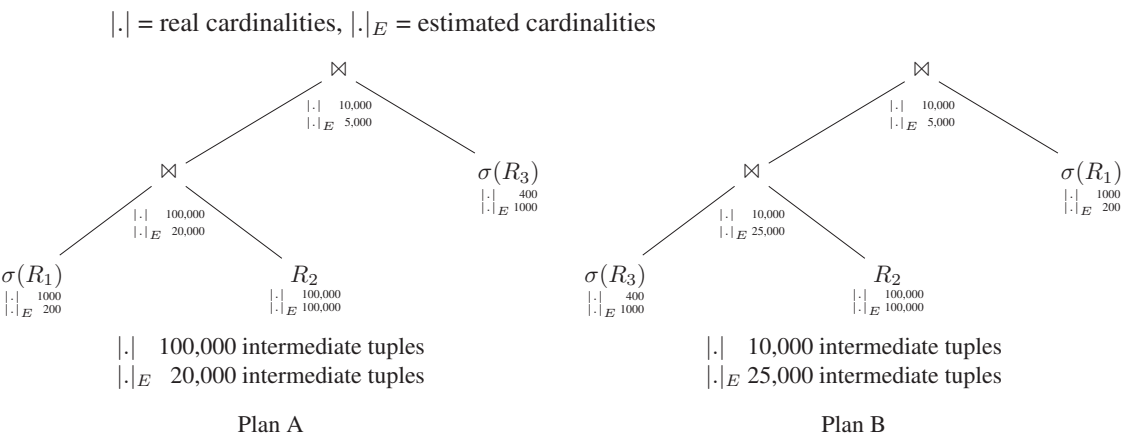


Figure 1: Impact of Cardinality Estimation Errors

the query predicates are complex and/or correlated. For example, the result cardinality of the following (admittedly constructed) SQL query over the TPC-H schema is quite hard to estimate:

```
select *
from lineitem,order
where l_orderkey=o_orderkey and
      log(l_extendedprice)>4 and
      log(o_totalprice-l_tax)>4
```

Such complex filter predicates are uncommon, but they do occur, and they are almost guaranteed to lead to poor cardinality estimates. Unfortunately the infrequently occurring bad estimates are much more noticeable than the more common good estimates, as they can lead to very poor execution plans. This is illustrated by an example in Figure 1. It shows two possible execution plans for a join query with three relations, annotated with real result cardinalities and estimated result cardinalities, where the estimates for two leaf nodes are somewhat off. When we look at the estimated cardinalities both plans seem to be very similar, with Plan A being slightly preferable due to the lower number of intermediate results. But when we look at the real number we see that in reality Plan A is much worse, producing 10 times the number of intermediate results of Plan B. Note that this mistake (preferring Plan A over Plan B) was not caused by a single estimation error, but only by combination of two unrelated errors. This is a quite common scenario, and implies that it is not sufficient to look at the accuracy of individual estimates, but that the whole execution query plan has to be taken into account.

The main idea of this work is to mitigate the effects of these cardinality estimation errors by using *incremental execution*: The query optimizer starts optimizing the query, and when it notices that cardinality estimation errors may lead to suboptimal execution plans it executes parts of the query to get the correct cardinalities. We give a formal description of this approach later, but roughly said the query optimizer executes the query fragments where the cardinality estimates are uncertain but important for the total plan choice. For the example

in Figure 1 this means that the optimizer might decide to execute $\sigma(R_1)$ or $\sigma(R_3)$ (but not both, as we will see in Section 3) to get the cardinalities right, and will then be able to pick the right plan. This incremental execution paradigm makes the optimization process much more robust against estimation errors, as estimation errors can be corrected during the optimization process now. Incremental execution induces some costs, of course, we therefore use incremental execution only when our algorithm decides that it is necessary for the given query.

The main contributions of this paper are as follows:

1. we present an algorithm that decides if incremental execution is necessary for a given query, and schedules the incremental executions as needed.
2. we show that this algorithm is guaranteed to lead to optimal execution plans under certain constraints, and is theoretically sound even if optimality cannot be guaranteed.
3. we present an error propagation framework that allows for deriving the error bounds necessary for deciding about incremental execution.
4. we demonstrate the validity of these techniques inside a commercial database system using large real-world and benchmark workloads.

The rest of the paper is structured as follows: First, we discuss related work in Section 2. Then, we introduce the incremental execution algorithms in Section 3, and show that they are theoretically sound. After that, we explain the error propagation framework necessary for incremental execution decisions in Section 4, followed by techniques to reduce the runtime costs of incremental execution. Finally, we give experiment results for real-world and benchmark workloads in Section 6. Conclusions are drawn in Section 7.

2 Related Work

Cost based query optimization has already been pioneered by System R [SAC+79], and accordingly cardinality estimation is a very well studied field. A comprehensive overview over the standard cardinality estimation techniques is given in [Ioa03], but these techniques usually do not consider the problem addressed in this paper, namely that cardinality estimates will be wrong at some point. Some histogram techniques acknowledge the fact that they are error prone and try to minimize the error, for example V-Optimal histograms [JKM+98] or some wavelet construction mechanisms like [GK05], but regarding the impact on the resulting execution plan they basically remain best-effort. A recent work tries to minimize the effect that cardinality-misestimations can have on query execution [MNS09], but it does not address the issue of recovering from large estimation errors. Their motivation is very similar to ours, but the approach is very different. They construct histograms in a way that minimizes the multiplicative error, which is very useful for the error computation discussed in Section 4, but they do not look at the issues caused by large estimation errors. Thus they delay the moment where misestimations will lead to bad plans, but when it happens they cannot cope with the estimation error. Therefore it makes sense to use their histograms in combination with our approach (as then the error estimates will be tight and low), as both approaches are really complementary. There has been some effort to cope with estimation errors within the optimizer, usually in the form of a feedback loop (see e.g., [SLMK01, LLZZ07]). The fundamental problem of these approaches is that

the query feedback is only available after the query was executed. The next query might profit from the new information, but the current query will still suffer from estimation errors. Furthermore some application scenarios have a lot of unique queries, which makes exploiting query feedback difficult.

A recent work that is similar in spirit to our incremental execution approach is the ROX optimizer [KBMvK09] for XQuery processing. It tries to address the cardinality estimation problem by relying on estimates only for the next step: When faced with a join ordering decision, it uses sampling to predict the join selectivity, executes the most promising join, materializes the result, and then optimizes again. As a consequence it always operates using exact cardinalities (as all inputs are materialized), the only uncertainty is the join selectivity (which is computed using sampling). The ROX optimizer fits nicely into the containing MonetDB framework, which materializes all intermediate results anyway, but in general it seems to be wasteful to materialize everything. The experiments in [KBMvK09] used a relatively small XML data set, but we found in our experiments that materializing all intermediate results causes a significant overhead when the intermediate results are larger than main memory (see Section 6). Furthermore the join ordering strategy proposed by ROX is overly greedy (being basically equivalent to [Feg98]), which can lead to suboptimal plans even without cardinality estimation errors.

While query optimizers traditionally have not done much to cope with cardinality estimation errors (besides improving the general accuracy, of course), there has been quite a lot of work on the runtime side. A good overview over these techniques is given in [DIR07]. These adaptive runtime techniques try to change the execution strategy according to the observed data distributions. The most extreme example are Eddies [AH00, TD03, RDH03], which – at least in principle – can adjust the data flow for each individual tuple. More conventional approaches [CG94, KD98, MRS+04] re-optimize parts of the query during query execution as they observe the cardinalities of intermediate results. The main problem there is deciding when to re-optimize. First, deciding at which point in the execution plan the cardinality should be checked (which usually requires materialization), and second, computing for which cardinalities the current execution plan should be changed. These are non-trivial issues, and the current adaptive runtime techniques have no satisfying solutions for them.

3 Using Incremental Execution

We now discuss how to incorporate incremental execution into query optimization, or more precisely, into plan construction. First, the name ”incremental execution” is not completely unambiguous, as there are multiple ways one could use an incremental execution paradigm. One extreme would be executing each query operator individually and then re-optimize after each step [KBMvK09]. But such a scheme seems wasteful, as incremental execution induces overhead, and, even more importantly, enforces serial execution of plan fragments. We therefore propose to use incremental execution only when needed, i.e., only to prevent plan construction mistakes caused by gross misestimates.

3.1 General Approach

We use the following incremental execution model: First, we construct the optimal execution plan using our cost model. (The plan optimality is relative to this cost model, of course). Then we identify plan fragments where cardinality estimation errors might have lead to wrong plan decisions higher up. Slightly simplified, we look for plan fragments where knowing the correct cardinality is important. We then execute the plan fragment, materializing the result, and thus getting the exact cardinality. If the new cardinality indicates that we have to choose a different plan we re-optimize, using the materialized result as available input. The advantage of this model is that we only execute plan fragments that we would have executed anyway, as they are parts of the (presumably) optimal plan. Furthermore we might decide not to execute any plan fragment if the cardinalities are not critical to the choice of plans.

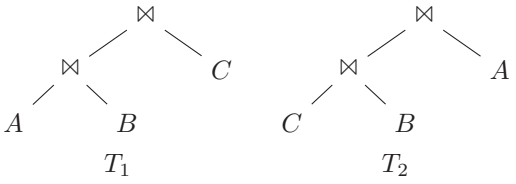
To incorporate incremental execution into query optimization we have to provide answers to two questions:

- given an optimal execution plan (relative to a cost model) and the cardinality estimation data, can we decide that the plan is indeed optimal?
- if not, which plan fragment should we execute to decide optimality afterwards?

As an illustrational example consider the following query fragment:

```
select *
from   (...)  A, (...)  B, (...)  C
where  A.x=B.x and B.y=C.y
```

Note that A, B and C could be anything from base relations to complex subqueries. When disregarding cross products, there are only two possible join trees for joining A, B and C (ignoring commutativity):



We now assume that T_1 was chosen as optimal plan (according to the cost model). This means that based upon the cardinality estimates, the costs of T_1 are lower than the costs of T_2. The interesting question is now, how much do the cardinalities have to change such that T_2 becomes cheaper than T_1. This is equivalent to computing the maximum cardinality estimation error for which T_1 is still guaranteed to be optimal.

The exact computations depend on the cost function. We use the cost function that minimizes the sizes of intermediate results here, but other cost functions could be used as well:

$$C(T) = \begin{cases} 0 & \text{if } T \text{ is a relation } R_i \\ |T| + C(T_1) + C(T_2) & \text{if } T = T_1 \bowtie T_2 \end{cases}$$

When using this cost function for T_1 and T_2, we notice that the optimality of T_1 does not depend on $|B|$. Due to the structure of the query B must occur in the first join, and its cardinality does not affect the optimal relative order of A and C. Therefore there is no point in using incremental execution for B, as its outcome would not change the constructed execution plan. The sizes $|A|$ and $|C|$ do have an impact on the plan. We stated that T_1 is supposed to be cheaper than T_2. Now assume that we underestimated $|A|$ by a factor α_A and overestimated $|C|$ by a factor α_C. We denote the join selectivities with f_{AB} and f_{BC}. Then, we can formulate the fact that T_1 is cheaper than T_2 as follows:

$$\begin{aligned} C(T_1) &\leq C(T_2) \\ f_{AB}(\alpha_A|A|)|B| &\leq f_{BC}|B|(\frac{1}{\alpha_C}|C|) \\ \alpha_A\alpha_C &\leq \frac{f_{BC}|C|}{f_{AB}|A|} \end{aligned}$$

In other words, as long as our estimation errors α_A and α_C are below the threshold $\frac{f_{BC}|C|}{f_{AB}|A|}$ we know that we picked the right join order and do not need incremental execution. Intuitively, this mimics the fact that large differences in relation sizes lead to clear join orders (the threshold is large), while small differences make it much easier to make mistakes (the threshold is small). One crucial part in this reasoning is the computation of estimation errors, of course, see Section 4 for details.

If the errors are above the threshold we might potentially make a mistake, and it therefore makes sense to use incremental execution. We have multiple plan fragments that could potentially be executed (in the example A and C), and only one of these must be picked. In principle any of them could be chosen, and different selection strategies appear to be reasonable (e.g., minimizing costs or minimizing materialized results). In our experiments we found that a very good strategy seems to be choosing the plan fragment that has the maximum estimated error associated with it. In the example we would execute A if $\alpha_A \geq \alpha_C$, and C otherwise. By executing the largest error first we remove the biggest chunk of uncertainty from the optimization problem, and therefore expect to perform less incremental executions in total. We also experimented with executing the smallest intermediate results first (as these are cheap to materialize), but this performed worse in our experiments.

3.2 Algorithms

After the high-level discussion of the incremental execution idea we now discuss the concrete algorithms. The basic framework is shown in Figure 2. Given a query Q, it first optimizes the query using the cost model, then checks if some parts of the resulting execution plan P warrant incremental execution. If not, it knows that P is indeed the best plan and returns it. Otherwise it examines all incremental execution candidates and picks the candidate P_C that has the largest estimation error α. This plan fragment is then executed, its result is stored in a temporary table, the query is updated to use the temporary table instead of the plan fragment, and the query is optimized again (using the cardinalities

OPTIMIZEINCREMENTAL(Q)
while true
  P=OPTIMIZE(Q)
  C=FINDINCREMENTALEXECUTIONCANDIDATES(P,P)
  if $C = \emptyset$
    return P
  $\hat{C} = \{P | P \in C \wedge \nexists P' \in C : P' \in P\}$
  $P_C = \arg\max_{P \in \hat{C}} \alpha(P)$
  execute P_C, store the result in R
  replace P_C with $scan\ R$ in Q

Figure 2: Using incremental execution

derived from the incremental execution). Note that it is not necessary to re-optimize the full query again, all parts that are independent of P_C can be reused directly. Further note that depending on the structure of the query the optimization time decreases exponentially with the number of operators contained in P_C, as these operators are removed from the search space. The repeated optimization is therefore not time critical in practice.

The main difficulty in using incremental execution is identifying plan fragments where the estimation errors can cause plan changes. The algorithm for finding candidates that can affect the join ordering is shown in Figure 3. It recursively traverses the execution plan until it reaches a join operator. For each join, it examines the "ancestors" of the join (i.e., the operators being executed after the join), and checks if would have been possible to execute them instead of the current join. This is illustrated in Figure 4. We assume that A, B, C, and D are complex subqueries. The join $\bowtie_3$ (which joins A and B) has two ancestors, $\bowtie_1$ and $\bowtie_2$. When examining these joins we see that A could have been joined with D instead, and B with C. To be sure that $\bowtie_3$ was the right choice we must therefore first compute

$$\frac{f_{AD}|D|}{f_{AB}|B|} = \frac{90}{5} = 18$$

and compare it with

$$\alpha_B \alpha_D = 1.2 * 1.5 = 1.8.$$

Here the error was less than the threshold, so A should indeed be joined with B first. Then, as we could also join B with C, we compute

$$\frac{f_{BC}|C|}{f_{AB}|A|} = \frac{20}{1} = 20$$

and compare it with

$$\alpha_A \alpha C = 10 * 2.5 = 25.$$

Here the error is larger, i.e., we are not sure that $\bowtie_3$ was indeed the right choice. Both A and C are candidates for incremental execution, we will execute A first to remove the most uncertainty from the system. Similar computations are performed for the other join operators. The actual computations get a bit more complicated as the input of the other joins are no longer base expressions, but the general principle is the same.

FINDINCREMENTALEXECUTIONCANDIDATES(P,P_{root})
$C = \emptyset$
for each P' input of P
  $C = C \cup$FINDINCREMENTALEXECUTIONCANDIDATES(P',P_{root})
if $P = T_1 \bowtie T_2$
  $C_1 = \emptyset$, $C_2 = \emptyset$
  for each P' ancestor of P in P_{root}
    if $P' = T_1' \bowtie T_2' \wedge P \in T_1'$
      if T_1 could be joined with T_2'
        $C_1 = C_1 \cup \{T_2'\}$
      if T_2 could be joined with T_2'
        $C_2 = C_2 \cup \{T_2'\}$
    if $P' = T_1' \bowtie T_2' \wedge P \in T_2'$
      if T_1 could be joined with T_1'
        $C_1 = C_1 \cup \{T_1'\}$
      if T_2 could be joined with T_1'
        $C_2 = C_2 \cup \{T_1'\}$
  for each $T' \in C_1$
    if $\alpha(T_2)\alpha(T') > \frac{f_{T_1T'}|T'|}{f_{T_1T_2}|T_2|}$
      $C = C \cup \{T_2, T'\}$
  for each $T' \in C_2$
    if $\alpha(T_1)\alpha(T') > \frac{f_{T_2T'}|T'|}{f_{T_2T_1}|T_1|}$
      $C = C \cup \{T_1, T'\}$
return C

Figure 3: Identifying candidates

We only considered using incremental execution to be certain about join ordering, as this usually has the largest impact on the overall query performance, but similar computations could be added for other operators, too: The optimizer must check if the cardinality estimation error could lead to a plan change, and execute the relevant plan fragments if the cardinality has to be known exactly. Note that we do not assume that the query is join-only, i.e., other operators like selections or group-by can occur within the operator tree (in particular group-by statements are frequently the cause for severe estimation errors), but we currently only consider plan changes affecting the join order when deciding about incremental execution. These were the most critical parts of the queries in our experiments with real-world queries (see Section 6).

3.3 Theoretical Foundation

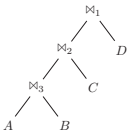
The candidate selection algorithm in the previous section uses some simplifying assumptions and some heuristical criteria, but it was derived from a solid theoretical basis. Now we will therefore explain the simplifying assumptions, and discuss the optimality guarantees that still hold even in the presence of these assumptions.

The most severe simplification used by the algorithm is that it disregards the estimation

query graph:

example sizes: $|A| = 20\ |B| = 100\ |C| = 200\ |D| = 100$
selectivities: $f_{AB} = 0.05\ f_{BC} = 0.1\ f_{AD} = 0.9$
errors: $\alpha_A = 10\ \alpha_B = 1.2\ \alpha_C = 2.5\ \alpha_D = 1.5$

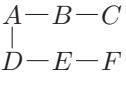
execution plan:



C_i sets for $\bowtie_1$: $C_1 = \emptyset\ C_2 = \emptyset$
C_i sets for $\bowtie_2$: $C_1 = \{D\}\ C_2 = \emptyset$
C_i sets for $\bowtie_3$: $C_1 = \{D\}\ C_2 = \{C\}$

Figure 4: Example plan with computation results from Figure 3

query graph:



execution plan:

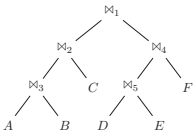


Figure 5: Bushy execution plan

error for the top-most join selectivities. In the example, the algorithm decides about the join of A with B or D by comparing $\alpha_B\alpha_D$ with $\frac{f_{AD}|D|}{f_{AB}|B|}$. The comparison itself is theoretically justified, but it implicitly assumes that the values f_{AD} and f_{AB} are known exactly, while in reality the join selectivity will only be known approximately. As a consequence the algorithm could decide that the join order is correct and that no incremental execution is necessary, as the errors in join inputs are not severe enough, even though the error on the join selectivity itself caused a bad join order. This is unfortunate, but unavoidable. The error on the join selectivity can only be known after executing the join itself, and then it is too late to change the join order. This observation is also the reason our algorithm pretends that the top-most join selectivities are known exactly. Note that we only assume this for an individual join when considering plan alternatives. If the input of the join contains other join operators we take the estimation error of these join predicates into account (see Section 4). We might therefore detect an estimation error one operator too late, but in practice this does not seem to be a severe limitation.

The second simplification concerns the way the algorithm finds join alternatives. For each join operator the algorithm checks its ancestors to find alternative join partners, producing

$O(n)$ join alternatives (where n is the number of base expressions). When only considering linear join trees (e.g., left-deep join trees) the resulting C_i lists contain precisely the possible join alternatives. However when considering bushy join trees, there can be an exponential number of join alternatives, which are not all considered by the algorithm. This is illustrated in Figure 5: When examining the join $\bowtie_3$, the algorithm will determine that A could be joined with B and with $(D \bowtie_5 E) \bowtie_4 F$, even though there are more alternatives (like for example joining only with D). However this is mainly a theoretical concern, and does not not affect optimizations based upon data flow minimization like minimizing the sizes of intermediate results. This can be seen by symmetry arguments. First, the algorithm considers all joins contained in the execution plan, in particular also $\bowtie_4$ and $\bowtie_5$. If these contain uncertainties that warrant incremental execution it will execute them first, before considering executing $(D \bowtie_5 E) \bowtie_4 F$, as sub-trees are always considered before super-trees. If the algorithm does not use incremental execution we are certain about the relative order of D, E, and F, as we always consider the best plan under the given cost model. But then it must always be beneficial to perform $\bowtie_4$ and $\bowtie_5$ before $\bowtie_1$, at least concerning the data flow, as otherwise the optimizer would have pulled the joins up. It is therefore usually not necessary to consider the exponential number of alternatives.

These two limitations are concessions to the practical usage of incremental execution, as lifting them would require unreasonably large processing costs (in particular for the first limitation). The second limitation is largely harmless, the first one is a bit unfortunate, but does not seem to be easily avoidable. However, we will now show that the fundamental construction of the incremental execution algorithm is sound, in the sense that it can guarantee optimal execution plans for certain classes of queries. This can be seen by considering the IK/KBZ family of algorithms [IK84, KBZ86]. These algorithms construct the optimal linear join trees for acyclic join queries with ASI cost functions (e.g., minimizing the sizes of intermediate results). They roughly work in two steps, where the first constructs the precedence graph that describes which join has to be performed before another join becomes possible (to avoid cross products). The second step sorts the join operators by rank (i.e., the perceived "benefit" of the join) and merges them together according to the rank and the restrictions imposed by the precedence graph.

As the IK/KBZ algorithms construct linear trees, it is sufficient to identify an execution plan with a sequence of relations (joining from left to right). Then, the rank function of a sequence S is defined as

$$rank(S) = \frac{T(S) - 1}{C(S)}$$

where $T(S)$ is the result size change (i.e., the selectivity relative to the first relation) and $C(S)$ are the costs per input tuple. For a relation R_j that is ranked relative to a precedence root R_i this results in a rank of

$$rank(R_i R_j) = \frac{|R_j| f_{R_i R_j} - 1}{|R_j| f_{R_i R_j}} = 1 - \frac{1}{|R_j| f_{R_i R_j}}.$$

Using this formulation it is clear that if one relation R_j has a lower rank than a relation R'_j, and the cardinality estimation errors $\alpha_{R_j}, \alpha_{R'_j}$ are bound by

$$\alpha_{R_j}\alpha_{R'_j} \leq \frac{f_{R_iR_j}|R_j|}{f_{R_iR'_j}|R'_j|}$$

then the relative rank of R_j and R'_j remains the same. Or, in other words, using our incremental execution algorithm guarantees finding the optimal execution plan in this case.

For general queries we cannot guarantee optimality, but as illustrated above there is strong evidence that the algorithm is sound and that it will find the critical join alternatives. For linear execution plans we could even derive an optimality bound, as the worst case depends purely on the limitations of join selectivity estimation errors.

4 Error Propagation

The incremental execution framework from Section 3 needs to know the error, i.e., the estimation uncertainty, associated with each cardinality estimation. In most database systems this information is not readily available, we therefore now discuss how we can estimate and propagate error bounds within execution plans.

At a first glance computing the estimation error seems to be nearly as difficult as the estimation itself; at least the two problems are closely related. However in practice the error estimation is a much more good-natured problem, as the error estimate does not have to be as accurate as the cardinality estimate. Misestimations of the error do have consequences, of course. Overestimating the error can lead to unnecessary incremental executions, while underestimating the error can lead to suboptimal executions plans. But overall a bad error estimate usually has much less severe consequences than a bad cardinality estimate. Furthermore it is much easier to learn the typical estimation error from observed query executions than to learn the cardinalities themselves (in particular since the error bound does not have to be tight). Therefore it is relatively easy to integrate the error propagation techniques explained below into an existing database system.

Before discussing the error propagation framework we first have to define the error metric that we use. The incremental execution framework from Section 3 needs to know the factor α that gives the maximum derivation from the real cardinality. Given an algebraic expression E we therefore define the error function $\alpha(E)$ as follows:

$$\begin{aligned} |E| &:= \text{result cardinality of } E \\ |E|_E &:= \text{estimated result cardinality of } E \\ \alpha(E) &:= \frac{\max(|E|, |E|_E)}{\min(|E|, |E|_E)} \end{aligned}$$

Thus an error of $\alpha(E) = 2$ means that the estimate is at most twice as large as the result cardinality (or at least half the size of the result cardinality). Note that we implicitly assume that $|E| \geq 1$ and $|E|_E \geq 1$, i.e., we assume that the query result is non-empty. Empty queries are a special case that has to be handled (and detected) efficiently by the runtime system, the query optimizer always assumes non-empty results.

$$\begin{aligned}
\alpha_S(R_i) &= 1 \\
\alpha_S(\sigma_p(E)) &= \alpha_S(E)\alpha_S(p) \\
\alpha_S(E_1 \times E_2) &= \alpha_S(E_1)\alpha_S(E_2) \\
\alpha_S(E_1 \bowtie_p E_2) &= \alpha_S(E_1)\alpha_S(E_2)\alpha_S(p) \\
\alpha_S(\Gamma_{agg}(E)) &= \alpha_S(E)\alpha_S(agg) \\
\alpha_S(\rho a \to b(E)) &= \alpha_S(E) \\
\ldots &\quad \ldots
\end{aligned}$$

Figure 6: Error propagation within an operator tree

Using this error metric we now have to derive error bounds for algebraic expressions. We distinguish two kinds of error bounds, first the *maximum* error α_M that can be derived from schema information, and second the *structural* error α_S that stems from predicate analysis and error propagation within the the operator tree. Both metrics provide bounds for the real error, thus

$$\alpha(E) = \min(\alpha_M(E), \alpha_S(E))$$

The maximum error is derived from the known cardinality bounds that are maintained in most query optimizers anyway. Using the upper cardinality bound $|E|_E^{max}$ and the lower cardinality bound $|E|_E^{min}$ we can define the maximum error as

$$\alpha_M(E) := \max(\frac{|E|_E}{|E|_E^{min}}, \frac{|E|_E^{max}}{|E|_E}).$$

This bound is a hard error bound, but in most cases it is not very useful as $|E|_E^{max}$ tends to be very large (it assumes that all predicates always match etc.). Still, it is a useful bound, as in some cases it is tight. The simplest example are (sub-)queries of the form

select count(*) **from** ... **where** ...

Here we know that the result cardinality is exactly one tuple. Other examples include constraints on key attributes, group by queries with grouping on attributes with known domains (e.g., keys), etc. In these cases the maximum error that is purely derived from schema information will give useful bounds. Note that α_M is really a bound, not a function that distinguishes exactly known cardinalities from uncertain cardinality. In particular $\alpha_M(E)$ can be > 1 even in the examples mentioned above due to additional predicates, but it tends to be a tight bound in these cases.

For general queries the maximum error will be very loose, we will have to examine the algebraic expressions themselves to get tighter bounds. The basic principles of error propagation are shown in Figure 6. For base relations the error is 1, as the cardinality is usually known exactly. Selections introduce an error, therefore the total error is the error of the input times the error caused by the selection predicate. We will discuss predicates below. Cross products are simple, as the two input errors can simply be multiplied here (this follows naturally from the definition of the Cartesian product). A join is a cross product followed by a selection, the error propagates accordingly. Group by operators (Γ) are a bit

special. In principle they are similar to selections, but in practice it is often hard to predict the number of resulting groups. This will be discussed further below. Finally, there is a number of operators that do not affect the cardinality at all (e.g., ρ), these just propagate the errors up. As we can see, the main problem here is computing the error induced by individual predicates.

For simple predicates of the form $x = 7$ or $x \geq 4$ this error can be computed relatively easily. Database systems usually maintain statistical synopses like histograms about their data, and we can give error bounds for these: When constructing a histogram bucket there will be minimum frequency f_{min} of values within the bucket, a maximum frequency f_{max} and the average frequency f_{avg} that is usually used when estimating the result cardinality. The maximum error induced by this bucket is therefore

$$\max(\frac{f_{avg}}{f_{min}}, \frac{f_{max}}{f_{avg}}).$$

During histogram construction we can derive this value for each bucket, and then remember the maximum over all buckets as the maximum error induced by this histogram.

This gives a hard bound, but might be too large in practice. As we maintain the maximum error, this value is very susceptible to outliers. Ideally we would use histograms that minimize the maximum error [MNS09], or, if we have to cope with existing systems that cannot easily change their histogram implementation, we would not take the maximum error but the 95% quantile or use some similar dampening technique. The later is not as satisfying from a theoretical point of view, as we might now miss incremental execution candidates, but it can greatly reduce the number of incremental executions, as most of the time the error will not be that large.

For more complex predicates we can start by combining estimated errors for simple predicates, which works reasonably well for $\wedge$ and $\vee$, but at some point we must fall back to guessing. This is similar to selectivity estimation, which will also have to fall back to guessing at some point. Interestingly this means that in the cases where we have to guess, we basically know that the estimation error will be high, as the selectivity estimation guesses, too. Similar for *group by* operations, either we have domain information available, in which cases we can derive an error bound (even though it will frequently be large [CCMN00]), or both the error estimate and the selectivity estimation will have to be guesses. In experiments we found that guessing the error is much easier, and in fact reasonable error bounds for complex predicates can be derived by examining available query feedback. The best way to address complex predicates is therefore probably a statistics warehouse based upon query feedback similar to [MMK+05].

5 Reducing the Runtime Costs

Incremental execution is a very useful technique, as it limits the effects of estimation errors, but it comes with a cost. Even though the experiments in Section 6 show that the incremental execution frameworks materializes only a few intermediate results, the theoretical worst case would be that (nearly) every operator materializes all of its inputs, roughly increasing the execution time by factor of two. In this section we therefore present

techniques to reduce the runtime costs of incremental execution. Note though that these techniques are strictly optional. Our experiments show that even a database system without specific runtime support can greatly benefit from incremental execution.

One very nice property of incremental execution is that the mechanism only executes plan fragments that would have been executed anyway. This means that even in the worst case, where incremental execution does not correct a single mistake resulting from estimation errors, the overhead stems only from materialization. On the other hand, materialization costs can be quite high, in particular when looking at pipelining plans: Most database systems try to avoid materializing and copying intermediate results as much as possible, passing data between operators in a pipelined fashion [Lor74]. These system distinguish pipelining operators, i.e., operators that simply pass their input data along, and pipeline breaks, i.e., operators that explicitly materialize and copy data. For pipeline breakers the additional costs of incremental execution are not that high, as they materialize anyway, but for pipelining operators materializing their input can add a noticeable overhead.

Fortunately this problem is not too severe in practice. As mentioned in Section 3, we concentrate on using incremental execution for join operators, and most join implementations are pipeline breakers anyway. We will discuss techniques for speeding up incremental execution in the presence of joins below. Pipelining operators are usually much more light-weight, the most prominent example is the selection operator. Here, incremental execution could potentially be expensive, but in fact it would never be used. As discussed in Section 3, incremental execution is only triggered if the cardinality of the intermediate result could affect the plan choice. This is not the case for selections, not even in the presence of expensive predicates [HS93]. Assuming a selection σ_i has a selectivity s_i and causes c_i costs per input tuple, we observe that

$$\begin{array}{rrcl} & C(\sigma_2(\sigma_1(R))) & \leq & C(\sigma_1(\sigma_2(R)) \\ \Leftrightarrow & c_1|R| + c_2 s_1|R| & \leq & c_2|R| + c_1 s_2|R| \\ \Leftrightarrow & c_2 s_1 - c_2 & \leq & c_1 s_2 - c_1 \\ \Leftrightarrow & \frac{s_1-1}{c_1} & \leq & \frac{s_2-1}{c_2} \end{array}$$

Or, to phrase it differently, the selections have to be sorted by $\frac{s_i-1}{c_i}$, independent of the input cardinality of the selection operator. Accordingly, incremental execution is not necessary for placing selections. The same is probably true for most other pipelining operators, as these tend to be decreasing unary operators. One notable exception is the nested loop join, we will discuss this below.

Pipeline breakers, in particular joins, are more complex and are usually candidates for triggering incremental execution on their input. Now the key observation here is that by their very nature, pipeline breakers are particularly well suited for incremental execution! As these operators materialize data anyway, we can use this to get incremental execution more or less for free. This is illustrated in Figure 7. When the query optimizer needs to know the input cardinality of a hash join for its plan decision, it conceptually breaks the execution plan into two parts (marked with *incr. break*). The first part executes the input plan and materializes the result (and potentially triggers a re-optimization step). The second part takes this materialized result, builds a hash table from it, and then probes the hash table with the data from the second input expression (shown on the left hand side

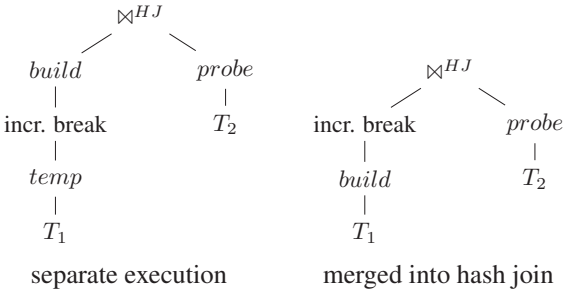


Figure 7: Merging Incremental Execution into Hash Joins

of Figure 7). Note that if the original cardinality estimate turns out to be accurate, the runtime system executes two materializing operators next to each other, namely *temp* and *build*. Therefore, one can improve the runtime performance by using only one of them, and directly materialize into the hash table (shown on the right hand side of Figure 7). If the optimizer sticks to the original plan after incremental execution this technique reduces the overhead of incremental execution to nearly zero, as again we only execute steps we would have executed anyway. If the optimizer does change the plan we have used a slightly more complex materialization operator than necessary, but in this case the benefits of the better plan far outweighs the costs of the more complex materialization.

Admittedly merging incremental execution into regular operator processing is not always as simple as the the example from Figure 7. For example materializing the *probe* side is more problematic, as the operator does not plan to materialize it. For symmetric join operators like the Grace Hash Join this is not an issue; in general an easy option would be to swap the roles of probe and build if both are reasonably small. But even if one indeed adds a materialization phase, it should prepare (e.g., partition) the data to help the subsequent operator. This strategy depends a lot on implementation details of the operators involved, but we found that most pipeline breakers can materialize any of their inputs in a way that helps subsequent processing. The same is true for nested loop joins, even though they are not pipeline breakers. Materializing the left side (i.e., the outer side) leads to block nested loop joins, which greatly reduce the number of passes over the inner side. Materializing the inner side can greatly reduce the costs if the inner side contains a complex execution plan.

In general, if the optimizer decides to trigger incremental execution, the runtime system should try to make use of this execution step. Note that once we materialize an intermediate result, we not only know its cardinality, but we have seen all the data. This allows for passing domain information throughout the execution plan, which can greatly reduce execution times. Therefore, if we decide to materialize an intermediate result, we should always build bitmap filters over join attributes that are used later on, as we get these bitmap filters nearly for free.

Overall incremental execution can be made quite cheap by exploiting the natural characteristics of the operators involved. The only disadvantage is that this requires some (minor) changes to the underlying runtime system, which is not always easy in commercial database systems. In the experiments in Section 6 we therefore measured both the behavior without

	comp. [s]	exec. [s]	total [s]
out-of-the-box optimizer	151.4	2437.5	2588.9
bottom-up join ordering	19.7	3508.0	3527.7
with incremental execution	1255.8*	515.5*	1771.3

* no clear distinction between compilation and execution

Figure 8: Query Processing for 40 Real-World Queries

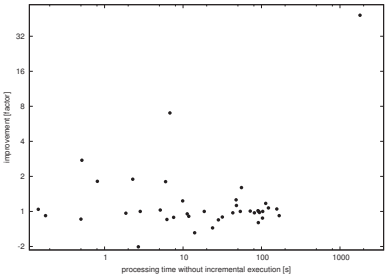


Figure 9: Effect of Incremental Execution on Individual Queries from Figure 8

explicit runtime support and the benefit of these runtime techniques.

6 Evaluation

We studied the effects of incremental execution on several large data sets. For the database system we used a development version of SQL Server, in which we integrated our techniques. To avoid a potential bias by the query optimizer, which out-of-the-box does not always use exhaustive search and therefore might produce plan differences due to timeouts, we implemented an exhaustive bottom-up join-ordering strategy [MN08] as basis for incremental execution. Note that the two optimizer approaches are not directly comparable. The bottom-up search guarantees that no join order is ignored just because of heuristical time reductions (which is important for the incremental execution experiments), but it misses some optimizations from the standard optimizer, in particular group-by optimizations. For completeness we always give results for both optimizers, but use the bottom-up construction to study the effects of incremental execution. Note that unless otherwise noted the results are without the runtime techniques from Section 5, i.e., they are purely query optimization effects.

For each data set and each of the approaches we ran a workload of queries and measured compilation time and execution time. If a query took more than half an hour (1800s) to execute we aborted it and counted the execution time as 1800s (this happened for the non-incremental algorithms). All experiments were conducted on a HP Compaq dc7900 with 8GB main memory and an Intel Core 2 Quad Q9400 CPU running Windows Server Enterprise.

6.1 Real-World Data

As a first data set we used data and queries provided by a customer where we knew that cardinality estimation had some difficulties. The data set is about 100GB in size (including indexes) and the 40 queries perform data-warehouse queries with reasonably complex predicates.

	comp. [s]	exec. [s]	total [s]
out-of-the-box optimizer	98.3	3418.9	3517.3
bottom-up join ordering	19.2	2140.1	2159.4
with incremental execution	386.6*	1593.8*	1980.5

* no clear distinction between compilation and execution

Figure 10: Query Processing for 99 TPC-DS Queries (scale factor 10GB)

We studied the query processing for all three approaches, the out-of-the-box optimizer, the bottom-up join ordering approach, and bottom-up join ordering including incremental execution. The results are shown in Figure 8. We notice two things: First, the bottom-up join ordering performs significantly worse than the out-of-the-box optimizer for this data set. This is caused by a combination of missing group-by movement and cardinality misestimation, which causes the bottom-up approach to choose a very poor execution plan. The second observation is that bottom-up join ordering with incremental execution performs much better than both, reducing query processing significantly, as it uses incremental execution to correct the problems of the non-incremental bottom-up optimizer. Note that there is no good notion of compilation time vs. execution time for incremental execution, we counted everything up to the final plan generation as compilation time and the final execution as execution time.

When looking at individual queries it is interesting to see that for most queries incremental execution has no effect all, but for some queries where the cardinality estimation has made a serious mistake it drastically improves query processing. This is shown in Figure 9: The x-axis is the query processing time of the bottom-up join ordering without incremental execution, and the y-axis shows the change by incremental execution. A value of ± 1 means that both approaches need the same query processing time, $+2$ means incremental execution improves query processing by a factor of 2, -2 means a degradation by a factor of 2. Note that the y-axis is on a logarithmic scale, which means that most queries are quite close to 1, i.e., incremental execution has no effect. However for some queries it improves query processing, in particular for a query that would have been very expensive otherwise. This matches the expectation we have of incremental execution: It is a tool to prevent gross mistakes caused by misestimations, which is exactly what happens in this data set.

Note that we do not just improve some outliers. Preventing outliers is the whole point of incremental execution! Customers do not really notice when most of their queries speed up by 10%. But when a single query slows down by a factor of 10 they notice immediately. Incremental execution allows us to mitigate these outliers caused by estimation errors.

Another interesting question is how often incremental execution is triggered. After all the query optimizer will only use incremental execution if some cardinalities were critical for plan generation choices. For this data set, the optimizer triggered 43 incremental executions in 32 out of the 40 queries. Which means that while many of these queries needed some incremental execution to clarify uncertainties, in most queries it was sufficient to execute one small part of the query to be certain about the correct join order.

6.2 TPC-DS

The customer data set was one of our original motivations for looking at incremental execution, but it has the disadvantage of not being publicly available. We therefore also studied some benchmarks to get results that are more easily reproducible. Unfortunately

build side		
	no plan change	plan change
regular spool	1.2%	1.2%
hash table spool	0%	<0.1%
probe side		
	no plan change	plan change
regular spool	7.9%	7.9%
hash table spool	<0.1%	0.5%

Figure 11: Overhead of forced Incremental Execution for a single Join

most synthetic benchmark data sets tend to be overly simplistic, with uniform value distributions, independence, etc., which does not exhibit some of the cardinality estimation problems visible for real-world data. One notable exception is TPC-DS [NP06], which contains data skew (but still no correlations between attributes), and more complex queries. This causes some challenges for cardinality estimation, although it is still much easier than for the real-world data set.

The results for the 10GB scale factor are shown in Figure 10. (Note that this is not an official benchmark result and the number are from a development version of SQL Server). Again, incremental execution improves query processing, but the gains are more minor, as the original cardinality estimates were already quite good. Furthermore we noticed that we use incremental execution much less than for the real-world data set, simply because it is not necessary in most cases. Incremental execution was triggered in 37 out of the 99 queries, executing 91 query fragments in total. As incremental execution was used much less than for the other data set, we manually checked the 10 most expensive queries where incremental execution was not used to make sure that no opportunity for incremental execution was missed, and indeed that seems to be the case. This is encouraging, as it means that the incremental execution mechanism adapts to the necessity of incremental execution.

For the TPC-DS data set we also tried using a ROX-style optimizer [KBMvK09] that executes one operator at a time and then greedily re-optimizes after each execution. This gave very disappointing results, with an total execution time of over 5,840s and over 250GB of intermediate results materialized on disk due to poor plan choices caused by the greedy algorithm. We did not pursue this any further for the other data sets, as it seems to be clear that the greedy ROX optimization strategy is too simple for many of the more complex queries.

6.3 Runtime Improvements

So far we intentionally showed results without using any of the runtime techniques from Section 5, as commercial database systems currently do not offer specific support for incremental execution. And indeed incremental execution performs very well, even without runtime support. However, we implemented a prototype of our techniques and used it to study the improvements for TPC-DS queries.

As an initial micro-benchmark, consider the join from TPC-DS Query 1 between *time_dim* and *store_sales*. One input of the join is relatively small, and will be used as *build* side, and

the other is reasonably large. In the context of a larger query, both might be candidates for incremental execution. We therefore study the overhead of incremental execution relative to regular execution along multiple dimensions: First, we measure the overhead of incremental execution when using regular *spool* operators versus the hash table spooling from Section 5. Second, we differentiate between the build or the probe side. And finally, we study the costs when we change the plan (i.e., cannot reuse the hash table and just use it as spool). The results are shown in Figure 11. Two things are noticeable. First, incremental execution is not that expensive here, even with regular spool operators, as the intermediate result can be materialized into main memory. Second, materializing directly into the hash table greatly improves performance, and nearly completely removes the overhead of incremental execution, even disregarding the potential benefits of better estimates.

For the complete TPC-DS query suite the effect is a bit more difficult to characterize fairly. The runtime techniques help a lot, but some of these gains stem from implementation changes, and even more from additional benefits like the piggy-back bitmap filter construction. By comparing query execution times we estimated that ignoring the gains (better plans, more filters, etc.), the overall overhead of incremental execution is reduced to about 1% for the 99 TPC-DS queries when using the techniques from Section 5. This is a very modest price for robustness regarding estimation errors, as this 1% overhead can frequently prevent outliers of a factor of 10 or more, which means that overall incremental execution speeds up query processing quite a lot.

7 Conclusion

For complex queries cardinality estimation errors are nearly unavoidable, and can lead to very poor executions plans. We propose using incremental execution to address the cases where cardinality estimation cannot reach the accuracy required for optimization purposes. By identifying plan alternatives and their sensitivity to estimation errors we can limit incremental execution to the cases where it is really necessary. Our experiments on large real-world and benchmark workloads showed that our incremental execution techniques can greatly reduce the effects of cardinality estimation errors, and thus lead to more robust query processing.

Future work should include integrating a feedback loop into our incremental execution framework similar to [SLMK01, LLZZ07], as the materialized intermediate results could provide a lot of information about cardinalities, error rates, value distributions etc., which would be useful for the query optimization but are currently discarded.

References

[AH00] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously Adaptive Query Processing. In *SIGMOD Conference*, pages 261–272, 2000.

[CCMN00] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. Towards Estimation Error Guarantees for Distinct Values. In *PODS*, pages 268–279, 2000.

[CG94] Richard L. Cole and Goetz Graefe. Optimization of Dynamic Query Evaluation Plans. In *SIGMOD Conference*, pages 150–160, 1994.

[Cha09] Surajit Chaudhuri. Query optimizers: time to rethink the contract? In *SIGMOD Conference*, pages 961–968, 2009.

[DIR07] Amol Deshpande, Zachary G. Ives, and Vijayshankar Raman. Adaptive Query Processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.

[Feg98] Leonidas Fegaras. A New Heuristic for Optimizing Large Queries. In *DEXA*, pages 726–735, 1998.

[GK05] Minos N. Garofalakis and Amit Kumar. Wavelet synopses for general error metrics. *ACM Trans. Database Syst.*, 30(4):888–928, 2005.

[HS93] Joseph M. Hellerstein and Michael Stonebraker. Predicate Migration: Optimizing Queries with Expensive Predicates. In *SIGMOD Conference*, pages 267–276, 1993.

[IK84] Toshihide Ibaraki and Tiko Kameda. On the Optimal Nesting Order for Computing N-Relational Joins. *ACM Trans. Database Syst.*, 9(3):482–502, 1984.

[Ioa03] Yannis E. Ioannidis. The History of Histograms (abridged). In *VLDB*, pages 19–30, 2003.

[JKM+98] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal Histograms with Quality Guarantees. In *VLDB*, pages 275–286, 1998.

[KBMvK09] Riham Abdel Kader, Peter A. Boncz, Stefan Manegold, and Maurice van Keulen. ROX: run-time optimization of XQueries. In *SIGMOD Conference*, pages 615–626, 2009.

[KBZ86] Ravi Krishnamurthy, Haran Boral, and Carlo Zaniolo. Optimization of Nonrecursive Queries. In *VLDB*, pages 128–137, 1986.

[KD98] Navin Kabra and David J. DeWitt. Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans. In *SIGMOD Conference*, pages 106–117, 1998.

[LLZZ07] Per-Åke Larson, Wolfgang Lehner, Jingren Zhou, and Peter Zabback. Cardinality estimation using sample views with quality assurance. In *SIGMOD Conference*, pages 175–186, 2007.

[Lor74] Raymond A. Lorie. XRM - An Extended (N-ary) Relational Memory. *IBM Research Report*, G320-2096, 1974.

[MMK+05] Volker Markl, Nimrod Megiddo, Marcel Kutsch, Tam Minh Tran, Peter J. Haas, and Utkarsh Srivastava. Consistently Estimating the Selectivity of Conjuncts of Predicates. In *VLDB*, pages 373–384, 2005.

[MN08] Guido Moerkotte and Thomas Neumann. Dynamic programming strikes back. In *SIGMOD Conference*, pages 539–552, 2008.

[MNS09] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. *PVLDB*, 2(1):982–993, 2009.

[MRS+04] Volker Markl, Vijayshankar Raman, David E. Simmen, Guy M. Lohman, and Hamid Pirahesh. Robust Query Processing through Progressive Optimization. In *SIGMOD Conference*, pages 659–670, 2004.

[NP06] Raghunath Othayoth Nambiar and Meikel Poess. The Making of TPC-DS. In *VLDB*, pages 1049–1058, 2006.

[RDH03] Vijayshankar Raman, Amol Deshpande, and Joseph M. Hellerstein. Using State Modules for Adaptive Query Processing. In *ICDE*, page 353, 2003.

[SAC+79] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access Path Selection in a Relational Database Management System. In *SIGMOD Conference*, pages 23–34, 1979.

[SLMK01] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. LEO - DB2's LEarning Optimizer. In *VLDB*, pages 19–28, 2001.

[TD03] Feng Tian and David J. DeWitt. Tuple Routing Strategies for Distributed Eddies. In *VLDB*, pages 333–344, 2003.

Resource Description and Selection for Range Query Processing in General Metric Spaces

Daniel Blank, Andreas Henrich

Media Informatics Group
University of Bamberg
D-96045 Bamberg
{daniel.blank|andreas.henrich}@uni-bamberg.de

Abstract: Similarity search in general metric spaces is a key aspect in many application fields. Metric space indexing provides a flexible indexing paradigm and is solely based on the use of a distance metric. No assumption is made about the representation of the database objects.

Nowadays, ever-increasing data volumes require large-scale distributed retrieval architectures. Here, local and global indexing schemes are distinguished. In the local indexing approach, every resource administers a set of documents and indexes them locally. Resource descriptions providing the basis for resource selection can be disseminated to avoid all resources being contacted when answering a query. On the other hand, global indexing schemes are based on a single index which is distributed so that every resource is responsible for a certain part of the index.

For local indexing, only few exact approaches have been proposed which support general metric space indexing. In this paper, we introduce RS4MI—an exact resource selection approach for general metric space indexing. We compare RS4MI with approaches presented in literature based on a peer-to-peer scenario when searching for similar images by image content. RS4MI can outperform two exact general metric space resource selection schemes in case of range queries. Fewer resources are contacted by RS4MI with—at the same time—more space efficient resource descriptions.

1 Introduction

The efficient processing of similarity queries (e.g. range queries searching for all database objects within a given search radius from the query object) is a key aspect in many domains and application fields such as multimedia and 3D object retrieval, similarity search on business process models, data compression, pattern recognition, machine learning, bioinformatics, statistical data analysis, malware detection, and data mining [ZADB05, HCS09, KW11, BKSS07]. Hereby, many similarity search problems are modeled in general metric spaces where no assumption is made about the representation of the database/feature objects. The only assumption is that distances between feature objects can be measured by a distance metric.

Furthermore, in many search scenarios, centralized architectures are no longer sufficient and large-scale solutions are necessary. Here, as a particular technique for distributed

query processing, resource selection techniques provide a valuable solution. They are for example applicable in dynamic environments such as peer-to-peer (P2P) information retrieval (IR) systems with data sources joining and leaving frequently.

In the P2P IR domain, it can become infeasible to solely apply *global indexing schemes*, i.e. distributed indexing structures with every peer being responsible for a certain range of the feature space and peers transferring their indexing data to remote peers according to their "region(s) of interest". Peers entering the system and updating indexing data might induce a high network load the system can hardly cope with [LLOS07].

Summary-based resource selection approaches and thus *local indexing schemes* such as the ones discussed and evaluated in this work are one possibility to deal with this problem. Here, every peer indexes the data it administers and describes it in form of data summaries which are transfered to remote peers. During search, promising peers are selected based on the resource descriptions and the query is sent to them. In summary-based P2P IR systems, peers leaving the network ungracefully do not take indexing data of other peers' documents with them. Furthermore, leaving peers do not take documents with them for which indexing data is still present in the network and the documents thus still can be found. Peer autonomy is better respected compared to distributed index structures. On the other hand, many distributed index structures offer query processing with logarithmic cost [DVNV10] which is hard to guarantee for local indexing schemes.

The work in this paper focuses on space efficient resource description and corresponding selection techniques which allow for efficient distributed query processing in general metric spaces. As a proof-of-concept and application scenario being assumed, the resource description and selection techniques are designed for the use within a particular P2P IR scenario. However, they can also be applied for traditional resource selection in distributed IR and within other variants of P2P IR systems. Furthermore, there is a range of possible application fields beyond P2P IR systems, such as (visual) sensor [ERO+09] and ad-hoc networks [LLOS07], to name only a few.

As the contribution of this paper, we present RS4MI (Resource Selection for Metric Indexing), a new exact resource description and selection technique applicable for similarity search in general metric spaces. Its design is motivated by application scenarios where space efficient resource descriptions are required. As a rule of thumb, the average size of a resource description in our scenario should be below 1 kB. However, the presented techniques are by no means limited to this scenario. We review related work in the field of resource selection and identify techniques applicable in general metric spaces. The only exact technique presented in literature so far is compared against RS4MI. In addition, another baseline technique relying on local k-medoid clustering is included in the analysis.

The remainder of this paper is organized as follows. In Sect. 2, we briefly recapitulate main concepts of similarity search in metric space. Sect. 3 discusses further related work by presenting existing solutions to the resource description and selection problem in general metric spaces. RS4MI and the two competing approaches are in detail outlined and evaluated in Sect. 4. The paper concludes with an outlook on future work in Sect. 5.

2 Metric Space Indexing

Multi-dimensional (spatial) access methods (SAMs; for an overview cf. [Sam06]) are designed for vector spaces whereas metric access methods (MAMs) can be applied in any metric space. An overview on MAMs is for example given in [CNBYM01, ZADB05]. A metric space $\mathcal{M}$ is defined as a pair $\mathcal{M} = (\mathbb{D}, d)$. $\mathbb{D}$ represents the domain of objects $o \in O$ with $O \subset \mathbb{D}$ and $d : \mathbb{D} \times \mathbb{D} \to \mathbb{R}$ corresponds to a metric distance function which satisfies the metric postulates $\forall x, y, z \in \mathbb{D}$ [ZADB05]:

$$
\begin{aligned}
&d(x,y) = 0 \iff x = y && \textit{identity}\\
&d(x,y) > 0 \iff x \neq y && \textit{non-negativity}\\
&d(x,y) = d(y,x) && \textit{symmetry}\\
&d(x,y) + d(y,z) \geq d(x,z) && \textit{triangle inequality}
\end{aligned}
$$

Many MAMs rely on a set $C = \{c_i | 1 \leq i \leq n\}$ of reference objects (also called pivots or centers) in order to structure the feature space. There are different ways of how to partition the feature space. Within *ball partitioning* methods [ZADB05], the feature space is partitioned by often multiple hyper-spheres. In contrast, many structures relying on *hyperplane partitioning* conceptually rely on a list L_o, ordering the pivot IDs i by increasing $d(c_i, o)$. In case of *generalized hyperplane partitioning* [ZADB05], o is assigned to the cluster (i.e. a region of the feature space induced by the space partitioning) with ID $L_o[1]$ of the closest reference object $c^* = \arg\min_{c_i \in C} d(c_i, o)$. In other cases, the list L_o truncated after position l with $l \in \{2, \ldots, n\}$ identifies the cluster where o lies in (cf. [NBZ11]).

The distance between feature objects is frequently used to model the similarity between them. Usually, it is assumed that the smaller the distance the higher the similarity. In this context, range queries are a popular type of similarity queries [Sko06, p. 4].

A *range query* $R(q, r)$ with query object $q \in \mathbb{D}$ and search radius $r \in \mathbb{R}^+$ retrieves all database objects from $O \subset \mathbb{D}$ which are within distance r from q, i.e. $\{o \in O \,|\, d(q, o) \leq r\}$. The subspace $\mathbb{V} \subset \mathbb{D}$ for which $\forall v \in \mathbb{V} : d(q, v) \leq r$ and $\forall v' \in \mathbb{D} \backslash \mathbb{V} : d(q, v') > r$ is called the *query ball* [SB11].

For the space partitioning methods outlined above as well as hybrid combinations, various pruning criteria can be applied. They are in the following described in the context of range queries following the notation of [ZADB05].

Pruning criteria in metric spaces

When only *per-cluster* information (in contrast to *per-object* information) is stored in the resource descriptions, range query processing can be summarized as follows. The data descriptions of the resources are iteratively analyzed. If all populated database clusters of a resource can be pruned, i.e. no populated cluster intersects the query ball, the very resource can be discarded from search. Remaining resources have to be contacted. Criteria capable of cluster and hence resource pruning are outlined in the following, similarly to [NBZ11].

If a query lies in the cell of center c^* (i.e. reference object c^* is the closest center out of the set C of all available reference objects according to a given query object q), by exploiting the triangle inequality, any cluster $[c_i]$ can be pruned if $d(c_i, q) - d(c^*, q) > 2r$, where r corresponds to the search radius (*double-pivot distance constraint*).

If a maximum cluster radius r_i^{max} for a cluster $[c_i]$ is given, i.e. the maximum distance of any object o in the cluster from its center c_i, the very cluster can be pruned if $d(c_i, q) - r > r_i^{max}$ (*range-pivot distance constraint*). A similar condition can be applied according to the minimum cluster radius r_i^{min}, i.e. the minimum distance of any object o within the cluster from its center c_i. Cluster $[c_i]$ can be pruned if $d(c_i, q) + r < r_i^{min}$.

The range-pivot distance constraint can also be used in an inter-cluster way. To this end, two matrices *MAX* and *MIN* are applied to store maximum and minimum cluster radii $r_{i,j}^{max}$ and $r_{i,j}^{min}$ respectively for $i, j \in \{1, \ldots, n\}$, where $r_{i,j}^{max}$ represents the maximum distance of any object from cluster $[c_i]$ to cluster center c_j, and $r_{i,j}^{min}$ represents the minimum distance of any object from cluster $[c_i]$ to cluster center c_j. Elements $r_{i,i}^{max}$ and $r_{i,i}^{min}$ on the diagonal of the matrices *MAX* and *MIN* thus capture the maximum cluster radius r_i^{max} and the minimum cluster radius r_i^{min} of cluster $[c_i]$, respectively, as described above. Cluster $[c_i]$ can be pruned if there exists a cluster $[c_j]$ for which $d(c_j, q) + r < r_{i,j}^{min}$ or $d(c_j, q) - r > r_{i,j}^{max}$ [Woj02].

Fig. 1 visualizes a search situation in case of a range query with search radius r where cluster $[c_1]$ can be pruned successfully. By solely using the double-pivot distance constraint, cluster $[c_1]$ cannot be pruned, since the query ball $\mathbb{V}$ intersects cluster $[c_1]$. If, for every cluster, we administer only the minimum and the maximum cluster radius of objects in the cluster (shown by the hyper-ring $\mathbb{H}_{1,1}$ around cluster center c_1 in Fig. 1), cluster $[c_1]$ can still not be pruned. The matrices *MIN* and *MAX* are thus necessary to successfully prune cluster $[c_1]$. If we also apply the radii $r_{1,2}^{min}$ and $r_{1,2}^{max}$, i.e. the minimum and the maximum distance of feature objects in cluster $[c_1]$ from c_2, it can be determined that there are no relevant feature objects in the intersection area of the query ball $\mathbb{V}$ and the hyper-ring $\mathbb{H}_{1,1}$. The region of possible feature objects is limited to the two dark gray shaded intersection areas of $\mathbb{H}_{1,1}$ and $\mathbb{H}_{1,2}$, and since the query ball $\mathbb{V}$ does not intersect any of these regions, cluster $[c_1]$ does not contain any database objects relevant to the query.

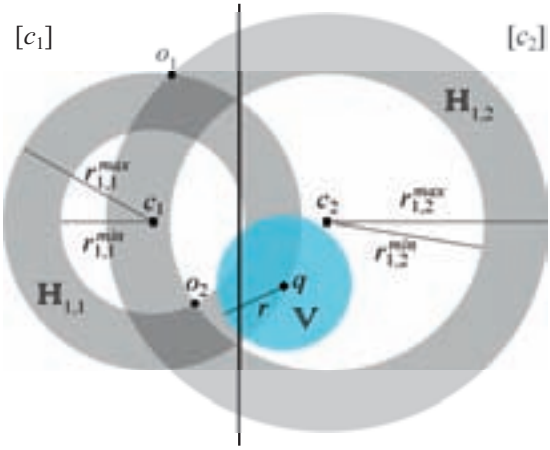


Figure 1: Cluster pruning example.

A further pruning constraint can be applied on an object level rather than a cluster level. The application of this constraint in a resource selection scenario requires per-object in-

formation to be stored in the resource descriptions—either solely or in addition to storing per-cluster information. Contacting a resource can be avoided by storing distance values $d(c_i, o)$ in the summaries. If $|d(c_i, q) - d(c_i, o)| > r$, object o can be pruned without computing $d(q, o)$. This is called the *object-pivot distance constraint*.

In order to enhance the pruning power for complete resources, $d(c_i, o)$ values can be stored for multiple cluster centers c_i. Hence, contacting a resource is not necessary if $\max_{c_i} |d(c_i, q) - d(c_i, o)| > r$ is fulfilled for all database objects of a resource. This so called *pivot filtering* is a direct application of the object-pivot distance constraint. Although appearing impracticable at first glance due to the space requirements, per-object information might be useful in hybrid approaches for peers with few objects. This will be considered in Sect. 4.4.

3 Related Work on Resource Selection in General Metric Spaces

There is plenty of work on the description and selection of text databases in distributed IR (cf. [SS11]). In addition, some resource description and selection schemes have emerged in the context of content-based multimedia IR such as in content-based image retrieval (CBIR). Our work addresses *local indexing approaches* and in particular the ones which consider the resource selection task as a *geometric problem*. Here, certain properties of the feature space or distance information are used in order to prune resources which cannot contribute database objects to the search result. The remaining resources can be ranked by the "proximity" of their feature objects and the query object (which can be beneficial when e.g. performing k-nearest neighbor (k-NN) queries). We will discuss these approaches in the following. Probabilistic (cf. e.g. [NF03, EBMH08]) as well as geometric resource selection techniques only applicable in vector spaces (cf. e.g. [KLC02]) are out of the scope of our present work. We also do not consider database selection approaches based on one-dimensional numeric values (e.g. [YSMQ01]).

In the following, two approaches applicable in general metric spaces are described. The approach by Berretti et al. [BDP04] is the only approach which represents an exact resource selection scheme, the latter approach is an approximate technique. However, it is presented here, because RS4MI can be considered as an extension of this approach w.r.t. exact query processing.

Berretti et al. [BDP04] applies a special form of hierarchical clustering based on the M-tree [CPZ97] to a resource's set of feature objects to generate a resource description. A cluster radius threshold θ is used for determining the cluster centers which are included in the resource description. Every path in the clustering tree built for the local collection is descended as long as the cluster radius of a node is bigger than the predefined threshold θ. The centers of the nodes where the search stops are included in the resource description. In addition, per cluster, the maximum cluster radius, i.e. the maximum distance of a database object from its cluster center as well as the number of objects within the cluster are stored in the resource description (the latter may be beneficial for ranking peers when performing k-NN queries). By varying θ, the granularity and size of the resource descriptions can be

adjusted. It is suggested in [BDP04] to set θ to the maximum possible distance value if the distance metric has an upper bound. The block size of the M-tree nodes is the second tuning parameter of this approach. When it comes to resource selection, a resource cannot be pruned from search if the query ball intersects any cluster ball of the resource.

Eisenhardt et al. [EMH$^+$06] extends the cluster histogram technique initially proposed in Müller et al. [MEH05a]. To compute a cluster histogram as resource description, a set with a moderate number of reference objects c_i is applied: $C = \{c_i | 1 \leq i \leq n\}$ with e.g. $n = 256$. Every feature object of a resource's collection is assigned to the closest reference object and a histogram captures how many objects have been assigned to a certain reference object. Eisenhardt et al. [EMH$^+$06] shows that a random selection of reference objects might replace distributed clustering. Resource selection performance slightly decreases, but network load can be reduced because distributed clustering becomes obsolete. For performing k-NN queries, during peer ranking a list L_q of reference object IDs i is sorted in ascending order according to $d(q, c_i)$, i.e. the distance from the query object q to a cluster center c_i. The first element of L_q corresponds to the ID of the cluster center being closest to q. A peer with more documents in the corresponding cluster—indicated by the summary—is ranked higher than a peer with fewer documents in the very cluster. If two peers p_a and p_b administer the same amount of documents in the analyzed cluster, the next element out of L_q is chosen and—based on the indicated number of documents within the very cluster—it is tried to rank peer p_a before peer p_b or vice versa. When the end of the list L_q is reached, a random decision is made. The resource descriptions of this approach are further improved in [BEMH07, BH10]. They are binarized and the number of used reference objects is increased to e.g. $n = 8192$ or even more. Compression techniques are applied to prevent a huge increase in average summary sizes.

4 Exact Resource Selection Approaches for Metric Space Indexing

In the following, we describe and compare three different resource description and selection schemes for metric space indexing. The experimental setup is outlined in Sect. 4.1. In Sect. 4.2, the technique introduced in Berretti et al. [BDP04] (cf. Sect. 3) is analyzed. Another approach—based on k-medoid clustering and used as second comparison baseline for RS4MI—is presented in Sect. 4.3. RS4MI is explained and analyzed in Sect. 4.4. Finally, Sect. 4.5 subsumes the main results of the experimental comparison.

4.1 Experimental setup

We analyze a scenario where every peer knows the resource description of every other peer. Of course, such an approach would not scale. However, this scenario is for example typical in a subnet of a scalable Rumorama-based P2P IR network. Rumorama [MEH05b] can cope with multiple subnets and thus scale to much higher workloads than the ones analyzed in this work.

As underlying data collection, 233827 images crawled from Flickr are used (cf. [BH10]). They are assigned to peers based on the Flickr user ID in order to reflect a realistic scenario, i.e. distribution to resources. Hence, we assume that every Flickr user operates a peer of its own. In this way, the images are mapped to 10601 peers/users. Fig. 2 shows the distribution of peer sizes, i.e. the number of images which are maintained per peer. The general characteristic is typical for P2P file sharing applications, with few peers managing large amounts of the images and many peers administering only few images [SKG02].

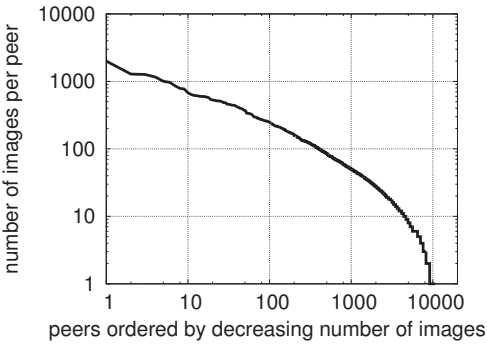


Figure 2: Distribution of peer sizes, i.e. the number of images per peer.

Pivots for summary creation and peer ranking in case of RS4MI are randomly chosen from a secondary data collection consisting of 45931 Flickr images. This reflects a scenario where the reference objects are transfered to the resources with updates of the P2P software in order to reduce network load. All resources administer the same set of pivots. The external (secondary) collection is disjoint from the underlying collection according to the unique Flickr image and user IDs. However, there is some minor natural overlap amongst collections according to image content; 24 of the 233827 images also appear in the external collection because some images are uploaded independently by multiple users on Flickr.

In the experiments, query objects are randomly chosen from the underlying data collection. This seems reasonable in case of range queries relying on the query-by-example paradigm. Retrieval performance is measured by analyzing peer selectivity, i.e. the fraction of peers which must be contacted to retrieve all images with feature objects lying within distance r from q. In addition to search efficiency, the size of the resource descriptions is analyzed. If not mentioned otherwise, summaries are compressed with gzip[1].

As feature descriptor, we use the unquantized version of the CEDD descriptor[2] (144-dimensional vector of 4 byte floats and thus in total 576 byte per descriptor). CEDD has the potential to outperform the MPEG-7 features for CBIR [CZBP10]. The Hellinger metric $d(q,o) = d_H(q,o) = (2 \cdot d_{SC}(q,o))^{\frac{1}{2}} = (2 \cdot \sum_i (\sqrt{q[i]} - \sqrt{o[i]})^2)^{\frac{1}{2}}$ (cf. [DD09]) is applied converting the non-metric squared chord distance d_{SC} into a metric. It is shown in [LSR$^+$08] that d_{SC} provides good retrieval results in case of CBIR. Internal studies with two collections of groundtruth images reveal that the Hellinger metric in combination with CEDD features offers promising retrieval results, outperforming many other distance measures. However, our analysis does not focus on search effectiveness in CBIR and thus the choice of an effective feature descriptor in combination with a distance metric is not the

[1]The time requirements for building the resource descriptions are not analyzed in this work. This task is parallelized in a real-world scenario with every peer computing its resource description and hereby all promising approaches subsumed in Sect. 4.5 are suitably fast.

[2]Features were extracted using the Lire library obtained from *http://www.semanticmetadata.net/lire/*.

	min	q25	median	mean	q75	max
database objects	1	4	21.5	126.7	94.3	2028
peers	1	3	18	72.0	76.3	654

Table 1: Statistics of the number of relevant *database objects* and the number of *peers* administering relevant documents for the 200 range queries with search radius $r = 0.5$.

main focus of our work. Our general setting offers an intrinsic dimensionality (as defined in [CNBYM01, p. 303]) of almost 10 and thus represents a rather hard indexing task.

We evaluate 200 range queries with search radius $r = 0.5$ for every parameter setting. Tab. 1 shows statistics of the number of database objects lying within the search radius. Relevant documents are on average found at 72 peers. An optimal resource selection would thus on average only contact $\frac{72}{10601} \approx 0.7\%$ of the peers to retrieve the relevant documents.

4.2 M-tree based local clustering

To our knowledge, the approach by Berretti et al. [BDP04] is the only exact approach which has so far been proposed for general metric space indexing. Thus, we apply this technique as a comparison baseline for RS4MI. In order to do so, we use revision 27 of the M-tree library from *http://mufin.fi.muni.cz/trac/mtree/* (last visit of all URLs in this paper on 27.09.12) and acknowledge its contributors. The approach mainly depends on two parameters. A cluster radius threshold θ and the block/node size of the M-tree are the keys for trading-off the granularity of the resource descriptions (cf. Sect. 3) versus their selectivity. The influences and interactions when varying these two parameters are evaluated in the following.

The insertion of all database objects of a resource into an M-tree and the threshold-based search algorithm for generating the resource description leads to a partitioning of the feature space based on multiple hyper-spheres. A reference object m_i together with the cluster radius r_i^{max}, both maintained in a node entry of the M-tree, has to be stored in the resource description for every cluster (i.e. hyper-sphere) to be able to perform exact range queries. With this information, the range-pivot distance constraint (cf. Sect. 2) testing the overlap of the query ball with any cluster ball can then be applied during search in order to prune irrelevant peers, i.e. peers with no relevant documents.

Analysis of M-tree based local clustering

In the upper left quadrant of Fig. 3, the selectivity of the summaries of the M-tree based local clustering approach is shown. The lower left quadrant depicts the corresponding summary sizes. To understand this figure, the following aspects have to be considered: (1) A block size of 576 byte corresponds to leaf nodes containing one object each. In this case, the M-tree implementation assures that inner nodes (including the root node) are bigger and the degree of each inner node is two. In general, a block size of s_b means that

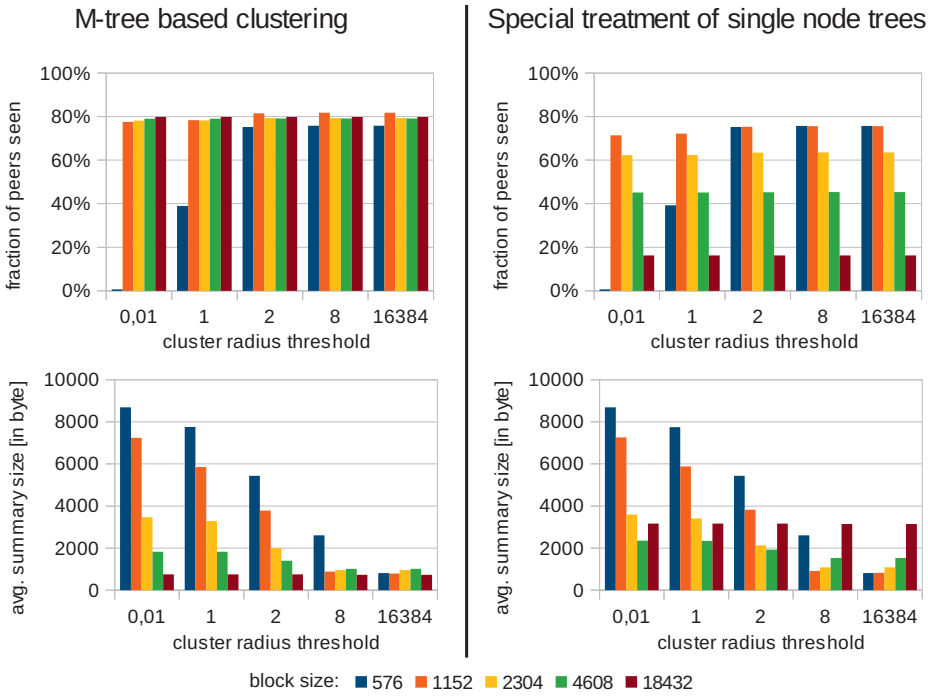


Figure 3: M-tree based local clustering (left) with special treatment of single node trees (right).

a leaf node contains at most $s_b/576$ objects. Hence, e.g. a node size of 18432 corresponds to leaf nodes containing at most 32 objects. (2) A cluster radius threshold of 0.01 has the consequence that the summary roughly contains clusters describing exactly the leaf nodes. At the other extreme, a cluster radius threshold of 16384 would yield a summary containing only one cluster representing the root node of the M-tree and in consequence the complete set of objects on the peer[3].

With the above information in mind, we can interpret the left side of Fig. 3. If we consider the average summary size in dependence of the cluster radius threshold (lower left quadrant) it becomes obvious that the summary sizes decrease for higher threshold values. The reason is that for higher threshold values the clusters for the summaries are taken from higher levels of the M-tree. Obviously, this effect is only given for small block sizes (blue, orange and yellow bars), because for higher block sizes (e.g. dark red bars) the height of the M-trees is extremely low anyway.

The upper left quadrant of Fig. 3 shows the selectivity of the summaries measured by the fraction of peers seen. Let us first consider the fraction of peers seen in dependence of the cluster radius threshold. As a special case, the block size of 576 together with a cluster radius threshold of 0.01 has to be considered. In this situation, each leaf node contains

[3]Please note that all object distances are at most 2. However, due to heuristic upper bound approximation of the cluster radii in the inner nodes of the M-tree, values bigger than 2 exist in the tree.

only a single item and because of the low threshold value, the clusters describing the leaf nodes are included in the summaries. Consequently, the summaries exactly represent the objects on each peer. Based on this information, a querying peer can exactly determine the peers containing objects in the query ball and therefore, the fraction of peers seen corresponds to the theoretical optimum of 0.7%. However, this result is achieved by a complete replication of all objects within the network on all peers. Unfortunately, these parameter settings are not realistic for huge networks. Neither a threshold value yielding only leaf nodes nor a node size storing only one object per node are practical.

Despite from these special cases, the fraction of peers seen is roughly between 70% and 80%. It is also interesting to consider the effect of the block sizes e.g. for a cluster radius threshold of 8. With this threshold, only in very rare cases the clusters used in the summary are taken from lower levels of the tree. With the block size of 576 byte, peers with only one image are represented by one cluster in the summary and peers with 2 or more images are (with some exceptions) represented by two clusters, since the fan-out of the root node is 2 in this case. With the block size of 1152 byte, peers with one or two images are represented by one cluster in the summary and peers with 3 or more images are (with few exceptions) represented by two clusters. The less precise representation of peers with 2 images results in an increase of the peers which have to be considered from 75.8% to 81.8% and at the same time reduces the average summary size drastically. With the block size of 2304 byte, peers with one to four images are represented by one cluster in the summary and peers with 5 or more images are (with few exceptions again) represented by two to four clusters. Hence, the summaries of small peers become less accurate but the summaries of bigger peers become more accurate, since the root node of the M-tree now has up to 4 successors. Obviously, these considerations can be continued for bigger block sizes.

The above results achieved for the originally proposed M-tree based local clustering approach inspired us to change the approach marginally in order to exploit the long-tail distribution of images on peers (cf. Fig. 2). Over 50% of the peers contain 7 or less images. As a consequence: If the summaries of these small peers would contain the exact objects, only the peers out of these 50% which really contribute to the result of the range query must be visited. With such a technique we can easily outperform the approaches presented above which have to address 70% to 80% of the peers.

To integrate this idea into the M-tree based local clustering approach we use a special treatment for situations where the M-tree consists of only one (leaf) node—which is typical for small peers. In this case the summary now contains one cluster with radius zero for each object in this leaf node instead of one single cluster with a huge radius describing the whole node. As a consequence e.g. at a block size of 18432 byte a peer maintaining 32 objects fitting into one single leaf node is now represented by a summary containing these 32 objects as single clusters with the objects as centers and radius zero.

The effect of this variation can be seen on the right hand side of Fig. 3. Let us—again at a cluster radius threshold of 8—consider the green bars representing a node size of 4608 byte, resp., at most 8 objects. In this case 5643 (= 53%) of all peers are represented exactly in the summaries. This allows to reduce the number of peers to be contacted during query processing to 45.3%. The avg. summary size is 1531 byte (compared to 1010 byte without the special treatment of small peers).

Although, the improvements achieved with this variation are impressive, it remains a bit problematic that we have such indirect and hard to handle parameters; the threshold value θ, the block size of the M-tree and the special treatment of trees comprising only one node. According to the threshold value θ, Fig. 3 shows that the heuristic of setting $\theta = 2$, i.e. the maximum possible distance value, might not be a suitable solution in all cases. In fact, it might be much easier to use an explicit clustering approach with more intuitive parameters. This directly leads us to the k-medoid clustering.

4.3 Local k-medoid clustering

Some approximate resource selection approaches for the use in vector spaces apply k-means clustering to cluster the database objects of a peer (cf. e.g. [EBMH08]). However, k-means, due to the mean calculation, is not applicable in general metric spaces. When using k-medoid clustering instead (or any other suitable algorithm applicable in general metric spaces), an additional baseline technique for the comparison with RS4MI can be designed. In this case, each peer clusters its local data collection and stores cluster centers m_i and maximum cluster radii r_i^{max} in its resource description. This results in a similar data space partitioning and similar resource descriptions as the approach proposed in Berretti et al. [BDP04] (cf. Sect. 4.2). The resource description of a peer in case of range queries thus consists of a list of cluster center and corresponding maximum cluster radius pairs.

There are two general options for determining k, i.e. the number of clusters of a peer needed as an input parameter to k-medoid clustering. As one alternative, the maximum number of allowed clusters per peer k can be set as a global threshold being identical for all peers. Of course, peers with less than k distinct database objects directly transfer these and do not apply clustering. On the other hand, algorithms which automatically detect an appropriate number of clusters can be used. Multiple of these algorithms are presented in literature (for references see e.g. [TWH01]). Our choice of algorithms in the following is by no means exhaustive. It is our intention to evaluate different techniques which return a range of average numbers of clusters per peer when applied to our scenario.

Rule of thumb (r.o.t.): A coarse rule of thumb is presented in [MKB79, p. 365]. It is suggested to calculate the number of clusters of a data set of size $|O|$ as $k \approx \sqrt{|O|/2}$. Thus, we use $k = \lceil \sqrt{|O|/2} \rceil$.

This rule of thumb directly calculates the number of desired clusters. In contrast, the techniques presented in the following are applied in an iterative process. A single key figure results for a specific value of k. Various values of k are thus to be tested to select the best k minimizing/maximizing the key figure. To reduce runtime performance, even when applying the rule of thumb, an adaptation of the original k-medoid clustering algorithm is used. The FAMES extension to k-medoid clustering uses pivots in order to speed-up k-medoid clustering [PNT11]. FAMES avoids the calculation of all pair-wise distances when computing the medoid of a certain cluster. In addition to improving efficiency, it is shown in [PNT11] that FAMES can also increase the effectiveness of the clustering since the efficiency gain is not due to the consideration of a random sample of database

objects as medoid candidates—which is the approach of some traditional algorithms. For determining the initial candidate set of medoids, we minimize in 10 runs the sum over all clusters of within-cluster object-to-medoid distances.

Besides the rule of thumb, we apply three variants of the well-known GAP statistic. The GAP statistic [TWH01] is frequently used and offers the property that—in contrast to many alternative approaches—it can also detect the presence of only a single cluster.

GAP: The GAP statistic as originally defined in [TWH01] is based on a sampling process which is not directly applicable in all metric spaces. However, as suggested in [TWH01], when only distance information is available, a specific mapping technique such as multidimensional scaling can be used to obtain feature vectors in a low dimensional space, which provide the basis for the sampling process. In our experiments, we directly apply ten sampling steps on the feature vectors without the use of an additional mapping technique in order to obtain a best case comparison baseline against which we can compare our approach RS4MI.

GAP$_w$, introduced in [YY07], modifies the weighting scheme of the GAP statistic.

GAP$_n$ represents another slight modification of the GAP statistic, where all logarithms used in the formulae of the GAP statistic are removed [MES10].

Sil$_1$ **and Sil**$_2$**:** The Silhouette technique [Rou87] is also adapted as a means for calculating the desired number of clusters of a peer. It is only applicable in case of $k > 1$. Thus, two alternatives are used in our experiments. If two is indicated as optimum cluster number, we set $k = 1$ in case of Sil$_1$; $k = 2$ is used in case of Sil$_2$. Peers with only a single database object—of course—only encode a single cluster in the resource description.

To determine an appropriate value for k, the above mentioned approaches based on the GAP statistic and the Silhouette technique are iteratively tested on every peer till $k = \lfloor \min(2\sqrt{nDocs}, nDocs) \rfloor$ with *nDocs* denoting the number of documents/images of a peer.

Analysis of local k-medoid clustering

Tab. 2 (top) shows the average fraction of visited peers, the average number of clusters per peer, as well as average summary sizes in case of local k-medoid clustering when automatically determining the number of clusters of a peer. The rule of thumb (r.o.t.) leads to decent retrieval performance at the cost of comparatively large summaries. Better peer selectivity is achievable by the SIL$_2$ approach with more space efficient resource descriptions.

Using the GAP statistic for determining the number of clusters of a peer results in average summary sizes of approximately 1 kB and 73.7% of peers being contacted for retrieving all relevant documents. GAP$_w$ and GAP$_n$ lead to fewer numbers of clusters per peer and thus more space efficient resource descriptions. However, both perform worse than GAP.

SIL$_1$ leads to similar average summary sizes as GAP. The average number of clusters per peer is in both cases approximately 2.3, but GAP offers better peer selectivity. SIL$_1$ always assumes one cluster when there might be two (which GAP might detect). SIL$_2$ shows better peer selectivity than the other competing approaches (even better than the

	r.o.t.	GAP	GAP_w	GAP_n	SIL_1	SIL_2
visited peers	67.4%	73.7%	75.3%	76.4%	76.4%	65.7%
clusters per peer	3.1	2.3	1.9	1.5	2.3	2.8
summary size	1350.3 B	1048.4 B	880.7 B	722.9 B	1029.5 B	1232.2 B

	$k=1$	$k=2$	$k=4$	$k=8$	$k=32$	$k=128$
peers seen	79.9%	68.7%	54.4%	37.7%	13.1%	2.7%
clusters per peer	1.0	1.9	3.4	5.6	12.2	18.2
summary size	525.0 B	867.2 B	1.4 kB	2.3 kB	4.9 kB	7.3 kB

Table 2: Results for local k-medoid clustering with automatic determination of the number of clusters k (top) and all peers using the same global k (bottom).

rule of thumb which identifies on average 3.1 clusters per peer) at the cost of storing on average 2.8 clusters per peer in the summaries. Overall, GAP_n and GAP_w seem promising approaches with average summary sizes clearly below 1 kB.

A main drawback of the k-medoid approaches analyzed in this section so far is that the summary sizes cannot be influenced by any kind of design parameter of the approach. An alternative in this respect is to globally specify k, the maximum allowed number of clusters per peer. In this case, peers with *nDocs* $\leq k$ store all feature objects in their summary. Since for some peers the number of feature objects is smaller than k, the average number of clusters per peer becomes smaller than k as well. This scenario which is thus similar to the special treatment of single node trees in Sect. 4.2 is evaluated in Tab. 2 (bottom).

The explicit definition of an upper bound for the number of clusters allows for a direct and accurate adjustment of summary sizes and selectivity. This gives a clear advantage over the M-tree based approach and also over the approaches which automatically determine a suitable number of clusters per peer. However, if very small summaries are necessary, the flexibility is restricted by the discrete values of k.

It can be observed from Tab. 2 (bottom) that only in cases where the maximum desired number of clusters per peer is set to $k = 1$ or $k = 2$, average summary sizes with less than 1 kB can be achieved. If a maximum of two clusters is allowed, 68.7% of the resources are visited with an average summary size of 867 byte. In order to further reduce this figure, only a single cluster per peer can be allowed. However, almost 80% of peers are contacted in this case with an average summary size of 525 byte.

4.4 RS4MI: Resource Selection for Metric Indexing

RS4MI can make use of all pruning criteria mentioned in Sect. 2. A set of n reference objects C—globally unique for all resources—is applied in order to assign a database object o of a resource to the closest cluster center $c^* = \arg\min_{c_i \in C} d(c_i, o)$. The set of reference objects is transfered to remote peers together with updates of the P2P software,

so that no additional network load is imposed during the operating phase of the P2P IR system. Such an approach is for example proposed in [BEMH07].

Different variants of RS4MI resource descriptions are evaluated in the following to find the best alternative. These variants can make use of different pruning criteria and thus result in different peer selectivity and average summary sizes.

RS4MI$_{\mathbf{1xxxx}}$: Here, only a single bit is stored per cluster in order to indicate if any database objects lie in the very cluster or not. This results in a bit vector of size n and thus resource descriptions with $\mathcal{O}(n)$ space complexity. The double-pivot distance constraint outlined in Sect. 2 is the only pruning constraint which can be used in this case to prune peers from search.

RS4MI$_{\mathbf{x??xx}}$: Resource descriptions offering $\mathcal{O}(n)$ space complexity can also be designed by storing the minimum and/or maximum cluster radii. By doing so, the range-pivot distance constraint can be applied on an intra-cluster level (cf. Sect. 2). In addition to storing both minimum and maximum cluster radii for the n clusters (i.e. RS4MI$_{\text{x11xx}}$), we test parameter settings of RS4MI$_{\text{x1xxx}}$ and RS4MI$_{\text{xx1xx}}$ where only minimum or maximum cluster radii are stored respectively. A single distance value is always represented as a four byte float.

Of course, the double-pivot distance constraint can also be applied in this case. If no minimum/maximum cluster radius is set for a particular cluster, it is indicated by the summary that the corresponding peer does not administer any database objects within the very cluster. So, the double-pivot distance constraint is used by all of the following resource selection schemes whenever applicable.

RS4MI$_{\mathbf{xxx??}}$: If all criteria for cluster pruning described in Sect. 2 should be applied, two matrices *MIN* and *MAX* have to be administered by every database as resource description (RS4MI$_{\text{xxx11}}$). This requires $\mathcal{O}(n^2)$ space per resource. As before, a single matrix cell requires four byte in order to store radius information. Both matrices are sent as resource description and used for the pruning of resources without querying them. We also test parameter settings where only a single matrix *MIN* (RS4MI$_{\text{xxx1x}}$) or *MAX* (RS4MI$_{\text{xxxx1}}$) is used.

Two further combinations are included in the analysis. RS4MI$_{\text{x1xx1}}$ stores minimum cluster radii and the matrix *MAX* as resource description. In opposition, RS4MI$_{\text{xx11x}}$ applies maximum cluster radii and the *MIN* matrix to prune peers during search.

We also evaluate a hybrid resource selection scheme where either per-cluster or per-object information is stored in the resource description of a peer.

Analysis of RS4MI

In the following, different ways of how to best design summaries in case of RS4MI are evaluated. First, summaries storing only per-cluster information are analyzed. Later, hybrid summaries are evaluated. We should note here that *hybrid* in case of RS4MI means storing per-cluster or per-object information. RS4MI can of course, similar to the ap-

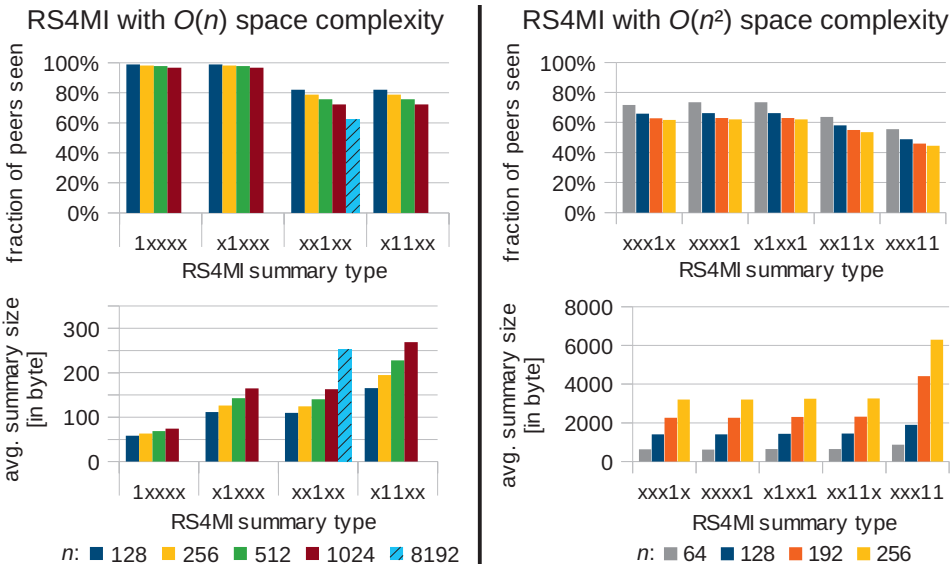


Figure 4: Results of RS4MI for summaries with space complexity $\mathcal{O}(n)$ (left) and $\mathcal{O}(n^2)$ (right).

proaches evaluated in Sect. 4.2 and Sect. 4.3, be extended to store feature objects for peers with few images directly in the summaries. An analysis is part of future work.

RS4MI approaches storing per-cluster information: Fig. 4 (top left) visualizes retrieval performance for resource descriptions with $\mathcal{O}(n)$ space complexity. It can be observed that RS4MI$_{\text{1xxxx}}$ and thus only applying the double-pivot distance constraint does not lead to an acceptable peer selectivity. RS4MI$_{\text{1xxxx}}$ with a bit-vector as underlying data structure however results in very space efficient resource descriptions, even in case of larger values of n (e.g. $n = 1024$ in Fig. 4 (bottom left)).

Comparing RS4MI$_{\text{x1xxx}}$ with RS4MI$_{\text{xx1xx}}$, it can be observed that although both approaches have similar average summary sizes, RS4MI$_{\text{xx1xx}}$ can prune clearly more peers than RS4MI$_{\text{x1xxx}}$. Even RS4MI$_{\text{x11xx}}$ cannot noticeably improve peer selectivity. RS4MI$_{\text{xx1xx}}$ with a very large number of reference objects being used (e.g. $n = 8192$ or even more) seems the best choice amongst the approaches considered in the left part of Fig. 4.

In addition, resource descriptions with $\mathcal{O}(n^2)$ space complexity are analyzed. For these approaches a binning technique is applied in order to reduce summary sizes. Every four byte distance value is quantized into a single byte. The minimum and maximum distance value from the database objects of the external collection to every cluster center c_i is determined. The range between these two boundaries per reference object c_i is uniformly quantized into 253 intervals. From the remaining three values, two are used to represent distance values below and above the boundaries. The third remaining value is used to indicate an empty cluster with no entry. Here, it is again assumed that the minimum and

	gzip	bzip2	lzma	png	paq8o8	webpll
summary size	867.0 B	1020.1 B	863.4 B	880.4 B	803.1 B	777.7 B

Table 3: Average summary sizes for $\text{RS4MI}_{\text{xxx11}}$ with $n = 64$.

maximum distances from feature objects of the external collection to the cluster centers c_i are known to all peers in advance and transfered to them by updates of the P2P software so that all peers can correctly estimate the true distance from the quantized values. However, this information is also small enough to be transfered to participating peers during the operation phase of the P2P IR system.

Fig. 4 (bottom right) shows that the average summary sizes in case of $\text{RS4MI}_{\text{xxx1x}}$, $\text{RS4MI}_{\text{xxxx1}}$, $\text{RS4MI}_{\text{x1xx1}}$, and $\text{RS4MI}_{\text{xx11x}}$ are very similar. According to retrieval performance (cf. Fig. 4 (top right)) $\text{RS4MI}_{\text{xx11x}}$ applying the *MIN* matrix and an array of length n with maximum cluster radii clearly outperforms the other three approaches. Also $\text{RS4MI}_{\text{xxx11}}$ encoding the quantized *MIN* and *MAX* matrices with a small value of n is promising (e.g. $n = 64$). For the feature set being indexed, $\text{RS4MI}_{\text{xxx11}}$ with n being small or $\text{RS4MI}_{\text{xx1xx}}$ with n being big seem to be the most promising RS4MI approaches.

To further reduce the summary sizes of $\text{RS4MI}_{\text{xx1xx}}$ with $n = 8192$, alternative compression algorithms might be suitable. When changing the compression algorithm to bzip2, summary sizes are reduced on average from 253.2 (gzip) to 222.1 byte and to 234.2 byte in case of lzma. Thus, a reduction of approximately 10% seems easily possible[4].

Summary sizes for $\text{RS4MI}_{\text{xxx11}}$ with $n = 64$ can also be reduced. The bzip2 implementation seems to be inappropriate with average summary size noticeably increasing, and also lzma does not lead to a significant reduction (cf. Tab. 3). Thus, in addition to gzip, bzip2, and lzma, three image compression algorithms are tested, where the concatenation of the quantized *MIN* and *MAX* matrices is interpreted as a 2-dimensional 256 bit gray-scale image of size 64×128 pixels. Tab. 3 shows the results. Standard png compression provides some overhead, but paq8o8[5] and especially webp[6] lossless image compression provide more space efficient resource descriptions; webp in particular by significantly improving the memory requirements of the summaries of the peers with images in few clusters.

Hybrid RS4MI approaches storing per-object information: The RS4MI approaches presented so far solely rely on cluster pruning principles. Object pruning and thus the encoding of per-object information in the resource descriptions is not considered. However, the distribution of peer sizes (cf. Fig. 2) indicates many peers with few documents. Thus, at least for peers with very few documents it might be beneficial to encode per-object summary information and apply pivot filtering (cf. Sect. 2).

[4] Additional compression results are based on the at4j library (*http://at4j.sourceforge.net/*). We acknowledge the contributors of at4j and of contributing libraries such as 7-zip (*http://www.7-zip.org/*) and apache commons compress (*http://commons.apache.org/compress/*).

[5] cf. *http://mattmahoney.net/dc/#paq*

[6] cf. *https://developers.google.com/speed/webp/*

	$n = 1$	$n = 2$	$n = 4$	$n = 8$	$n = 12$	$n = 16$
peers seen	97.3%	95.5%	90.9%	82.5%	75.8%	73.5%
summary size	136.5 B	216.9 B	372.5 B	676.2 B	976.1 B	1274.6 B

Table 4: Results when purely applying pivot filtering.

	$s_d = 100$ (8439)	$s_d = 200$ (9612)	$s_d = 400$ (10202)	$s_d = 600$ (10385)	$s_d = 800$ (10459)	$s_d = 1000$ (10503)
$n = 512$	81.1%	76.0%	69.9%	66.0%	63.4%	61.4%
	174.9 B	251.9 B	426.9 B	585.1 B	759.6 B	919.2 B
$n = 4096$	80.3%	75.7%	69.8%	65.9%	63.4%	61.4%
	214.8 B	277.8 B	443.0 B	596.8 B	768.9 B	926.8 B

Table 5: Results for hybrid summaries. Table cells show the fraction of contacted peers (top) and the avg. summary size (bottom). The number of peers applying pivot filtering is given in brackets.

First, we analyze settings where only object-pivot distances (and thus no per-cluster information) are used in the resource descriptions. Tab. 4 shows the results for different numbers of reference objects. Such an undifferentiated approach is inappropriate and results in very big summary sizes for peers with many documents. When using 16 reference objects and thus encoding 16 object-to-pivot distance values per database object, 73.5% of peers are contacted with resource descriptions of 1.3 kB on average.

We also analyze a hybrid resource description scheme with peers choosing between either per-object or per-cluster summarization, depending on *nDocs*, the number of images a peer administers. In order to very roughly estimate the number of possible reference objects per database object for which object-pivot-distances are stored in the summary, the formula *nRefsPerObject* = $\lfloor \frac{s_d}{4 \cdot nDocs} \rfloor$ is applied. The parameter s_d hereby denotes the desired average summary size in byte and a factor of four in the denominator is used since a single distance value is represented as a four byte float. From Tab. 5, it can be seen that this estimate of the average summary size roughly holds. If *nRefsPerObject* > 0, pivot filtering is applied on the basis of per-object resource descriptions. Otherwise, per-cluster summaries $\text{RS4MI}_{\text{xx1xx}}$ are applied as before.

Table 5 visualizes results of the hybrid resource selection scheme when varying s_d and n. The number of peers applying pivot filtering is denoted in brackets. If these results are compared with the ones applying only per-cluster information, several approaches can be outperformed; for example a parameter settings with $n = 512$ and $s_d = 600$ seems promising. However, peer selectivity of $\text{RS4MI}_{\text{xx1xx}}(n = 8192)$ can only be achieved with much bigger average summary sizes, since compression techniques in case of $\text{RS4MI}_{\text{xx1xx}}(n = 8192)$ can dramatically reduce the summary size of peers with documents in only few clusters.

4.5 Brief comparison of approaches

In Sects. 4.2 and 4.3, we saw that techniques yielding an exact representation of small peers, either applying a special treatment for single node M-trees or defining a desired value for k, are promising in situations with a long-tail distribution of the objects over the peers. Of course, such techniques can also be applied for RS4MI—a consideration of this approach is planned for the near future. To assess the potential in large scale networks, let us concentrate on the summary sizes and the selectivity of the basic techniques here.

Table 6 gives a brief overview of different approaches discussed in Sect. 4.2, Sect. 4.3, and Sect. 4.4. All of them result in average summary sizes below 1 kB. Conceptually, the source selection techniques based on M-tree and k-medoid clustering are similar to each other both applying local clustering and transferring medoids and cluster radii. The parametrization of the techniques is crucial for both approaches. In this regard, the k-medoid based local clustering approach with its easy to interpret design parameter k is more handy than M-tree based clustering and also retrieval performance (as briefly summarized in Tab. 6 and in more detail outlined in Sect. 4.2 and Sect. 4.3) does not give a clear evidence for using the approach based on the M-tree. $\text{RS4MI}_{\text{xx1xx}}(n = 8192)$ leads to better retrieval results with significantly smaller average resource description sizes. The number of contacted peers is further reduced by $\text{RS4MI}_{\text{xxx11}}(n = 64)$ at the cost of larger summaries, comparable with those of 2-medoid. Of course, it is also possible to use different RS4MI summary types within a single P2P IR system. We will analyze this in future work.

	peers seen	avg. summary size
M-tree(576;16384)	75.8%	813.2 byte
2-medoid	68.7%	867.7 byte
$\text{RS4MI}_{\text{xx1xx}}(n = 8192)$	62.2%	253.0 byte
$\text{RS4MI}_{\text{xxx11}}(n = 64)$	57.2%	880.7 byte

Table 6: Comparison of the different approaches with results averaged over ten runs.

5 Conclusion and Outlook

We presented RS4MI—an exact resource selection scheme for general metric spaces and showed how the processing of range queries can be performed. RS4MI can outperform an M-tree based exact resource selection scheme and a selection scheme based on k-medoid clustering w.r.t. the number of peers which are contacted and w.r.t. memory requirements. In case of range queries, RS4MI is especially beneficial in scenarios when the memory requirements of the database objects are huge since RS4MI does not store them in the summaries. Furthermore, with large numbers of reference objects fine-grained adaptations are possible w.r.t. the avg. summary size. Peer(s) monitoring the network can adaptively adjust summary sizes since every peer per se knows the summary size of all other peers.

In future work, we will also analyze the processing of k-NN queries and strategies for determining the reference objects. In addition, we will derive approximate extensions providing a good compromise between runtime performance and adequate retrieval quality.

References

[BDP04] S. Berretti, A. Del Bimbo, and P. Pala. Merging Results for Distributed Content Based Image Retrieval. *Multimedia Tools Appl.*, 24(3):215–232, 2004.

[BEMH07] D. Blank, S. El Allali, W. Müller, and A. Henrich. Sample-based Creation of Peer Summaries for Efficient Similarity Search in Scalable Peer-to-Peer Networks. In *Intl. SIGMM Workshop on Multimedia Information Retrieval*, pages 143–152, Augsburg, Germany, 2007. ACM.

[BH10] D. Blank and A. Henrich. Binary Histograms for Resource Selection in Peer-to-Peer Media Retrieval. In *Proc. of LWA Workshop – Lernen, Wissen, Adaptivität*, pages 183–190, Kassel, Germany *www.kde.cs.uni-kassel.de/conf/lwa10/papers/ir4.pdf* (last visit: 27.9.2012), 2010.

[BKSS07] B. Bustos, D. Keim, D. Saupe, and T. Schreck. Content-Based 3D Object Retrieval. *IEEE Comput. Graph. Appl.*, 27(4):22–27, July 2007.

[CNBYM01] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín. Searching in Metric Spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 426–435, Athens, Greece, 1997. Morgan Kaufmann.

[CZBP10] S. A. Chatzichristofis, K. Zagoris, Y. S. Boutalis, and N. Papamarkos. Accurate Image Retrieval Based on Compact Composite Descriptors and Relevance Feedback Information. *Intl. J. of Pattern Recognition and Artificial Intelligence*, 24(2):207–244, 2010.

[DD09] M. M. Deza and E. Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 1st edition, 2009.

[DVNV10] C. Doulkeridis, A. Vlachou, K. Nørvåg, and M. Vazirgiannis. Part 4: Distributed Semantic Overlay Networks. In X. Shen, H. Yu, J. Buford, and M. Akon, editors, *Handbook of Peer-to-Peer Networking*, pages 463–494. Springer Science+Business Media, 1st edition, 2010.

[EBMH08] S. El Allali, D. Blank, W. Müller, and A. Henrich. Image Data Source Selection Using Gaussian Mixture Models. In *Adaptive Multimedia Retrieval: Retrieval, User, and Semantics: 5th International Workshop*, pages 170–181, Berlin, Heidelberg, 2008. Springer LNCS 4918.

[EMH+06] M. Eisenhardt, W. Müller, A. Henrich, D. Blank, and S. El Allali. Clustering-Based Source Selection for Efficient Image Retrieval in Peer-to-Peer Networks. In *Proc. of the 8th Intl. Symp. on Multimedia*, pages 823–830, San Diego, CA, USA, 2006. IEEE.

[ERO+09] B. M. Elahi, K. Römer, B. Ostermaier, M. Fahrmair, and W. Kellerer. Sensor ranking: A primitive for efficient content-based sensor search. In *Proc. of the 8th Intl. Conf. on Information Processing in Sensor Networks*, pages 217–228, San Francisco, CA, USA, 2009. ACM.

[HCS09] X. Hu, T. Chiueh, and K. G. Shin. Large-scale malware indexing using function-call graphs. In *Proc. of the 16th ACM Conf. on Computer and Communications Security*, pages 611–620, New York, NY, USA, 2009. ACM.

[KLC02] D.-H. Kim, S.-L. Lee, and C.-W. Chung. Heterogeneous image database selection on the Web. *The Journal of Systems and Software*, 64:131–149, 2002.

[KW11] M. Kunze and M. Weske. Metric Trees for Efficient Similarity Search in Large Process Model Repositories. *Business Process Management Workshops*, 66:535–546, 2011. Springer Lecture Notes in Business Information Processing.

[LLOS07] M. Lupu, J. Li, B. C. Ooi, and S. Shi. Clustering wavelets to speed-up data dissemination in structured P2P MANETs. In *Proc. of the 23th Intl. Conf. on Data Engineering*, pages 386–395, Istanbul, Turkey, 2007. IEEE.

[LSR+08] H. Liu, D. Song, S. M. Rüger, R. Hu, and V. S. Uren. Comparing Dissimilarity Measures for Content-Based Image Retrieval. In *Proc. of the 4th Asia Infomation Retrieval Symposium*, pages 44–50. Springer LNCS 4993, 2008.

[MEH05a] W. Müller, M. Eisenhardt, and A. Henrich. Fast retrieval of high-dimensional feature vectors in P2P networks using compact peer data summaries. *Multimedia Systems*, 10(6):464–474, 2005.

[MEH05b] W. Müller, M. Eisenhardt, and A. Henrich. Scalable summary based retrieval in P2P networks. In *Proc. of the 14th Intl. Conf. on Information and Knowledge Management*, pages 586–593, Bremen, Germany, 2005. ACM.

[MES10] M. Mohajer, K.-H. Englmeier, and V. J. Schmid. A comparison of Gap statistic definitions with and without logarithm function. *Online:* http://arxiv.org/abs/1103.4767*, (last visit: 27.9.2012)*, 2010.

[MKB79] K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate Analysis*. Academic Press, London, 1979.

[NBZ11] D. Novak, M. Batko, and P. Zezula. Metric Index: An efficient and scalable solution for precise and approximate similarity search. *Inf. Syst.*, 36:721–733, June 2011.

[NF03] H. Nottelmann and N. Fuhr. Decision-theoretic resource selection for different data types in MIND. In *Distributed Multimedia Information Retrieval: Proc. of the Intl. Workshop on Distributed Information Retrieval*, pages 43–57. Springer LNCS 2924, 2003.

[PNT11] A. A. Paterlini, M. A. Nascimento, and C. Traina Jr. Using Pivots to Speed-Up k-Medoids Clustering. *J. of Information and Data Management*, 2(2):221–236, 2011.

[Rou87] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, 20:53–65, 1987.

[Sam06] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[SB11] T. Skopal and B. Bustos. On Nonmetric Similarity Search Problems in Complex Domains. *ACM Computing Surveys*, 43(4):34:1–34:50, October 2011.

[SKG02] S. Saroiu, P. Krishna Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *ACM/SPIE Multimedia Computing and Networking*, pages 156–170, San Jose, CA, USA, 2002.

[Sko06] T. Skopal. *Similarity Search In Multimedia Databases*. PhD thesis, Charles University, Prague, Czech Republic, 2006.

[SS11] M. Shokouhi and L. Si. Federated Search. *Foundations and Trends in Information Retrieval*, 5(1):1–102, 2011.

[TWH01] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a dataset via the Gap statistic. *J. R. Stat. Soc. Ser. B*, 63(2):411–423, 2001.

[Woj02] A. Wojna. Center-Based Indexing in Vector and Metric Spaces. *Fundam. Inf.*, 56:285–310, 2002.

[YSMQ01] C. Yu, P. Sharma, W. Meng, and Y. Qin. Database selection for processing k nearest neighbors queries in distributed environments. In *Proc. of the 1st ACM/IEEE Joint Conf. on Digital Libraries*, pages 215–222, New York, NY, USA, 2001.

[YY07] M. Yan and K. Ye. Determining the Number of Clusters Using the Weighted Gap Statistic. *Biometrics*, 63:1031–1037, 2007.

[ZADB05] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*. Springer New York, Inc., Secaucus, NJ, USA, 2005.

Logical recovery from single-page failures

Goetz Graefe; Bernhard Seeger
Hewlett-Packard Laboratories; Philipps-Universität Marburg
goetz.graefe@hp.com; seeger@informatik.uni-marburg.de

Abstract: Modern hardware technologies and ever-increasing data sizes increase probability and frequency of local storage failures, e.g., unrecoverable read errors on individual disk sectors or pages on flash storage. Our prior work has formalized single-page failures and outlined efficient methods for their detection and recovery.

These prior techniques rely on old backup copies of individual pages, e.g., as part of a database backup or as old versions retained after a page migration. Those might not be available, however, e.g., after recent index creation in "non-logged" or "allocation-only logging" mode, which industrial database products commonly use.

The present paper introduces techniques for single-page recovery without backup copies, e.g., pages of new indexes created in allocation-only logging mode. By re-deriving lost contents of individual pages, these techniques enable efficient recovery of data lost due to damaged storage structures or storage devices. Recovery performance depends on the size of the failure and of the required data sources; it is independent of the sizes of device, index structure, etc.

1 Introduction

Efficient recovery from transaction, media, and system failures has long been a hallmark of database technology and has been employed in other domains as well, e.g., metadata in modern file systems. All these recovery techniques relied on copies of old and new values of pages, records, and individual fields. For example, when a transaction fails and needs to roll back its updates, "undo" log records or "undo" components of "redo-undo" log records permits copying old values into the appropriate pages and records. Logical and "physiological" logging rely on the same principle; their logical aspect is the location of the record, which might have moved from one page to another between the original "do" action and the "undo" action, e.g., due to a node split in a B-tree index.

Single-page failures, a fourth class of database failures that matches failure scenarios of high-density disk drives and of present and future semi-conductor storage, has recently been described [GH 12]. The suggested recovery techniques for an unreadable or inconsistent page rely on recovery logs as commonly used in database systems and on an earlier copy of the page. These copies may come from a database backup, delayed deallocation after a page migration, or a log record describing initial formatting of a page newly allocated from free space. Both the earlier page image and the recovery log rely on copying and on copies, either of an entire page or of individual records and field values.

For some operations commonly used in database system, however, these techniques do not work. For example, creation of a secondary index for a table usually is implemented without logging the individual index entries or the new index pages. Instead, merely the page allocation actions are logged. For example, if 500 index entries of 16 bytes fit on each database page of 8 KB, logging individual index entries of 16 bytes may take log records of 40 bytes (equivalent to 20 KB of log volume for each database page of 8 KB); logging entire new pages takes 8 KB per page plus a little overhead; and logging only the page allocation may take 80 bytes of log per database page of 8 KB. Thus, allocation-only logging produces about 100× less log volume than full-detail logging.

Before a transaction with non-logged index creation can commit, it must flush all new index pages from the buffer pool to persistent database storage. This has been termed a "no steal – force" policy [HR 83]. If the system suffers a system or media failure soon thereafter, recovery of the newly created secondary index relies on repeating the entire index operation, i.e., it reaches back to the base table. Recovery of the index contents from log records is not possible after allocation-only logging.

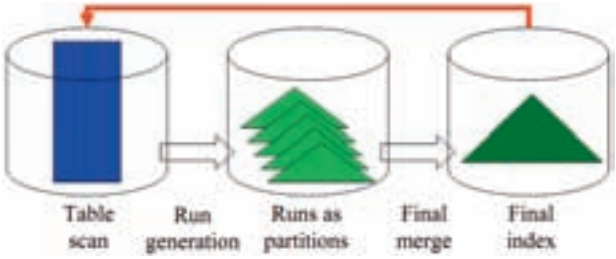


Figure 1. Traditional recovery.

Figure 1 schematically illustrates this recovery process. Runs in the external merge sort are shown as partitions of a B-tree; the final merge step creates the desired B-tree index; and recovery after failure reaches back to the base table, i.e., repeats the entire effort of index creation.

After a successful index creation, the entire new index must be backed up, too, during the next backup of the recovery log; otherwise, the log backup may contain log records describing updates to index pages that cannot be repeated in a possible later failure. In other words, while "allocation-only logging" optimizes the size of the recovery log, it does not reduce the volume of the log backup. Thus, existing techniques for "non-logged" or "allocation-only logged" index creation solve only part of the problem.

These techniques also fail to enable recovery of single-page failures in the newly created index, at least until the index has been backed up as part of a database backup or a log backup. If an individual index page (or a small set of index pages) becomes unreadable, it is also unrecoverable except by dropping and recreating the entire index. Thus, a new technique is needed for single-page recovery for database pages filled by non-logged operations.

This lack of support includes, unfortunately, many database utilities that modify the logical and physical database design. In addition to index creation, this includes initial materialization of views as well as splitting a table in two vertical slices in order to permit new one-to-many or many-to-one relationships. For example, if the original database design is limited to one address per customer, yet an application change requires multiple shipping addresses (e.g., to enable gift orders), the original table for customers must be split into two.

It is desirable to enable recovery from single-page failures even after changes in the logical and physical database design. Prior designs and techniques failed at this goal; therefore, the present paper proposes a design to fill this void. The foundation is focused repetition of an operation such as index creation. In order to ensure efficiency, the design relies on specific data structures and on incremental creation and optimization of new data structures, e.g., new secondary indexes.

Future work will cover more complex data structures, e.g., materialized views and their initial materialization. Once a view has been materialized, the techniques here suffice to add secondary indexes. For the initial materialization step, the scan techniques (including snapshot isolation by single-page "undo" recovery) need to be extended to multiple data sources.

The remainder of this paper is organized as follows. The next section reviews a variety of related work that influenced and informs the proposed design. Section 3 lays out the principles and foundation for the design, whereupon Section 4 applies them to a specific case of single-page recovery in newly created secondary indexes. Section 5 demonstrates the performance potential with a few preliminary experiments and Section 6 offers our conclusions from this research effort.

2 Related prior work

The following reviews relevant system designs, failure classes and their recovery techniques, and data structures. The subsequent sections combine those prior techniques into logical recovery from small failures by re-deriving lost contents.

2.1 Bubba

A proposal for data storage, reliability, and recovery in the Bubba database machine [CAB 88] suggested that a lost index may be recovered from one or more other indexes, with those indexes containing a super-set of the columns in the lost index. Novel at the time, it seems that this design has never been implemented.

2.2 Vertica

In the Vertica columnar database [LFV 12], tables are stored as one or more "projections." Each projection has a sort order. A "super-projection" is similar to a traditional primary index as it contains all rows and all columns of the table. It is different from a traditional primary index as it may include functionally dependent columns from other tables and it uses

a separate file for each column. Other projections store a subset of columns for all rows, similar to a traditional secondary index.

For reliability, each projection may be stored multiple times for "k-safety" against data loss even in the case of losing k copies (e.g., 2 copies for 1-safety). These "buddy projections" must have the same column set (as well as all rows of the table) but may have a different sort order. In case of a failure, recovery of one projection may use its buddy projection, i.e., may require a sort operation.

In contrast, our work focuses on efficient recovery of parts of an index (e.g., a single page or a few pages) that does not require a full sort operation. Recovery after loss within a single column only is left to future work.

2.3 Traditional index operations

Traditional index operations include creation, defragmentation, removal, consistency check and repair, etc. of primary (clustered) and secondary (non-clustered) indexes on tables and (materialized) views. These operations may run offline (with a shared lock on the table) or online (permitting updates by concurrent transactions), logged (each index entry or each index page) or non-logged (also known as allocation-only logging), using temporary space (for runs of the preparatory external merge sort) or in target space (recycling target space during the final merge), serial or parallel (in scan, sort, or write), etc. The indexes may be hashed or sorted, single- or multi-dimensional, unique or non-unique (permitting duplicate key values), uncompressed or compressed (using run-length encoding, prefix- and suffix truncation [BU 77], etc.), using pointers (e.g., record identifiers or key values in the primary index) or bitmaps, partitioned or not, versioned (for snapshot isolation and multi-version concurrency control) or not, etc.

This variety of indexes and of index operations may seem bewildering except for developers working on or with full-featured commercial databases. Therefore, we focus here on creation of secondary B-tree indexes. Applying the principles introduced here to all indexes and index operations certainly is a significant development effort but not a research problem. The exception to this statement are online index operations, which are considered later, e.g., in Section 4.4.

With respect to keeping runs in the target space of the database, we assume sufficient space to keep 2-3 copies of each index entry, i.e., runs are not deleted immediately after a merge operation but somewhat later. Moreover, we assume that logging all new index entries, with a log record per index recovery or per index page, is prohibitive due to the required space in the recovery log, as is the case in most real-world index operations.

2.4 Traditional failure classes

The traditional failure classes are transaction failures, media failures, and system failures [G 78]. Early designs for recovery from those failures relied on "idempotent" recovery actions, which all relied on byte-for-byte copying but also limited the finest granularity of locking to page locks. Subsequent concurrency control and recovery techniques enable row-

level locking (including key range locking in B-tree indexes) but still copy records and fields to and from the recovery log [G 12, HR 83, MHL 92, W 91]. In other words, detailed logging remains required and both logging and recovery still copy field values, records, or pages.

2.5 Single-page failures

This recently proposed fourth failure class and its proposed recovery technique [GH 12] is another prototypical example for "recovery by copy." Starting with a backup page copy, replaying "redo" log records obtains an up-to-date instance of a data page. Recovery might be achieved after reading tens or at most hundreds of log records, i.e., within about a second and thus much faster than media recovery.

The default recovery technique for single-page failure relies on traditional backups and a traditional recovery log, i.e., copying records or field values between database page and recovery log. The locations of both the backup page and the most recent log record are kept in the "page recovery index," one per database. Recovery of a single-page failure from a formatting log record (i.e., the operation immediately following allocation from free space) has aspects of both recovery from a copy and recovery by re-deriving contents. On the other hand, the log record with formatting parameters is more a compressed copy than a recipe for deriving database contents, which is the focus of the recovery techniques in this paper.

Logical recovery, as proposed in a subsequent section, is similar to single-page recovery in the sense that it recovers individual pages rather than an entire device. Logical recovery recovers an entire key range, however, not just a single page at a time. More significantly, it recovers the key range by re-deriving the lost contents by repeating the original logic rather than copying the lost contents from a backup and from log records.

2.6 Self-repairing indexes

Self-repairing indexes [GKS 12] combine two facilities, self-diagnosing faults and self-healing. Faults may be unreadable storage pages or implausible page contents, e.g., an inconsistency between a parent node and a child node or between two neighboring child nodes. Self-healing requires efficient automatic recovery of the correct, up-to-date page contents.

Symmetric fence keys in each B-tree page enable continuous, incremental, and comprehensive verification of all cross-node invariants of a B-tree structure [GS 09], i.e., self-diagnosing indexes. Self-healing can be achieved by moving information for single-page recovery from the database-wide page recovery index into the index itself. In a B-tree variant with only one (incoming) pointer per node, e.g., a Foster B-tree [GKK 12], this information can be associated with each child pointer.

The resulting self-repairing B-tree is just one example for localized detection and recovery of errors in a complex data structure. Frequent local (and thus inexpensive) checks enable efficient root cause analysis during quality assurance as well as reliable data structures after

deployment. Embedding and maintaining consistency information within the B-tree data structure requires a single pointer to each node, e.g., Foster B-trees.

Compared to the initial design for recovery from single-page failures, self-repairing B-trees differ in the bookkeeping, i.e., keeping track of the latest log record for each page without a page recovery index. Logging and recovery are unchanged from the original design for single-page failures, i.e., continue to rely on copying records and field values between database page and recovery log.

2.7 Partitioned B-trees

Partitioned B-trees [G 03] are standard B-trees with a partition identifier added as prefix to the user-defined index key. Prefix truncation (compression) reduces the additional storage requirement to one integer per page in most pages. The resulting indexes preserve sort order (within each partition), enable ordered scans (by merging), and support (reasonably) efficient query execution for key range predicates. In addition to the standard advantages of B-trees, partitioned B-trees enable incremental creation and optimization. Each step produces a valid index, even the initial extraction of index entries from the base table and even if a merge step covers only a partial key range within the index.

Partitioned B-trees can be used for efficient loading (adding new data as one or more memory-sized partitions – the advantage is that loading proceeds at full sequential write bandwidth but the index is immediately complete and searchable), for index creation by external merge sort (with each run stored as a B-tree partition, not as a traditional run file – the advantage is that the index is complete and can be searched immediately after run generation), and for sorting (external merge sort with deep read-ahead, parallel merge using range partitioning, 'pause and resume' without delay and without duplicated effort).

2.8 Adaptive indexing

If the optimal set of indexes for a database or for a table cannot be predicted, adaptive indexing creates useful indexes as side effect of query execution. If the set of desired indexes can be predicted, they can be defined in the catalogs but data movement and data structure optimization can be accomplished as side effect of query execution. Alternatively, the catalogs might also indicate indexes that are permissible or prohibited. An index tuning tool may set such properties, and query processing can take the information as guidance during query optimization and query execution. For example, query optimization may assume that a desirable index will exist at run-time even if it does not exist at compile-time. Repeated query execution will, as side effects and in multiple steps, create and optimize such an index.

There are two forms of adaptive indexing, plus hybrids [IMK 11]: database cracking [IKM 07] optimizes in-memory column stores, whereas adaptive merging [GK 10] optimizes partitioned B-trees. Each form of adaptive indexing has two types of steps: initial index creation and incremental index optimization. In database cracking, the initial index is a single unsorted partition and each optimization step divides an existing partition using a Pivot key equal to the key value of an exact-match query or to an end point of a range query. In

adaptive merging, run generation produces the initial index and each optimization step applies one merge step to key values within or around the range query or the exact-match query.

For side effects with acceptable efficiency, concurrency control and recovery must be practically free. Full detailed logging is not acceptable. Concurrency control is a solvable problem, because index optimization is merely a change in physical database representation, not in logical database contents; thus, merely latches are required but not locks [GHI 12]. Logging and recovery, however, could introduce excessive overhead to query execution.

A simple and effective technique is to let only one thread (one query) perform a merge step for a given key range and let all other queries access that key range in the existing data structure in read-only mode. When the merge step is complete, all new query executions may use the merge result; all existing executions may continue using the older partitions in read-only mode. Only when all existing executions are complete, the older partitions may be removed. In other words, it is not the merge step that removes its input partitions and reclaims their space but an asynchronous process invoked based on usage. A pragmatic implementation might use fixed key ranges and reference counting for each key range, which permits space reclamation as soon as possible but also multiple threads (queries) merging disjoint key ranges at the same time.

3 Recovery by re-deriving contents – principles

In order to re-derive lost database contents, the source of the last derivation step must remain available. For example, if the result of a merge step is lost, it can be recovered if the merge inputs still exist. If only a small section of the merge result is lost, and if small sections of source and destination, e.g., specific key ranges, can be accessed directly, then the lost section can be recovered very efficiently.

Thus, we propose to retain these merge runs even after index creation is complete. Delayed removal of such intermediate files adds little cost; storage space is nowadays plentiful and inexpensive in most environments. There is no need to include these merge runs in the next database backup, and in fact the next database backup enables more efficient recovery techniques than logical single-page recovery.

This is somewhat similar to data processing of years past, with a master tape and files with recent changes, with new master tapes created by merging old master tape and all changes, and with a lost master recovered by repeating the appropriate merge step. The difference of the proposed techniques and those “ancient” techniques is that only the required key ranges are recovered, with no need to repeat completed steps in their entirety.

The proposed recovery techniques are quite different from recovery using copies or replicas. If each data page exists twice (or thrice, or even more times), then a single lost copy can be readily re-created simply by copying. The proposed technique holds multiple (typically two) copies of the logical contents but only one copy of each physical page. If some data page is lost, there is no way to recover the loss by copying. Instead, a processing step must be

performed. In the simplest case, which is the focus here, a processing step must be repeated. Ideally, it is repeated only partially, optimized to re-produce the lost data and no more.

Logical recovery by re-deriving database contents and their data structures requires that data processing steps are non-destructive. In other words, rather than modifying an existing structure, the original derivation step (as well as the re-derivation steps during logical recovery) must merely read the existing data structure and create new ones. For example, database cracking (i.e., adaptive index improvements in an in-memory column store by partitioning steps similar to those of quicksort) does not qualify for logical recovery, because the partitioning steps occur in place in order to minimize the number of data items that need to move. On the other hand, in the other prototypical adaptive indexing technique, adaptive merging, each step merges multiple runs and put the result into a different run. Thus, even if adaptive merging keeps all runs in a single B-tree, the merge input remains unchanged in each merge step. Thus, adaptive merging and its data structures can serve as prototypical use case for logical recovery by re-deriving lost data pages, but the technique also applies to other index formats and indexing techniques as well as to materialized and indexed views.

The following discussion focuses on failure and recovery of leaf pages in B-tree indexes. Non-leaf pages, typically only 1% to 1‰ of all pages in a B-tree, should be fully logged such that existing recovery techniques suffice, e.g., log-based single-page recovery [GH 12].

4 Recovery of index pages and key ranges

Partitioned B-trees and adaptive merging lend themselves to logical single-page recovery, i.e., re-deriving lost contents from retained prior data. This is due to index operations proceeding in distinct simple steps with valid and useful states in between, even if each merge step merges only a small key range. The following sub-sections cover index creation, index optimization, index maintenance, and recovery after updates.

4.1 Index creation

Creation of new secondary B-tree indexes usually employs an external merge sort. The first step, run generation, scans the table's primary data structure, extracts all required information for future index entries, and produces initial runs for the external merge sort, perhaps as partitions in a partitioned B-tree. For the discussion here, index creation is complete when all index entries are in the future index structure. Index optimization merges partitions in order to organize all index entries into a single sorted sequence with query and update efficiency of a traditional B-tree.

Should one of the initial runs become unreadable, it can be recovered if the appropriate part of the primary data structure can be identified, retrieved, and re-sorted. If only a key range within an initial run becomes unreadable, this key range translates to a predicate when re-scanning the primary data structure, which reduces the sort effort but not the scan effort.

Figure 2 illustrates the technique, where recovery of a single run (center, red) reaches back to the original table (left, blue) but scans only a part of it (left, red). The final index (right) does

not participate in this scenario. It might not even exist yet and is thus drawn with dashed lines. A comparison with Figure 1 identifies the difference: whereas traditional logical recovery can re-derive only the final index and only from the original table using a complete table scan, single-step recovery can re-derive individual runs by scanning only parts of the original table.

Figure 2. Single-step recovery: run generation.

Such recovery works very efficiently if each partition in the new index maps to a specific segment of the source data structure. Ideally, a table's primary data structure is a B-tree index (a clustered index also known as index-organized table), the scan providing input to run generation uses the index order (as opposed to an allocation-order scan), and run generation proceeds in read-sort-write cycles (e.g., using quicksort, not using a continuous process such as replacement selection). In this case, the read-sort-write cycles and the index-order scan provide a simple mapping from a run in the new index to a key range in the data source, and the primary index provides efficient access to just that key range. In contrast, run generation by replacement selection permits only less precise mappings, and an allocation-order scan or a primary data structure other than an index requires an unusual predicate on a page range rather than a standard predicate on a key range.

If only a single page within a run is unreadable, it can be re-derived efficiently using a partial scan of the original table. Differently from the partial table scan in Figure 2, this partial scan applies a predicate matching the key range of the unreadable page. If a B-tree represents each run or if a single partitioned B-tree represents all runs, the parent page in the B-tree structure can provide the required key range.

Figure 3. Single-page recovery by run generation.

Figure 3 illustrates recovery of a single page in a partition. Scanning the appropriate fraction of the data source quickly produces the index entries that belong into the unreadable page of the index partitions.

4.2 Index optimization

Run generation is only the first step of index creation. The second step, required only for very large tables and indexes, merges initial runs to form intermediate runs. The third step merges a small set of final runs to form the desired index.

Should an intermediate run or a key range within such a run become unreadable, logical recovery repeats the merge logic for that key range. The same is true for the final merge step producing the final, fully optimized index: Should a part of the final index become unreadable, it can be recovered by re-merging data from the final runs.

Repeating a merge step for the entire domain of user-defined keys requires that the input runs have been retained, i.e., not deleted immediately during or after the merge step. Repeating a merge step only for a limited key range requires that the data in the required key range can be retrieved efficiently, i.e., these runs are organized in a partitioned B-tree.

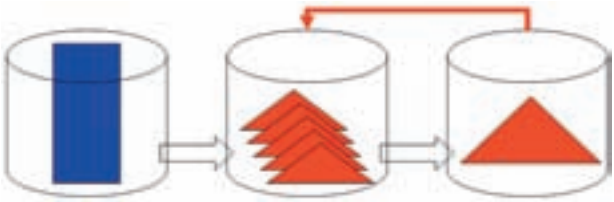


Figure 4. Single-step recovery by merging.

Figure 4 illustrates single-step recovery from intermediate runs, i.e., it complements the single-step recovery illustrated in Figure 2. If intermediate runs still exist, recovery of the final index can omit table scan and run generation, instead repeating only the final merge step.

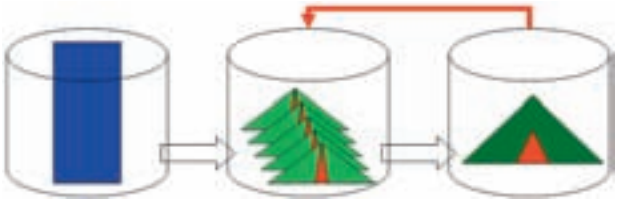


Figure 5. Single-page recovery by merging.

Figure 5 illustrates single-page recovery by partially repeating a merge step. If a single page (or a small set of pages) is unreadable in the final index, intermediate runs stored in a B-tree permit direct access to the required key range. A short merge operation can reproduce precisely the unreadable key range without wasting any effort on other key ranges.

During merge steps in a partitioned B-tree, e.g., during adaptive merging, a limited merge fan-in reduces the memory allocation required for the side effect of query execution. Thus, there is a tradeoff between efficiency of a merge step (favoring a high merge fan-in) and the overhead of memory allocations (favoring a small merge fan-in). Logical recovery adds another consideration: a small merge fan-in permits logical recovery from less source data. In other words, the reliability of the storage technology and the probability of data loss requiring logical recovery influence the heuristics setting the merge fan-in. Of course, the same arguments apply to index creation and the memory allocation for run generation.

4.3 Maintenance of existing indexes

An entire index is fully optimized when only a single partition remains. In that state, searches and updates in a partitioned B-tree are just as efficient as in a traditional B-tree without partitions. This single final partition is permanent in the sense that it will not serve as merge input in future merge steps. A key range is fully optimized when all index entries within the key range are in the partition intended to be the only permanent partition. Note that an index or a key range can lose this status, e.g., when a large load operation adds new partitions to the B-tree.

When a key range is fully optimized, updates modify the permanent partition. Otherwise, it might be most pragmatic to leave each partition read-only once a merge step has created it, while updates go to one or two dedicated partitions that absorb all updates. These partitions remain in memory and sort the updates similarly to run generation by replacement selection. Read-only and read-write partitions require different techniques for logical recovery.

Read-only partitions can be recovered, if necessary, simply by repeating the run generation logic or the merge step that created them (as discussed in Section 4.2). Read-write partitions require recovery in two stages. First, the original partition contents are recovered by repeating the logic that created the partition. Second, single-page “redo” recovery must carry pages forward by finding and replaying the appropriate log records.

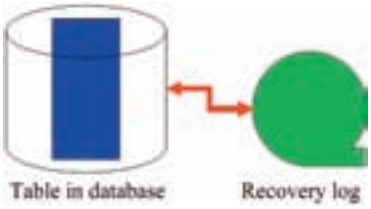


Figure 6. Traditional log-based recovery.

Figure 6 sketches traditional recovery based on write-ahead logging. Update operations produce log records with before- and after-images of database records, index entries, and individual column values, whereas "redo" and "undo" recovery copy from the log into the database. This technique also applies to read-write partitions during the second stage of recovery.

Read-write partitions absorb not only insertions but also deletions. A deletion in a read-only partition inserts a "tombstone" or "anti-matter" record into the appropriate read-write partition. When merged with a valid record, neither anti-matter nor valid record with the same key value appear a merge output or a query result. An update is a deletion and an insertion.

4.4 Updates between index operations and their recovery

If an index contains the same data item twice, typically once in a recent merge output and once in a retained merge input, then only the latter is used to answer queries. If a recent merge output is a read-write partition, then updates modify only that partition. In other words, each update is applied only once. A partition that has been merged into another one is frozen (no further queries, no further updates) and therefore can serve as backup (or input) during recovery. Once a former merge output has served as merge input, the oldest merge input is no longer useful as backup and is dropped. In other words, each data item exists only in two places, not in three or more places.

Online index operations permit concurrent updates, contrary to offline index operations that retain read locks on the entire table for the operation's entire duration. Depending on the precise timing of updates and in particular properties of index operations, it may be impossible to repeat a merge operation if the merge output subsequently becomes unreadable. Moreover, a merge output may become unreadable only after some updates subsequent to the merge operation. In both those cases, log-based recovery (based on write-ahead logging during updates) complements logical recovery.

In order to repeat a prior merge step in spite of subsequent updates to the merge input, rolling back pages of the merge input ensures precise repetition. Thus, if logical recovery encounters a page with a PageLSN (timestamp, pointer to a log record) newer than the original merge operation, single-page "undo" recovery based on per-page chains of log records rolls back a temporary page copy in the buffer pool. This design for single-page rollback is very similar to the implementation of snapshot isolation in Oracle databases. After the repeated merge operation is complete, single-page "redo" recovery can repeat updates applied after the original merge operation.

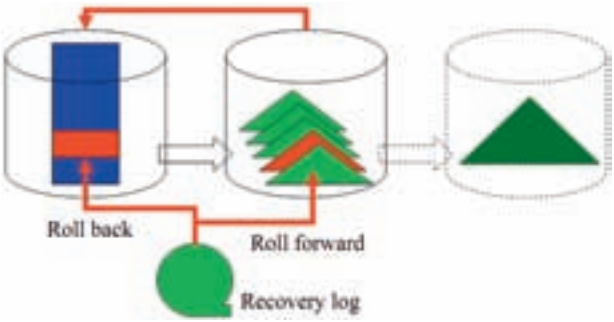


Figure 7. Single-step recovery of run generation and updates.

Figure 7 adds individual updates and their log-based recovery to the scenario of Figure 2. While rollback of merge input pages occurs during the scan feeding the repeated merge operation, roll forward of merge output may occur either in a subsequent step or during the merge operation, i.e., immediately after the merge logic has filled an entire output page.

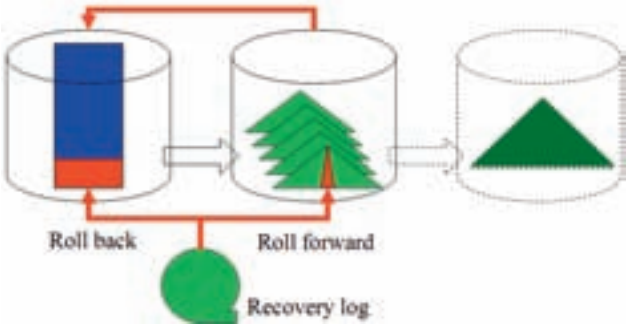


Figure 8. Single-page recovery of run generation and updates.

Figure 8 illustrates logical single-page recovery by repeating run generation. The scan repeats only that part of the source scan that corresponds to the run with lost pages; the run generation logic consumes all scanned data but overwrites only the lost pages. The scan integrates rollback of pages in the original table, if required. Note that single-page rollback must precede predicate evaluation. Writing the run also rolls forward the page (or pages) found unreadable in the run. Alternatively, single-page “redo” recovery may be a separate step.

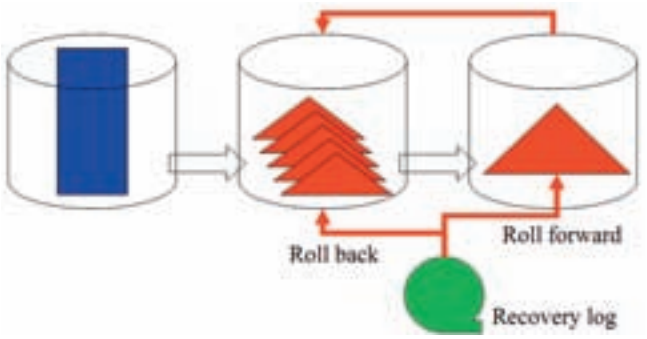


Figure 9. Single-step recovery of merging and updates.

Figure 9 illustrates recovery of an entire merge operation with updates to both merge input and merge output. The scans feeding the merge logic roll back input pages in order to enable precise repetition of the original merge operation; the merge output pages are rolled forward either immediately or in a subsequent step.

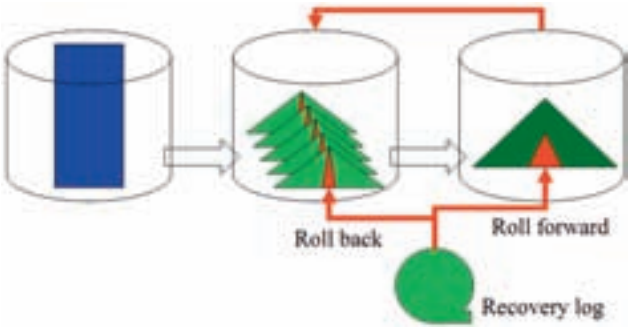


Figure 10. Single-page recovery of merging and updates.

Finally, Figure 10 illustrates single-page recovery in the final index by extracting and merging appropriate pages in intermediate partitions. Log-based single-page "undo" applied to all merge input pages ensures precise repetition of the merge logic for the required key range; log-based single-page "redo" of the merge output ensure correct final index contents and structure.

4.5 Summary of logical recovery for indexes

In summary, logical recovery for index operations merely requires retaining data structures, i.e., delaying their removal from temporary storage space. Doing so enables efficient recovery of both large and small failures, e.g., single-page failures in intermediate data

structures (e.g., runs during index creation) and in final index structures. While the prior design for single-page recovery requires extensive logging, the new design relies on data structures created in the standard sequence of steps.

In addition to offline index operations (with the underlying table locked in read-only mode), logical recovery also support online index operations (i.e., updates by concurrent user transactions). Concurrent updates must be logged using standard write-ahead logging with appropriate "undo" and "redo" information. Using techniques known from Oracle's implementation of snapshot isolation, single-page "undo" takes the data source back to the time of the original index creation step; using techniques from the original design for single-page recovery, single-page "redo" recovery rolls the merge output forward to reflect updates subsequent to the original merge step.

5 Performance and scalability

This section reports the results of a preliminary performance comparison of logical recovery with the traditional approaches to recovery, with the focus on secondary indexes and their creation in a row-store database. The first set of experiments is designed to assess the performance improvements due to logical recovery of an entire index in comparison to reloading the entire index from scratch. A second set of experiments focuses on cases where only a few pages of the index need recovery. All these experiments assume a static scenario with no updates after index creation. In other words, the experiments reflect the technique illustrated in Figure 4 and Figure 5.

Parameter	Values
Main memory	20, 80 MB
Page size	8 KB
Tuple size	40, 200, 500 bytes
Tuple count	100 M
Key size	8 bytes
Pointer size	8 bytes

Figure 11. Experimental parameters.

5.1 Experimental environment

The experiments use a base relation with 100'000'000 tuples of 200 bytes, for a table size of 20 GB. A secondary B-tree index is created on an attribute of size 8 bytes. An index entry of this B-tree occupies 16 bytes; 8 bytes for the search key (attribute) and 8 bytes for a pointer.

Search keys are to be uniformly distributed. All experiments use a page size of 8 KB, a setting that is in agreement with the recommendations of database vendors.

All software used in these experiments are implemented in Java using the open-source XXL library [BBD 01]. In particular, the following components of XXL are employed: external sorting algorithm with replacement-selection, B-tree implementation, and index loading algorithm. All experiments are conducted on a machine with an Intel Core i7 2600 / 3.4 GHz with 8 GB of main memory and a WD Caviar Black WD1002FAEX disk with 64 MB cache. Rather than using the ordinary I/O interface, this implementation uses the raw interface to avoid the interference of buffering in the operating systems. Note however, that caching and prefetching are common features within modern disks today. This standard feature remained enabled during the experiments. However, each recovery experiment started with clearing the disk cache (simply by reading useless data). We conducted all experiments for various main memory sizes. Because the observations did not differ qualitatively, the following presents only the results for an available memory of 20 MB or 2'500 pages. Figure 11 summarizes the experimental parameters with default values underlined.

5.2 Logical recovery of an entire index

The first set of experiments addresses the problem of recovering the entire B-tree. The standard recovery method is to create the B-tree from scratch starting from the base relation. For 20 MB main memory, the sorting algorithm generates 40 initial runs in the run-generation phase (each of them occupies about 40 MB of main memory on average due to replacement selection). The run generation phase requires 871 s in total (21.8 s per initial run), while the final merge phase (including creation of the B-tree) only needs 225 s, for a sum of 1'096 s. The standard recovery method thus requires 1'096 seconds, whereas logical recovery invokes only the final merge and thus requires only 225 seconds.

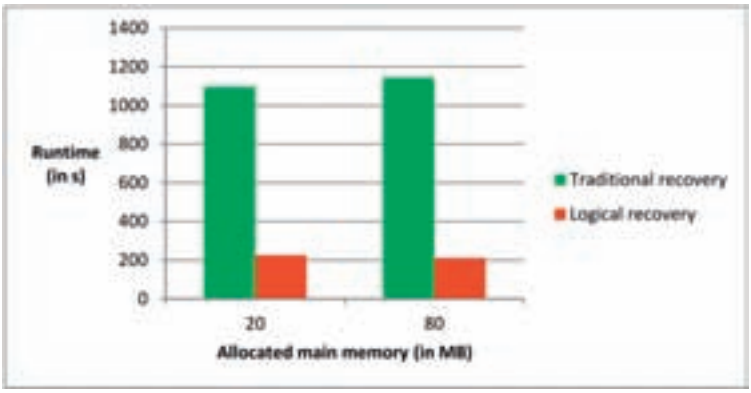


Figure 12. Recovery times for an entire secondary index.

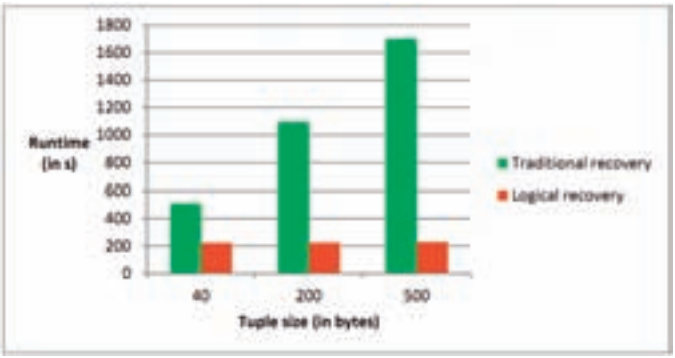


Figure 13. Effect of tuple size.

Figure 12 shows the performance of traditional recovery and logical recovery for memory sizes of 20 MB and 80 MB. Note that results are quite similar. In both memory settings, logical recovery achieves performance benefits of about a factor of five if the initial runs are still available. Even in case that some of the initial runs are not available anymore, it still would be better to reconstruct missing runs rather than using standard recovery (see Figure 2).

The reason for the performance improvements are that there is no need to read the tuples from the base relation again. In order to illustrate the influence of the tuple size, Figure 13 shows the performance of traditional recovery and logical recovery for different tuple sizes. As expected, the performance improvements increases in the tuple sizes, but performance improvements of a factor of 2 can still be achieved even for small tuples of size 40 bytes.

5.3 Logical single-page recovery

A second set of experiments considers the case that not the entire B-tree is lost but only a few of its leaf pages. Recall that traditional recovery does not take advantage of partial failures and must recreate the entire index from the base relation, which takes 1'096 seconds.

We assume that unavailable leaf pages correspond to an adjacent sequence of leaves within the B-tree. This assumption is justified for the following reason: adjacent leaf pages are often physically clustered on a single track of a disk. Thus, a track failure would cause the loss of adjacent leaf pages. We introduce parameter k to express the number of those leaves. If multiple key ranges (page sequences) are lost, logical recovery is invoked for each one in turn.

Recovery of a sequence of adjacent pages performs a range query on each of the final runs. It is therefore beneficial to have a B-tree maintained on the sorted runs. We examine two possibilities for indexing the sorted runs. The first is to create a separate B-tree on each of the runs, while the other is to create a partitioned B-tree over all the runs. The extra time to

create these indexes is similar to the time required to build the final B-tree from the runs (221 s in our experiments).

Figure 14 shows the performance of logical recovery for k = 1, 10, 100, 500 and 1'000 leaf pages. The figure shows a group of three bars (each the average of 10 experiments) for each setting of k. The first two bars within a group refer to the case where an index is built on each of the runs, while the third illustrates the performance in case of using a single partitioned B-tree for all runs. The first bar displays only the query time of the range queries performed on the different B-trees. As this does not include the time for opening files and initializing the B-tree, the second bar includes all these costs. It reveals that the cost of these preparatory actions is high and can become the dominant factor for small queries (k = 1). The third bar represents the recovery costs in case of using a partitioned B-tree including the costs for opening the corresponding file. Note that the cost of the preparatory actions is almost negligible for the partitioned B-tree. Another positive effect can be observed in case of partitioned B-trees for large values of k. The costs for processing queries are substantially smaller for a partitioned B-tree in comparison to using an index on each of the runs. The reason is that the clustering of pages on the disk is substantially better within a single index in comparison to the data being distributed among multiple indexes. Thus, the average cost for reading a page is lower for the partitioned B-tree.

Figure 15 compares these performance results with the ones obtained for traditional recovery. As discussed above, the standard recovery create an index from the base relation again; this takes 1'096 s in our experiments. If the runs are kept in a partitioned B-tree, logical recovery of a single page failure (k = 1) takes less than half a second. Note that the corresponding red bars are not visible anymore, thus the total time is indicated above the bars. In summary, the savings due to logical recovery exceed three orders of magnitude when compared to traditional recovery.

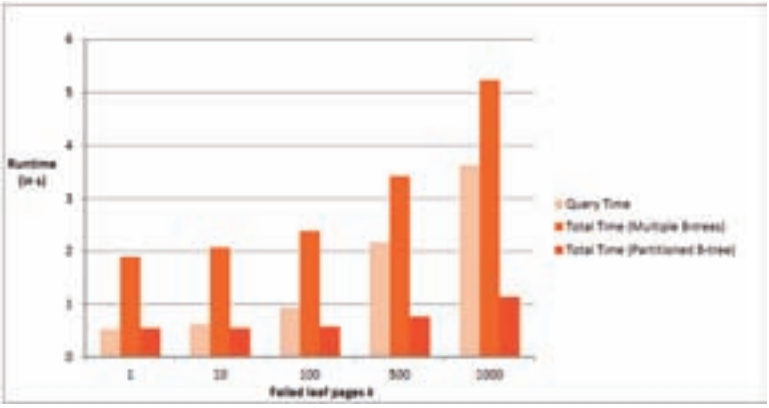


Figure 14. Logical recovery for page failures.

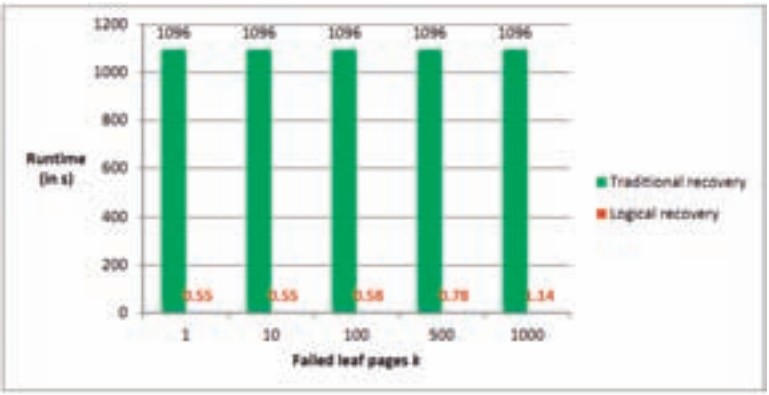


Figure 15. Traditional versus logical recovery.

6 Summary and conclusions

In summary, a number of commercial database systems optimize logging during index operations such as index creation. These techniques are known non-logged index creation, minimal logging, or allocation-only logging. These techniques require a "force" policy upon completion as well as log backups including the entire new index. The new technique requires neither flushing the new index to storage nor including the new index in the next log backup.

Moreover, the new techniques permit efficient recovery after small and large failures, e.g., due to locally worn-out flash storage. If an entire intermediate run is lost, its recovery merely repeats the step that created it. If only a few pages are lost, their recovery repeats only the minimal necessary index creation logic. For example, it repeats the logic to merge multiple intermediate runs into the final B-tree tightly limited to the key range of the lost pages.

An experimental performance evaluation demonstrates the efficiency of the new logical recovery techniques. Recovery of an individual page takes a fraction of a second; recovery of multiple contiguous pages proceeds with I/O bandwidth.

References

[BBD 01] Van den Bercken, J., Blohsfeld, B., Dittrich, J.-P., Krämer, J., Schäfer, T., Schneider, M., Seeger, B.: XXL – a library approach to supporting efficient implementations of advanced database queries. VLDB 2001: 39-48. Also see: http://code.google.com/p/xxl/.

[BU 77] Bayer, R., Unterauer, K.: Prefix B-trees. ACM TODS 2(1): 11-26 (1977).

[CAB 88] Copeland, G.P., Alexander, W., Boughter, E.E., Keller, T.W.: Data placement in Bubba. SIGMOD 1988: 99-108.

[G 78] Gray, J.: Notes on data base operating systems. In Bayer, R., Graham, R M., Seegmüller, G. (editors): Operating system – an advanced course. Lecture notes in computer science #60, Springer-Verlag Berlin Heidelberg New York (1978).

[G 03] Graefe, G.: Sorting and indexing with partitioned B-trees. CIDR 2003.

[G 12] Graefe, G.: A survey of B-tree logging and recovery techniques. ACM TODS 37(1): 1 (2012).

[GH 12] Graefe, G., Kuno, H.A.: Single-page failures. PVLDB 5(7): 646-655 (2012).

[GHI 12] Graefe, G., Halim, F., Idreos, S., Kuno, H.A., Manegold, S.: Concurrency control for adaptive indexing. PVLDB 5(7): 656-667 (2012).

[GK 10] Graefe, G., Kuno, H.A.: Self-selecting, self-tuning, incrementally optimized indexes. EDBT 2010: 371-381.

[GKK 12] Graefe, G., Kimura, H., Kuno, H.A.: Foster B-trees. ACM TODS 37(3): 17 (2012).

[GKS 12] Graefe, G, Kuno, H.A., Seeger, B.: Self-diagnosing and self-healing indexes. DBTest 2012: 8.

[GS 09] Graefe, G., Stonecipher, R.: Efficient verification of B-tree integrity. BTW 2009: 27-46.

[HR 83] Härder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM CSUR 15(4): 287-317 (1983).

[IKM 07] Idreos, S., Kersten, M.L., Manegold, S.: Database cracking. CIDR 2007: 68-78.

[IMK 11] Idreos, S., Manegold, S., Kuno, H.A., Graefe, G.: Merging what's cracked, cracking what's merged: adaptive indexing in main-memory column-stores. PVLDB 4(9): 585-597 (2011).

[LFV 12] Lamb, A., Fuller, M., Varadarajan, R., Tran, N., Vandiver, B., Doshi, L., Bear, C.: The Vertica analytic database: C-store 7 years later . PVLDB 5(12): 1790-1801 (2012).

[MHL 92] Mohan, C., Haderle, D.J., Lindsay, B.G., Pirahesh, H., Schwarz, P.M.: ARIES: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM TODS 17(1): 94-162 (1992).

[W 91] Weikum, G.: Principles and realization strategies of multilevel transaction management. ACM TODS 16(1): 132-180 (1991).

Privacy-Aware Multidimensional Indexing

Alexander Grebhahn[1], Martin Schäler[2], Veit Köppen[2], Gunter Saake[2]

[1] Brandenburg University of Applied Sciences
P.O. Box 2132, 14737 Brandenburg, Germany
grebhahn@fh-brandenburg.de

[2] Department of Technical and Business Information Systems
Otto-von-Guericke University Magdeburg
P.O. Box 4120, 39016 Magdeburg, Germany
{schaeler, vkoeppen, saake}@iti.cs.uni-magdeburg.de

Abstract: Deleting data from a database system in a forensic secure environment and in a high performant way is a complex challenge. Due to redundant copies and additional information stored about data items, it is not appropriate to delete only data items themselves. Additional challenges arise when using multidimensional index structures. This is because information of data items are used to index the space. As initial result, we present different deletion levels, to overcome this challenge. Based on this classification, we analyze how data can be reconstructed from the index and modify index structures to improve privacy of data items. Second, we benchmark our index structure modifications and quantify our modifications. Our results indicate that forensic secure deletion is possible with modification of multidimensional index structures having only a small impact on computational performance, in some cases.

1 Introduction

With an increasing usage of computer-aided systems, more sensitive information is stored in electronic formats. This may cause problems with respect to privacy. Laws and guidelines are created, to improve privacy, such as the Health Insurance Portability and Accountability Act (HIPAA) [Con96] in the USA, the Hard Drive Secure Information Removal and Destruction Guidelines [Roy03] in Canada, or the Bundesdatenschutzgesetz [Bun09] in Germany. According to these laws, private information has to be deleted or encrypted in such a way, that it cannot be reconstructed after deletion (forensic secure deletion). This is a complex challenge. Due to the fact that we need to remove *every* existing copy and *every* effect caused by the data item that we want to delete.

In our research project[1], we focus on private information, such as extracted features from fingerprints or micro traces. These features are multi- (less than 20 dimensions) or highdimensional data (more than 20 dimensions) containing, for instance, three-dimensional coordinates and classification attributes[2].

[1] https://omen.cs.uni-magdeburg.de/digi-dak-plus/
[2] Classification into multi- or highdimensional data according to [GG98].

In contrast to cloud computing models, where data and queries are outsourced to the cloud [HXRC11] and a privacy preserving storage (an encrypted storage) of data is necessary, we focus on systems, storing data in an unencrypted way. Additionally, to have a secure data life cycle, it is not only necessary to encrypt data, but also to delete data in a forensic secure manner [DW10].

Due to the huge amount of data, it is necessary to store data in systems that handle it in an appropriate way. As a result, we use multi- and highdimensional index structures to speed-up query response times. Within this paper, we make two major contributions:

1. We analyze, how we can reconstruct sensitive data from well-known index structures under certain assumptions (deletion strategies) and provide, to the best of our knowledge, the first empirical study regarding this topic.
2. We recommend improvements for forensic secure deletion for some of these indexes and evaluate their benefits and drawbacks exemplarily w.r.t. reduced reconstruction ability of data and run-time overhead. Based on our results, we define different levels of forensic secure deletion w.r.t. necessary reconstruction effort.

The remainder of the paper is organized as follows: In Section 2, we give an overview of related work and motivate our index structure selection. We also present background on functionality of analyzed index structures. In Section 3, we present four different strategies for deleting data from a database and their respective implementations for our index selection. In Section 4, we analyze these index structures w.r.t. stored information and how data can be reconstructed. Additionally, we present ideas how to modify index structures to minimize possibilities to reconstruct data. In Section 5, we evaluate index structures and index structure modifications w.r.t. performance and precision for an approximative index structure. Additionally, we show, how much information about data can be reconstructed from an index after deleting data. We draw a conclusion and present future work in Section 6.

2 Background

In this section, we briefly summarize the state of the art in database forensics, introduce terms, and provide necessary background on our selected index structures.

2.1 Related Work on Forensics in Databases.

To delete a data item from a database system forensically secure, it is necessary to consider more than the tuple stored in the table space [SML07]. This is because additional information about data items are stored by a database system. Beside the database (here, we mean the files containing the tuples), there are two other storage components communicating with the database system and an additional storage component for reconstructing the database when a media or system error occurs [BHG87]. A storage component communicating with the database system is the data dictionary. This component stores, information is stored e.g. histograms on data distribution or table schemas information. Third, the database log is stored by the database system. In this log, all information needed to recreate a consistent state of the database is stored. Stahlberg et al. give an

overview on challenges that have to be considered, in case of forensic secure deletion in database systems [SML07]. In detail, they cover information stored in database, in indexes, and in database logs exemplarily for InnoDB, a MySQL storage engine. As a representative of indexes, they cover a forensic secure deletion of data items from a B-Tree [SML07].

Within other work, see for example [Lit07, Fow08, FHMW10, Gre12], database systems, like Oracle, SQL Server, MySQL (using InnoDB as storage engine), PostgreSQL, and HSQLDB, are examined according to recomputation of data items from information left in the database files.

Besides deleting data items from a system in a forensic secure way, encrypting data is another solution for privacy preserving data management in highdimensional spaces. This is a commonly used technique for supporting privacy for outsourced data management as used within the cloud [HXRC11]. Many solutions are presented within the last years, see for example [KS07, HMCK12, WCKM09]. However, encryption is out of the scope of this paper.

2.1.1 Challenges in Definition of Forensic Secure Deletion.

To define, under which circumstances a data item is forensically secure deleted, we use the following intuitive definition for total forensic secure deletion:

A data item is deleted total forensic secure*, if absolutely no conclusions on exact or approximate values of any of the attributes of the data item can be drawn by using information stored in the system.*

However, due to non-trivial interdependencies (e.g., materialized aggregates) and non-obvious remains (e.g., in swap files or backups) using this definition is problematic as it is hardly reachable in practice. We further argue that, depending on data sensitivity, total forensic secure deletion is not always necessary. In the same sense, current laws state that the hurdles to access deleted data have be in an adequate relation to the value of the data (e.g., § 20(3) [Bun09]). Consequently, we define different levels of forensic secure deletion in Section 3. Before, we present an analysis of different deletion strategies.

2.2 Index structure selection

We give a brief overview of selected index structures in the following. For further information, see for example [GG98, Sam05]. In this paper, we: (1) show differences in data reconstruction derived from information stored in the index, (2) address different classes of index structures to generalize our results and address comprehensiveness, and (3) select well-known index structures.

For classification of index structures, we refer to existing classifications (see for example [GG98] and [WSB98]). They classify in data versus space organizing and exact versus approximative indexes. Consequently, we consider at least one data and one space organizing index as well as one exact and one approximative index. In detail, we focus on the R-Tree [Gut84] and respective extensions (e.g., [BKSS90, SRF87]), the VA-File [WB97], and the Prototype Based Approach (PBA) [CGFN08]. Note, due to the overlap of the classification attributes, we only use three index structures. We summarize our index selection in Table 1.

Table 1: Index structure selection.

Name	Data/Space Org.	Exact/Approx.	Remarks
R-Tree	Data	Exact	Tree-based
VA-File	Space	Exact	Improved sequential scan
PBA	Space	Approximative	Hash based.

2.2.1 R-Tree and its variants

One of the most popular multidimensional index structures is the R-tree. This index structure is presented by Guttman [Gut84]. Many improvements like R^+-Tree [SRF87], R^*-tree [BKSS90], X-Tree [BKK96], and SS-Tree [WJ96] are based on the ideas of the R-Tree. This basic idea is to partition the data space by the use of minimal bounding rectangles (MBR).

MBRs are organized in a hierarchical way as shown in Figure 1. The root MBR, namely R1 in Figure 1, includes the minimal space needed to include all child MBRs (R2 and R3). These MBRs again include the space of their child MBRs. This organization holds up to the leaf MBRs. Within these leafs (marked in gray in Figure 1), data items are stored.

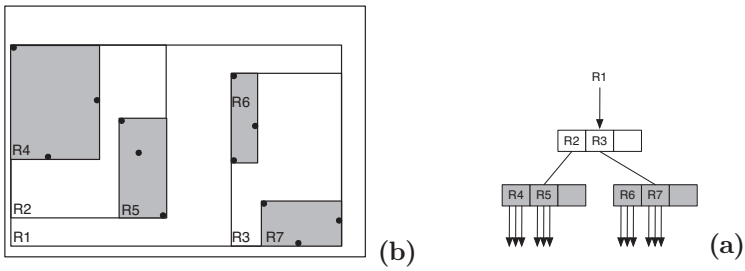


Figure 1: (a) Partitioning of a two-dimensional space by an R-Tree. (b) Hierarchical structure of the MBRs.

For each MBR of an R-Tree, two points are stored. These two points are both ends of one diagonal of the MBR. Additionally to these two points, each MBR contains pointers to all child MBRs for inner nodes and pointers to all data items indexed by the MBR for leaf nodes. According to the idea of Guttman, the storage size of a node should be correlated with the page size of the underlying system. For modeling such a correlation, the maximum and minimum number of data items per page can be defined by parameter m. The maximum number of data items is implicitly given by M, with $M = 2 \cdot m$. So, for every node of an R-Tree two points are stored together with pointers to child nodes for inner MBR or pointers to data items for leaf nodes.

2.2.2 VA-File

The VA-File is a space organizing index proposed by Weber and Blott [WB97]. The main idea of this index structure is to store a small representation of original data that fits more likely into the main memory. This representation addresses rectangular cells in form

of bit vectors used to filter and thus, to reduce the amount of points that are retrieved from hard disk.

As shown in Figure 2 (a), there are four distinct regions in each dimension when choosing a vector length of two for each dimension. This leads to 2^b regions, in the case of choosing a vector length of b. In general, in a d-dimensional space, the space is divided into 2^{bd} hyper rectangles. The formal allocation of a point to a region, by Weber and Blott [WB97], is stated in Equation 1, where $r_{i,j}$ defines the partition p_i is located in dimension j. Furthermore, m_i states the lower bound of the ith-partition. According to this definition, the bounds of partitions are defined by values of points.

$$m_i[r_{i,j}] \leq p_{i,j} < m_i[r_{i,j+1}] \tag{1}$$

For being adaptive to different data distributions, the regions width depends on the data distribution. We present an example in Figure 2 (a). Here, the width of region *10* of dimension x is larger than the width of region *00*. As a result, of this unequal distinction of the space, within the index structure a map has to be stored to describe the mapping between the original space and the resulting approximation vector. So, the VA-File stores an approximation vector for each data item and the mapping from the original space to the approximated one.

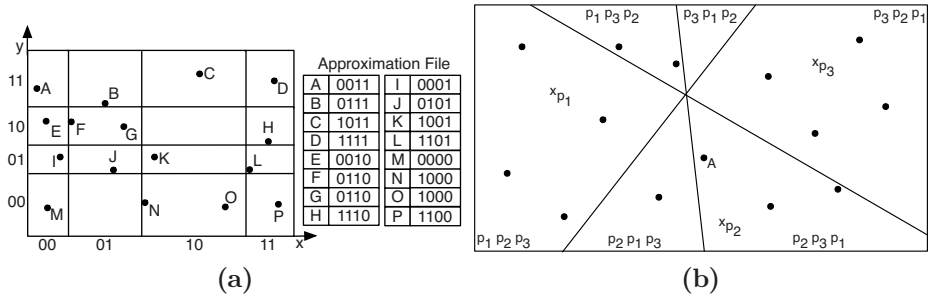


Figure 2: (a) Structure of a VA-File. (b) PBA using three prototypes.

2.2.3 Prototype Based Approach

An example for an approximation-based index structure is the Prototype Based Approach (PBA) presented by Chavez et al. as Ordering Permutations [CGFN08]. The basic idea is to use some points from a dataset to index the whole dataset. According to Chavez et al., we call these points *prototypes*. In general, this technique divides the whole space in convex regions based on distances to the prototypes. In Figure 2 (b), we show a space partition with three prototypes p_1,p_2, and p_3. When inserting a point, distances to all prototypes are computed. Next, the prototypes are ordered ascending according to their distances. Finally, this ordering is used as hash value or key of the point. For example, the value of point A in Figure 2 (b) is (p_2, p_3, p_1). As a result, the hash value of each data item and the coordinates of all prototypes are stored in the index.

3 Deletion Strategies and Respective Deletion Levels

In this section, we present and discuss different strategies for deleting data items from an index structure. Furthermore, based on the remaining possibilities to reconstruct data, we use these strategies to describe different levels of forensic secure deletion. Additional, to presented deletion levels, it is possible, to rebuild an index after every deletion.

3.1 Level 0: Delete Bits

The first strategy is using a delete bit for identifying, whether a data item is deleted or not. In other words, when data have to be deleted, not the whole item is modified, but a bit within the header of the data item for marking it as deleted. This strategy has disadvantages w.r.t. privacy of deleted data items. With the help of simple tools and knowledge of the structure of the database, it is possible to identify deleted data items and to reconstruct them completely [Lit07, Fow08]. A prototype of such a forensic tool for PostgreSQL is given in [Gre12].

In summary, using a delete bit or similar technique to mask deleted data items allows to easily reconstruct data items in total (with all attributes) with basic knowledge of the way how items (tuples) are stored. Note, this, is no deletion at all, and therefore we call it Level 0 (cf. Table 2), indicating that a data item is *not* forensically secure deleted. However, modifying the delete bit is very time efficient and requires no reorganization of index structures. Hence, this deletion strategy is very time efficient.

Integration in index structures. Integrating this strategy into known index structures is rather simple. Here, no reorganization of parts of indexes (e.g., MBRs within an R-Tree) is necessary, if an item is deleted. An additional challenge arises in frequently changing tables through the constantly increasing size of the index.

3.2 Level 1: Overwriting without reorganization

A next level strategy is deleting (and overwriting) the whole data item without modifying the index structure. Although the data item is removed and overwritten, it is possible to reconstruct (parts of) the deleted data item. This is due to remaining information (e.g., structure of the index) that can be used for an attempt to reconstruct the data. To sum up, reconstruction of data (a) is more laborious and (b) is not possible in all cases, and (c) requires more detailed knowledge on the way index structures store their data. Consequently, this deletion strategy forms forensic secure deletion Level 1, and thus, the first level that offers *basic* forensic deletion capabilities.

In contrast to Level 0 deletion strategy, we hypothesize that the amount of information that can be reconstructed, depends on the definition of the index structure and therefore, it is purpose of our analysis and experiments in the next sections. Using this deletion strategy, there are index-specific cases that still allow either (1) total reconstruction of a data-item, (2) reconstruction of some *attributes* with exact values, or (3) we can state upper and lower bounds of attribute values. First and more detailed considerations to determine probability of single

index-specific cases and an analysis for respective causes are also part of the next sections.

Integration in index structures. By using this strategy it is not necessary to consider, for example, underfull MBRs in an R-Tree. Furthermore, we do not have to recompute the partitioning of the VA-File or the PBA. However, since we have to overwrite possibly large datasets, the effort for this deletion strategy is higher than for Level 0.

3.3 Level 2: Overwriting with reorganization

To address remaining threat of reconstructable data, we introduce another level that offers *advanced* forensic secure deletion capabilities (Level 2). The goal of this level: it is *practically* impossible to reconstruct data items deleted from an index structure. The main reasons why it is possible to reconstruct data using Level 1 are remaining, index-specific traces due to missing reorganization of the index. Thus, the additional effort for reaching this level is reorganization of indexes as we describe in the next section.

Integration in index structures. The integration of this deletion strategy, within a system supporting multi-user, may cause some performance problems, because of concurrent operations on the index. Furthermore, the index reorganization strategy depends on its conceptual design.

3.4 Hypothetic Level ∞: Total forensic secure deletion in data-intensive systems

Although there are no (known) remaining traces in an index, there may be information that can be used to reconstruct data items, such as dependencies in the data (e.g., materialization of aggregates), or hidden copies (e.g., swap files, backups) that need to be considered too. To define the scope and limitation of database forensics, we therefore define a *hypothetic* deletion level that allows *no* reconstruction at all. This level is not defined for indexes only, but it is valid for data-intensive systems.

The basic idea is to have two systems. The first one is the original system (S_{org}) and the second one (S_{shadow}) a (bit-wise) copy[3] of S_{org}, which we denote by: $S_{org} \cong S_{shadow}$. Until the (initial) insertion of data item (d) that we want to delete, both systems behave the same way. That means, they store the same data, swap data from main memory to disk etc. The difference between both systems is that S_{shadow} ignores the insertion of d. After insertion of d, these systems perform again the same read and write operations. Under these circumstances, we consider a function f as total forensic secure deletion w.r.t. d iff $f(S_{org}) \cong S_{shadow}$ holds.

Since we are aware that building these shadow systems is probably practical impossible, we want to create systems that are approximations (S'_{shadow}) of S_{shadow}, where we know simplifications and thus, limit possible effects, we do not consider (e.g., swap files). This shall help to identify non trivial remains of datasets, which are part of future work.

In Table 2, we subsume our four levels of forensic secure deletion. Between this four levels, other level can be defined.

[3]This includes bit-wise copy of all HDDs, main memory, caches, and even CPU registers.

Table 2: Levels of forensic secure deletion.

Level	Technique	Application recommendation	Reconstruction effort	Runtime overhead
0	Delete Bit	No private data.	Low	Low
1	Overwrite	Private data	Medium	Medium
2	Reorganization	Sensitive data	High	High
∞	Shadow image	-	∞	∞

4 Problems with respect to privacy and improvements

In this section, we present privacy problems by information stored in index structures. Additionally, we show modifications for improving privacy of stored information. Within these modifications, we try to reach similar results w.r.t. privacy of *advanced* forensic secure deletion (Level 2) of data items even without reorganization of indexes (as in Level 1).

4.1 R-Tree

With the help of the structure of an R-Tree, conclusions on data distribution as well as single values of data items can be drawn. Firstly, the root node can be used to exclude non covered data space. This is because an R-Tree is a data partitioning method. As a result, it only indexes the space needed. Secondly, because of maximum number of points per MBR, within dense covered regions, more MBRs exist as in sparse covered regions. Thirdly, two points are stored within each MBR for defining size and location. Because an MBR covers the minimal space needed, exact values of points are used to define borders and edges of an MBR. Due to the fact that all data items are stored in leaf nodes, only these nodes have to be analyzed to reconstruct data item specific values.

For improving privacy of single data items within an R-Tree, it is possible to bounce the borders of the MBR away from the location of points dedicated to that node. This increases the overlapping of MBRs within an R-Tree. However, no exact values of single data items are used for defining the corners of the MBR. In bouncing the borders of the MBRs, one has to be aware of R-Tree properties. For example, a parent MBR covers at minimum the whole space covered by its child MBRs. As a result, when bouncing the border of a leaf MBR, all borders of all parent MBRs, sharing a border with it, have to be updated as well.

4.2 VA-File

By considering the information stored in a VA-File, three types of conclusions about the dataset or specific data items can be drawn. Firstly, because of adaptable division of the space, conclusions on the data distribution can be drawn, because all buckets have approximately the same amount of data items dedicated to them. As a result, if a bucket is larger

than a different one, it covers dense populated space. Secondly, the exact values of 2^b-1 data items are stored within the VA-File (see Equation 1). Having a d-dimensional space, this leads to $(2^b-1)^d$ exact values. Thirdly, by using the bit-vector, the approximate location of a data item can be reconstructed. Although, the approximate location of a data item may not lead to privacy problems, in some cases it is possible that the width of a bucket may not only reveal the approximate location but the exact one. For example, if $\frac{1}{2^b}$ data items have the same value in one dimension, the width of the bucket, the points are dedicated to, equals one.

For improving privacy of data items, we modify the VA-File in two different ways. Firstly, we adapt the VA-File in such a way that all buckets have the same width. This leads to some performance penalties when performing queries over none uniformly distributed data. Nevertheless, this partitioning of the data space has advantages for privacy of data items. Because, no information about the data distribution or single data items can be reconstructed from the information stored in the VA-File modification. Additionally, no data item specific information, like exact values of data items within some dimensions are stored. It may happen that a data item is located at the border of a bucket, but the location of the border is not defined by the data item and so not dependent from the data. Additionally, we extend the VA-File in a way that the length of single bit strings per dimension depends on the value domain of this dimension. In other words, we shorten the used bit string for each dimension until the number of regions per dimension is smaller than the used value domain of this dimension.

4.3 Prototype Based Approach

There are some possibilities to improve precision, performance, and privacy of the PBA. Choosing prototypes from the dataset is good for adapting the partitioning of the space to the distribution of the dataset. However, choosing prototypes in a random way leads to some negative effects because some data items may have a greater expressiveness for the distribution of the dataset than others. Additionally, choosing points from the dataset as prototypes leads to privacy problems if the prototype is used after deleting the data. Or it leads to performance problems, because permutations of all points have to be recomputed after a prototype is deleted.

Some modifications at prototype selection and respective position of the prototypes can be implemented, w.r.t. privacy. On the one hand, it is not necessary to choose points from the dataset, but points representing the distribution of the dataset in an optimal way. On the other hand, location of prototypes can be optimized w.r.t. different criteria. For example, it is possible, to choose prototypes, that all regions have the same size. This leads to some performance penalties in performing queries on non-uniform distributed datasets. However, this optimization criterion is good w.r.t. privacy, because the division of the space does not depend on data distribution.

5 Evaluation

In this section, we present a first empirical study on possibilities of reconstructing data items from information stored in multidimensional index structures. Later, we measure the performance penalties introduced by our modifications to improve privacy. In our evaluation, we use the framework QuEval[4]. With this framework, it is possible to measure performance of multidimensional index structures for specific use cases. The idea of this framework and the general structure is proposed in [GBS+12]. Due to the extensibility of the framework, it is possible to extend index structures and the framework with evaluation experiments.

5.1 Datasets

We perform all tests with three different real datasets. In Table 3, we give an overview of dataset properties. The first dataset has only a small number of dimensions (16). As a result, it is multidimensional according to [GG98]. In contrast, the remaining datasets are highdimensional, having 43 and 50 dimensions. With these two datasets, we evaluate the performance impact of the data space population, both having approximate same number of dimensions but different number of points and different value domains.

In detail, the first dataset is a freely available dataset based on extracted hand-writing features [AA96]. In the second dataset (*fingerprint features*), the spectral texture features of latent fingerprints are stored [KFV11]. The last dataset (*particle identification*) is again freely available. Within this dataset, 50 particle identification numbers are stored for 130,064 events [RYZ+05].

Table 3: Properties of datasets used for the evaluation.

domain	#dimension	#points	value domain
Hand-writing features	16	10,992	[0..100]
Fingerprint features	43	411,961	[0..255]
Particle identification	50	130,064	[0..1023]

5.2 Index structure evaluation

In Table 4, we give an overview of our evaluated index structures, modifications we implemented for improving privacy, and evaluations we performed. Due to space limitations, we are not able to present all possible evaluations. For a first insights into the problematic of privacy in multi and highdimensional index structures, we performed an evaluation of the reconstruction rate, modified index structures and evaluate performance and precision of the modifications. In detail, in Section 5.2.1, we evaluate the reconstruction rate (RR) of

[4] http://wwwiti.cs.uni-magdeburg.de/iti_db/research/iJudge/index_en.php

deleted data items stored in an R-Tree when performing deletion Level 1. In Section 5.2.2, we present performance of two different VA-File modifications for improving privacy. Finally, in Section 5.2.3, we present precision differences when we do not choose points from the dataset as prototypes, but points in their neighborhood.

Table 4: Evaluated index structures with performed evaluations.

index structure	modifications	target
R-Tree	Original	RR
VA-File	Original according to [WB97], commensurate regions, Adaptive bit vector length	Performance
PBA	Original, blur location of the prototypes	Precision

5.2.1 Reconstruction rate of deleted data items in an R-Tree

We define the reconstruction rate (RR) of a deleted data item as given in Equation 2. $rDim$ is defined as the number of those dimensions where the exact value of data items can be reconstructed and $allDim$ is the number of all dimensions. In this evaluation, we use some ideas presented in [Lin12].

$$RR = \frac{rDim * 100\%}{allDim} \tag{2}$$

Within our evaluation, we use the Level 1 deletion strategy. In detail, we delete the whole data item, but we do not modify the borders of MBRs. In Figure 3, we show the average RR as well as the maximum RR (dotted) of 10,000 deleted data items for all three datasets used in our evaluation. Additionally, we evaluate RR with different numbers of minimal and maximal points per MBR. Here, we vary m (minimal number of points per MBR) from 2 to 12. Note, maximum number is always two times the minimal number.

In all parts of Figure 3, our results indicate that the average RR of data items decreases with increasing minimum and maximum number of points per MBR. This is, because more points within an MBR decreases the possibility that one point defines a large number of borders. In addition, by comparing the average RR lines of Figure 3 (a), (b) and (c), we draw the conclusion, that the average RR decreases for a given minimum and maximum number of points per MBR with increasing number of dimensions. Additionally, the differences of average RR from (a) to (b) is larger than from (b) to (c). This is, due to the fact the differences of dimensionality between (a) and (b) is greater than between (b) and (c). As a result, we state the hypothesis, that dimensionality has an impact on the RR of data items. Beside this, for every test case, at least one data item can be reconstructed to probability of at least 60%. In detail, within the 50 dimensional space, for every case tested, at minimum one data item can be completely reconstructed.

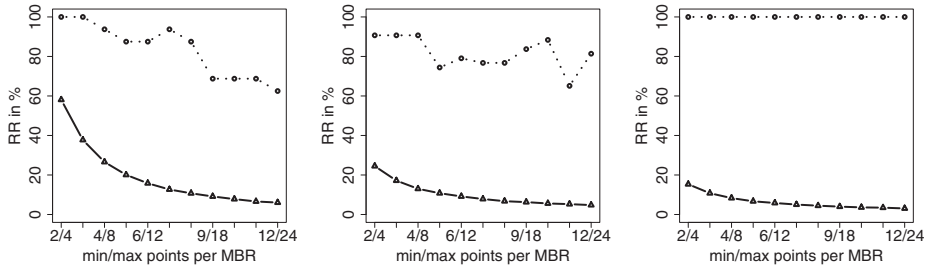


Figure 3: Reconstruction rate (RR) of deleted data items from the information stored in an R-Tree for a 16 (a), 43 (b) and 50 (c) dimensional dataset. The average RR is marked with a solid line and the maximum RR with a dotted line.

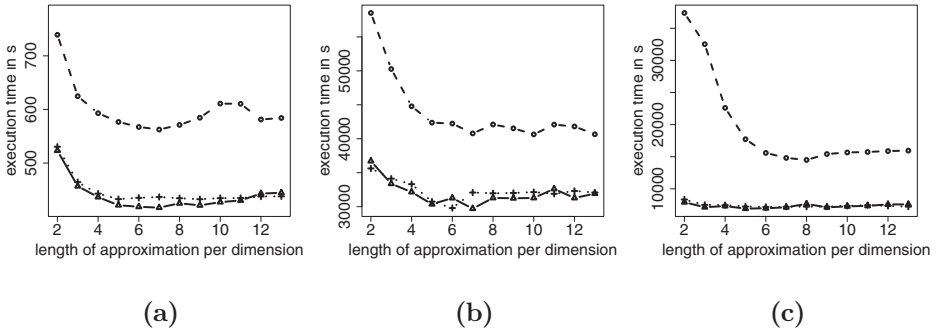
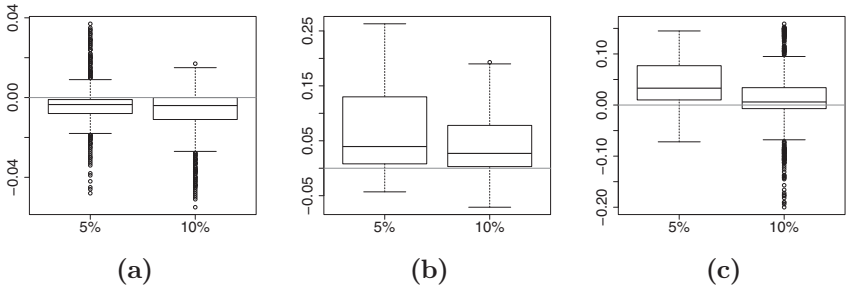


Figure 4: Performance differences of the three VA-File variants for 16 (a), 43 (b) and 50 (c) dimensional dataset. The performance of the original VA-File is marked with a solid line, the performance of the VA-File variant which is not adaptable to the distribution with a dashed line and performance of the variant with an adaptive bit-vector length with a dotted line.

5.2.2 VA-File

In Figure 4, we show the performance of the three different VA-File variants; namely the original VA-File as presented by Weber and Blott [WB97] (solid), the VA-File variant which is not adaptable to the distribution of the dataset (dashed line), and the variant with an adaptive bit-vector length within different dimensions (dotted line). In our experiments, we vary the length of the bit-vector in a range of 2 to 12.

Our results clearly show (cf. Figure 4) that the performance of the VA-File variant which is not adaptable to the distribution of the space is worse than both other VA-File variants. In our experiments, data items are stored on disk and without being adaptive to the data distribution, more points have to be accessed from it.



(a) **(b)** **(c)**

Figure 5: Precision differences between the original Prototypes Based Approach as presented in [CGFN08] to the index structure with blurred prototypes. Differences for the 16 (a), 43 (b) and 50 (c) dimensional dataset with a blur of 5% and 10%.

5.2.3 Prototype Based Approach

Choosing prototypes in a random way leads to poor results regarding to performance and precision. Additionally, in the case of deleting points from the dataset chosen as prototypes, it is necessary to choose new prototypes and to recompute the permutation of all indexed data items. Because of the permutation and the concrete values of all items from the dataset, the location of the prototypes can be recomputed. To overcome this, we modify locations of prototypes with a vector having normal distributed components between zero and given strength (in our examples 5% and 10% of the value domain). For not being affected from one parameter configuration, we performed about 1700 tests with two different blur factors and different parameter configurations for number of prototypes and considered points.

In Figure 5, we show the average difference of precision of the PBA for all three datasets. Within Figure 5, the precision differences for a blurring of 5% and 10% is given, for all three dataset. Blurring the location of prototypes has either a positive or a negative impact on the precision of the index structure depending on the dataset and index parameters. In detail, for our experiments with the 16 dimensional dataset, blurring has mainly a negative impact on precision. However, the average difference of precision is smaller than 0.02% and so, almost negligible. For our other two experiments, blurring the precision has a positive impact, but again the average difference is smaller than 0.1%. All in all, the impact on precision, when choosing random points near to dataset points instead of dataset points as prototypes is almost negligible. As a result, it is not necessary to choose points from the dataset as prototypes.

6 Conclusion & Future Work

To summarize, within this paper, we present four different deletion strategies that can be used within a database system. Additionally, we define forensic secure deletion of information from a database system and present a classification of different secure deletion levels. Furthermore, we examine three different multidimensional index structures (namely R-Tree, VA-File and PBA) in regard how information is stored and how this information

can be used for reconstruction of data items, where we performed an exemplarily evaluation for the R-Tree. Later, we exemplarily extend index structures to be privacy aware. Furthermore, we evaluate our index structure modifications with respect to performance and precision. Within this evaluation, we identify, that improving privacy may also have a positive but small effect on query performance such as improving precision of the PBA.

In future work, we want to show and evaluate a method for improving privacy of data items stored in R-Tree variants. Additionally, we want to evaluate different kinds of prototype selection methods for PBA with respect to privacy. Furthermore, we will extend index structures implementations of our QuEval framework with the presented deletion strategies and evaluate performance of index structures.

7 Acknowledgments

The work in this paper has been partially funded by the German Federal Ministry of Education and Science (BMBF) through the Research Program under Contract No. FKZ:13N10816 and FKZ:13N10817. Additionally, we want to thank Ina Lindauer for her implementation of the analysis for reconstruction of data items within an R-Tree.

References

[AA96] F. Alimoglu and E. Alpaydin. Methods of Combining Multiple Classifiers Based on Different Representations for Pen-based Handwriting Recognition. In *TAINN*, pages 637–640. IEEE, 1996.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[BKK96] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *VLDB*, pages 28–39, 1996.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331. ACM, 1990.

[Bun09] Bundesministerium der Justiz. Bundesdatenschutzgesetz, August 2009.

[CGFN08] E. C. Gonzalez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *TPAMI*, 30(9):1647–1658, 2008.

[Con96] United States Congress. Health Insurance Portability and Accountability Act (HIPAA). http://www.hhs.gov/ocr/privacy/, 1996.

[DW10] S. M. Diesburg and A. A. Wang. A survey of confidential data storage and deletion methods. *ACM Comput. Surv.*, 43(1):2:1–2:37, 2010.

[FHMW10] P. Frühwirt, M. Huber, M. Mulazzani, and E. R. Weippl. InnoDB Database Forensics. In *AINA*, pages 1028–1036. IEEE Computer Society, 2010.

[Fow08] K. Fowler. *SQL Server Forensic Analysis*. Addison-Wesley Professional, 2008.

[GBS+12] A. Grebhahn, D. Broneske, M. Schäler, R. Schröter, V. Köppen, and G. Saake. Challenges in finding an appropriate multi-dimensional index structure with respect to specific use cases. In *GvD*, pages 77–82. CEUR-WS, 2012.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[Gre12] A. Grebhahn. Forensisch sicheres Löschen in relationalen Datenbankmanagementsystemen. Master thesis, University of Magdeburg, 2012. In German.

[Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57. ACM, 1984.

[HMCK12] B. Hore, S. Mehrotra, M. Canim, and M. Kantarcioglu. Secure multidimensional range queries over outsourced data. *The VLDB Journal*, 21(3):333–358, 2012.

[HXRC11] H. Hu, J. Xu, C. Ren, and B. Choi. Processing private queries over untrusted data cloud through privacy homomorphism. In *ICDE*, pages 601–612, 2011.

[KFV11] T. Kiertscher, R. Fischer, and C. Vielhauer. Latent fingerprint detection using a spectral texture feature. In *MMSec*, pages 27–32. ACM, 2011.

[KS07] A. Khoshgozaran and C. Shahabi. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In *SSTD*, pages 239–257. Springer-Verlag, 2007.

[Lin12] I. Lindauer. Analyse des Rekonstruktionspotentials von multidimensionalen Indexstrukturen zum sicheren Löschen. Master thesis, University of Applied Sciences Brandenburg, Germany, 2012. In German.

[Lit07] D. Litchfield. Oracle forensics part 2: Locating dropped objects. *NGSSoftware Insight Security Research (NISR) Publication, Next Generation Security Software*, 2007.

[Roy03] Royal Canadian Mounted Police. G2-003. Hard Drive Secure Information Removal and Destruction Guidelines, October 2003.

[RYZ+05] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor. Boosted Decision Trees as an Alternative to Artificial Neural Networks for Particle Identification. *NIMPA*, 543(2-3):577–584, 2005.

[Sam05] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

[SML07] P. Stahlberg, G. Miklau, and B. N. Levine. Threats to privacy in the forensic analysis of database systems. In *SIGMOD*, pages 91–102, 2007.

[SRF87] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *VLDB*, pages 507–518, 1987.

[WB97] R. Weber and S. Blott. An Approximation-Based Data Structure for Similarity Search. Technical Report 24, Zurich, Switzerland, 1997.

[WCKM09] W. K. Wong, D. Wai-Lok Cheung, B. Kao, and N. Mamoulis. Secure kNN computation on encrypted databases. In *SIGMOD*, pages 139–152, 2009.

[WJ96] D. A. White and R. Jain. Similarity Indexing with the SS-tree. In *ICDE*, pages 516–523. IEEE Computer Society, 1996.

[WSB98] R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB*, pages 194–205. Morgan Kaufmann Publishers Inc., 1998.

Pack Indexing for Time-Constrained In-Memory Query Processing

Tobias Jaekel, Hannes Voigt, Thomas Kissinger, Wolfgang Lehner

Department of Computer Science
Dresden University of Technology
D-01062 Dresden
{firstname.lastname}@tu-dresden.de

Abstract: Main memory databases management systems are used more often and in a wide spread of application scenarios. To take significant advantage of the main memory read performance, most techniques known from traditional disk-centric database systems have to be adapted and re-designed. In the field of indexing, many main-memory-optimized index structures have been proposed. Most of these works aim at primary indexing. Secondary indexes are rarely considered in the context of main memory databases. Either query performance is sufficiently good without secondary indexing or main memory is a resource too scarce to invest in huge secondary indexes. A more subtle trade between benefit and costs of secondary indexing has not been considered so far.

In this paper we present Pack Indexing, a secondary indexing technique for main memory databases that allows a precise trade-off between the benefit in query execution time gained with a secondary index and main memory invested for that index. Compared to traditional indexing, Pack Indexing achieves this by varying the granularity of indexing. We discuss the Pack Indexing concept in detail and describe how the concept can be implemented. To demonstrate the usefulness and the effectiveness of our approach, we present several experiments with different datasets.

1 Introduction

The field of database management systems (DBMS) is changing; main memory database management systems (MMDBMS) become more important. Today, main memory is as large and cost-efficient as never before. To avoid expensive I/O operations on data, MMDBMS store entire databases in-memory. The advantage of MMDBS compared to disk-based DBMS is the significantly higher read performance and that the data can be processed much more efficiently. The trend towards MMDBS enables novel application scenarios such as live business intelligence [Pla09].

To fully exploit the advantage of main memory technology, most of the well-researched technologies of disk-based DBMS have to be rethought and redesigned for MMDBMS. For instance, storing tuples row-wise is less suitable for main memory since data compression is less efficient, sequential main memory accesses are more cache-efficient and the memory's bandwidth capabilities are exploited. In consequent, most MMDBMS are based on column-

wise data storage [CK85, ADHS01]. Other techniques such as access methods, page layouts, etc. are affected equally. In the area of indexing, a lot of indexing structures were invented to use the capabilities of main memory more efficiently [KCS+10, RR00, KSHL12].

Auxiliary indexing, however, is an often neglected topic, since table scans and primary indexes access are sufficient fast on main memory in the most cases. Nevertheless, indexing becomes important when queries have to be processed within a time constraint and a scan would exceed this constraint. Today, data grows rapidly and the need of secondary indexes will become more likely to keep up with the promises of main memory databases. Indexes consume main memory, which is in contrast to disk scarce resource. Usually, relations consist of columns that contain a lot of duplicate values. These duplicates can be compressed efficiently, but the index' size stays high caused by insufficient duplicate handling. Consequently, secondary indexes cannot be created as excessively in main memory as on disk. Indexing in MMDBMS requires careful decisions about which data to index.

Secondary indexes are created because of insufficient execution times. This implies concrete objectives about the targeted execution times. Traditional indexing techniques only allow coarse-grained indexing decisions: either a column is fully indexed or not indexed at all. Ideally, the database administrator is able to tailor the index for his requirements and aimed execution times. Especially, with query time constraints the database administrator wants to spend only as much memory as necessary on indexing to meet the time constraint.

In this paper, we present Pack Indexing for main memory databases management systems. Pack Indexing is able to (1) align the execution time to a time constraint and (2) limit the memory consumption for auxiliary indexing significantly. Further, Pack Indexing enables trading query execution time for memory consumption in a fine-grained way. Aligned to a given time constraint, our novel concept indexes packs of records instead of individual records to consume as little memory as possible. To meet the time constraint for every queried value the pack size can be configured for each value individually. We also discuss the implementation of the Pack Indexing concept and present a detailed evaluation of the concepts. Pack Indexing is applicable for row-oriented as well as column-oriented storages.

The rest of the paper is structured as follows: In the following section we present the concept of Pack Indexing. Section 3 gives an overview of our implementation. The Performance Rating component is described in Section 4. In Section 5 the Pack Configurator is discussed. How the Pack Index is used during runtime is described in Section 6. An Evaluation of the Pack Index system is given in Section 7. In Section 8 we discuss the related work and in Section 9 this paper is concluded.

2 Pack Indexing

A time-constrained system has an inherent characteristic: It does its work within a given time limit. In a database system the execution time highly depends on the quantity of read tuples. Reducing the number of read tuples means decreasing the execution time. An index scan reduces the number of read tuples to the expense of memory; A column scan reads all

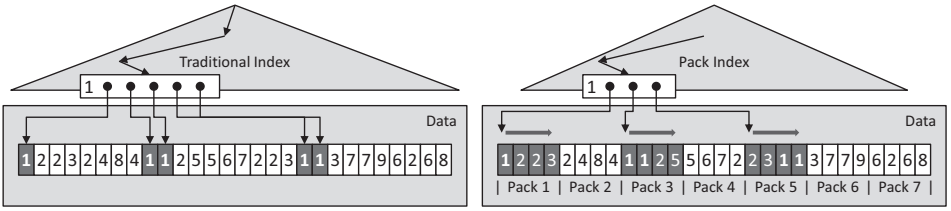


Figure 1: Traditional Index compared to Pack Index.

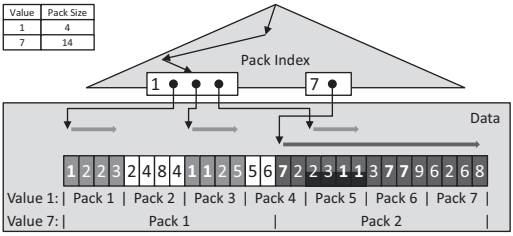


Figure 2: Pack Index with individualized pack sizes.

tuples without memory expenses. Granularity-wise, an index scan is the most fine-granular access and a column scan the most coarse-grained access.

Pack Indexing allows controlling the granularity of indexing. It provides an access method that can be configured for any granularity between a traditional index scan and a column scan. A more coarse-grained index consumes less memory than a more fine-grained index. Thus, Pack Indexing provides a mean to the trade between execution time and memory consumption.

Pack Indexing builds on two ideas: (1) Indexing packs of records instead of individual records and (2) Configuring the size of the packs for each value in the index individually depending on the value's frequency.

Figure 1 shows the first idea in contrast to traditional indexing. The example shows a column of 28 tuples and 9 distinct values. Consider Value 1, which occurs five times in the dataset. The traditional index indexes the tuples directly. This requires five index entries and allows reading the five qualifying tuples directly. In contrast, the Pack Index logically combines the tuples to packs and indexes the packs. Since Value 1 occurs multiple times in Pack 3 and Pack 5, the Pack Index needs merely three index entries. Reading the qualifying tuples, though, requires reading three complete packs with a total of 12 tuples. Assuming the targeted time constraint is at 50% of a complete scan, the Pack Index achieves the goal with two entries less than the traditional index. Where saving space is more important than the last pinch of execution time benefits, indexing packs of values allows saving index space in a controlled and directed way.

Figure 2 illustrates the second idea of Pack Indexing: Individual pack sizes for each value.

As we have seen, a pack size of four is sufficient to push queries on Value 1 below the targeted time constraint. For queries on Value 7, occurring only three times in the dataset, a pack size of 14 is a far better choice. The read costs of 14 tuples meet the assumed 50% time constraint with a single index entry. A pack size of 4 would push queries on Value 7 below the time constraint at a price of two index entries. Pack sizes individualized depending on a value's frequency allow further space savings by tailoring a Pack Index to the value distribution of a dataset.

The value distribution of a dataset is what primarily determines the concept's potential benefit. The benefit of the Pack Index approach is the amount of memory it can save for a given execution time constraint compared to a traditional index. Depending on the data distribution, the amount varies from dataset to dataset. Two kinds of distributions are to consider. The logical value distribution represents the frequencies with which the individual instances of a value domain occur in the dataset. The physical distribution denotes the physical clustering of the values on the storage. Logical value distribution and physical value distribution are orthogonal properties of a dataset and both influence the benefit of a Pack Index.

Generally, Pack Indexing aims to exploit disparities in the data distribution. Physically as well as logically uniformly distributed data has no disparities to exploit and constitutes the worst case with the lowest expectable benefit. Physical distribution has a stronger influence on the benefit than logical data distribution. If a value is uniformly distributed over the physical representation of a dataset the probability that matching records occur in a pack is equal for all packs independently from the pack size. In contrast, non-uniformly distributed values will cluster in a fraction of the packs and reduce the number of required index entries. The logical data distribution is what pack sizes are tailored to. Hence, logically uniformly distributed data, i.e., where each value occurs with the same frequency, will result in equal pack sizes for all values. The actual benefit a non-uniformly distributed value range allows, depends on the physical distribution and the index data structure the Pack Index builds on.

Summarizing, at best data is logically as well as physically non-uniformly distributed. Real-world data distribution is generally time-dependent cause by seasons, day-and-night cycle, and trends in development and style. Shopping for instance: Without doubt, you buy different things in the morning than in the evening, different things in summer than in winter, and different things in five years from now. The today increasingly common append-only databases reflect these variations in data distribution logically as well as physically. Thus, the best case for Pack Indexing is a likely case.

3 System Overview

The Pack Index system consists of three components: (1) A rating of the database systems memory read performance, (2) the Pack Configuration, and (3) the Pack Index Access Path. Figure 3 shows the three components and their basic interaction.

The Performance Rating component provides measures of the read performance of the database system. While the database system is setup – either initially or after hardware

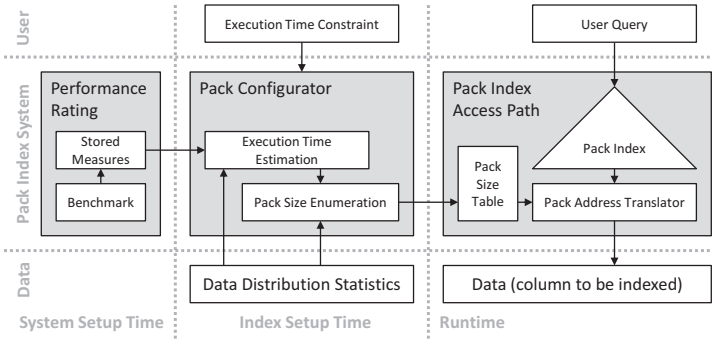


Figure 3: Pack Index System

reconfiguration – the Performance Rating component benchmarks the read performance. It stores the results of this measurement in the database catalog for the Pack Configurator to use. We detail the Performance Rating component in Section 4.

The Pack Configurator determines the individual pack sizes, if the user creates a new Pack Index. With the index create statement, the user provides the execution time constraint the index should achieve. Based on data distribution statistics taken from the system catalog, the configurator determines an individual pack size for each of distinct values in the column the index is created on. The determined pack sizes are stored in the Pack Size Table of the newly created index. We present the Pack Configurator in detail in Section 5.

The Pack Index Access Path is used during query execution. It consists of the Pack Size Table, the Pack Index, and the Pack Address Translator. The Pack Index holds the actual index entries. For a given query, it provides the pack numbers of the packs that contain matching tuples. The Pack Size Table lists the individual pack size for each value. The Pack Address Translator uses this information to translate pack numbers into memory addresses during queries and memory addresses into pack numbers during DML operations. We elaborate on the Pack Index in Section 6.

4 Performance Rating

Execution time of selection queries mainly depends on (1) how much data has to be read to find and fetch qualifying tuples and (2) how fast data can be read from memory. The Performance Rating component provides reliable measures for the memory read performance. During system setup, the component measure the read performance for a variance of read patterns and stores them in the system catalog. The Pack Configurator later uses the measures while determining individual pack sizes.

When using the Pack Index Access Path, the fetch operator has to scan a certain number of packs of a certain size. Both, number of packs and pack size influence the read performance.

Reading a small number of very large packs results in mainly sequentially memory access, while reading a large number of small packs leads to a predominantly random memory access.

The Performance Rating component captures this variance in memory read performance. It conducts a number of measurements while varying the read patter. For each read pattern the component measures read performance for a specific pack size and a specific probability that a pack has to be scanned. This results in a two-dimensional matrix of measures. Reading packs of 4 MB with a probability of 25% took 3.798 ms per pack, for instance.

To keep the number of measures in a reasonable range the Performance Rating component varies read patters logarithmically. This keeps the costs of conducting the measurements and storing the result low, while sufficiently capturing the variance in the system's read performance. To get a measure for a read pattern that has not been measured, the Pack Configurator interpolate between the two closest patterns measured. Note that the measures do not have to be stored in memory since they are only used by the Pack Configurator during index setup time. Further, the performance rating is system-dependent and is shared among all indexes in a system.

5 The Pack Configurator

The Pack Configurator determines the individual pack sizes during index creation. While creating a Pack Index the user gives the index an execution time constraint. For each value in the Pack Index, the configurator aims to find the largest pack size so that selection queries on that index fulfill the constraint. Therefore, the configurator enumerates difference pack sizes and estimates the resulting query execution time. The configuration closest to the constraint is selected.

5.1 Configuration Enumeration

Naïvely, the configurator investigates for each value every possible pack size. This guarantees to consider all possible configurations at cubic time complexity. Assuming N bytes of data and M distinct values, the configuration would require $O(M \cdot N/l)$ iterations, where l is the size of a tuple. For instance, given column store with an integer column of 1 GB containing 10,000 distinct values and an investigation speed of 10,000 iterations per second, the enumeration would last about 100 months.

We employ a more sophisticated enumeration strategy. It extends the naïve enumeration in two ways. First, we group values of the same frequency. The runtime estimation depends not on the value itself but on its frequency. Consequently, values with the same frequency result in the same pack size and we need to run the enumeration only once for each frequency. Second, we use binary search to find the largest fitting pack size for a given frequency. This reduces the number of considered pack sizes significantly to $O(M \cdot \log_2 N/l)$. In the example, the improved enumeration takes about 30 seconds to find the same configuration.

Algorithm 1 Pack Size Enumeration

```
1:  procedure ENUMERATION(f, t_c, n)
2:      ▷ f: value frequency   ▷ t_c: time constraint   ▷ n: number of tuples
3:      p_l ← 0, p_h ← log_2(n)        ▷ exponent of lowest and highest pack size
4:      c ← true
5:      t_l ← ESTIMATE(2^{p_l}, f, n)                  ▷ traditional index scan time
6:      t_h ← ESTIMATE(2^{p_h}, f, n)                  ▷ column scan time
7:      while c ∧ p_l < p_h do
8:          p ← p_l + (p_h − p_l)/2                    ▷ exponent of pack size to try
9:          s ← round(2^p)                             ▷ pack size to try
10:         t_e ← ESTIMATE(s, f, n)                    ▷ estimate execution time (section 5.2)
11:         if t_c − t_e < 0 then
12:             p_h ← p                                ▷ try smaller pack size
13:             if t_e − t_l < ϵ then                  ▷ close to lowest pack size
14:                 c ← false                          ▷ stop search
15:                 s ← 2^{p_l}                        ▷ use lowest pack size
16:         else if t_c − t_e > ϵ then
17:             p_l ← p                                ▷ try larger pack size
18:             if t_h − t_e < ϵ then                  ▷ close to highest pack size
19:                 c ← false                          ▷ stop search
20:                 s ← 2^{p_h}                        ▷ use highest pack size
21:         else                                       ▷ close to constraint
22:             c ← false                              ▷ stop search
23:     return s
```

Algorithm 1 shows the enumeration. For a given value frequency, a execution time constraint, and a total number of tuples the algorithm searches the largest pack size (in number of tuples) within the execution time constraint. Following the principle of binary search, the enumeration iteratively tests pack size configuration (line 7–22). If the estimated execution time of tested configuration exceed the given constraint, the algorithm reduces the pack size (line 11f). If the estimated execution time falls significantly below the constraint, it increases the pack size (line 16f). If the estimated time is in a small range ϵ below the constraint, the algorithm stops the search (line 21f).

In contrast to normal binary search, the enumeration increase and decrease the pack size exponentially (line 8f) instead of linearly. This follows the observation that large packs are exponentially more likely to contain a given value than small partitions, which renders large packs exponentially more unlikely to allow execution time below a given constraint. Note that exponent p is not an integer, so that the enumeration still can find any pack size. To avoid a large number of iterations towards the end of search without any reasonable change in the outcome, the algorithm stops if it gets ϵ-close to either the constraint, the execution time of the smallest possible pack size, or the execution time of the largest possible pack size. Thereby, ϵ controls the accuracy of the search.

5.2 Execution Time Estimation

To evaluate pack size configurations, the configurator estimates the execution time of a selection query given a certain pack size, the frequency of the queried value, and the total number of tuples. Essentially, query execution with the Pack Index Access Path consists of reading the Pack Index and fetching the matching tuples.

The index access consists of finding the matching entries and reading these entries. Most of the common index structures find entries in constant time C. Whereas reading the entries generally depends on the number of entries to read, specifically the number of matching packs. Hence, index access time is

$$t_{IX}(s, f) = C + r(size(n_p(s, f)))$$

where $n_p(s, f)$ is the number of matching packs, $size$ gives the size of the corresponding index entries in bytes, and r is the performance rating for a single chunk of the given size. Note that $size$ depends on the specific index structure chosen for the Pack Index.

The fetch operation has to read all matching packs. Accordingly, the fetch time is

$$t_F(s, f) = n_p(s, f) \cdot r(n_p(s, f) \cdot l, f)$$

where $n_p(s, f)$, again, is the number of matching packs, l is the tuple length in bytes, and r yields the performance rating for packs of the given size and at the given read probability.

To estimate the number of matching packs, we adopt the page fetch rate of row-level Bernoulli sampling given in [HK04]. In our case, with a total of n tuples, the expected number of matching packs for a pack size of s tuples and value frequency f is

$$n_p(s, f) = \frac{n}{s} \cdot (1 - (1 - f)^s) \quad .$$

This assumes physically uniformly distributed data. Clustered data would result in a smaller number of packs that would have to be read effectively.

6 The Pack Index Access Path

The Pack Index Access Path is used to answer queries at system runtime. It consists of the Pack Size Table, the Pack Index, and the Pack Address Translator.

The Pack Size Table contains the individual pack sizes of the index values. Formally, it is a function $\mathcal{S} : D \rightarrow \mathbb{N}$, which maps the indexed value domain to a natural number representing the pack size in number of tuples per pack. It can be implemented as a separated lookup structure or can be integrated with the Pack Index.

The Pack Index holds the actual index entries. Formally, the Pack Index is a function $\mathcal{I} : D \rightarrow \mathbb{N}^*$, which maps the indexed value domain to a list of pack numbers. Additionally, we assume the Pack Index returns matching pack numbers in ascending order. Most of the

common indexing data structures such as B-Trees, Hashing, or Bitmap Indexes can be used to implement the Pack Index.

The Pack Address Translator calculates memory addresses from pack numbers and vice versa. It realizes two functions: (1) Forward translation turns a pack number i and a pack size s into a memory address a with $a = \mathcal{F}(i, s) = s \cdot i \cdot l$. (2) Backward translation turns a memory address a and a pack size s into a pack number i with $i = \mathcal{B}(a, s) = a/(s \cdot l)$. Here, l is the tuple byte length. Query processing uses forward translation, while DML operations that have to update the Pack Index require backward translation. Addresses are relative to the beginning of the column.

The Pack Index Access Path answers selection queries with point predicates, range predicates, or predicate disjunctions. Naturally, the data structure used for the Pack Index has to support these kinds of predicates, too. Given a predicate, the Pack Index Access Path retrieves matching pack numbers from the Pack Index, translates the pack numbers to memory addresses and scans the corresponding packs to fetch matching tuples.

Point predicates are the simple case since they query a single value and all pack numbers returned by the Pack Index belong to the same pack size. Algorithm 2 shows the basic procedure. The Pack Index Access Path simply iterates the pack numbers returned by the Pack Index (line 3), forward translates the pack numbers, and scans the corresponding tuples (line 4). Because the Pack Index returns pack numbers in ascending order, the procedure scans the memory as sequentially as possible, hopping only where it has to omit not matching packs.

Algorithm 2 Answering Point Predicates

```
1: procedure POINTPREDICATEQUERY(v)                ▷ v: queried value
2:     s ← S(v)
3:     for all i ∈ I(v) do                          ▷ iterate pack numbers
4:         SCAN(F(i, s), s · l)                     ▷ scan pack (l: tuple byte length)
```

Processing range predicates and predicate disjunctions is more complex since matching packs may differ in their sizes. Regarding their treatment by the Pack Index Access Path, range predicates and predicate disjunctions can be generalized to set predicates. A value matches a set predicate if it is contained in a set of values given by the predicate. Naïvely, the Pack Index Access Path would process each value in the set predicate separately and unite the results of the scans. Main drawback of this approach is that tuples in overlapping packs would be read multiple times. For instance, consider the situation shown in Figure 2. If Value 1 and Value 7 are queried in the same set predicate, the naïve approach would scan the tuples in Pack 5 twice.

To avoid repeated scanning of tuples, the Pack Index Access Path has to align pack addresses and remove duplicates. Algorithm 3 shows the basic procedure. Given a set predicate, the Pack Index Access Path determines the common pack size, essentially, the greatest common divisor of the individual pack sizes of the queried values (line 2). Then it aligns for all queried values the pack numbers returned by the Pack Index to the common pack size (line 3–9). This can be done easily by forward translate the pack number with its corresponding pack size and backward translate the resulting address with the common

pack size (line 6). Unfortunately, the backwards translation returns only the address of the first pack with the common pack size. All following packs have to be added manually by the algorithm (line 8). This is done by adding as much packs as common pack size would fit into the original pack size subtracted 1, because the first pack number was created by the backwards translation. After aligning the pack numbers, the Pack Index Access Path merges the individual pack number lists into a single list. Thereby, it keeps the order of the pack numbers and removes duplicates. All of which can be done efficiently with the standard merge sort procedure (line 10). Finally, the Pack Index Access Path iterates the merged list of aligned pack numbers (line 11), forward translates the pack numbers with the common pack size, and scans the corresponding tuples (line 12).

Algorithm 3 Answering Set Predicates

```
1:  procedure SETPREDICATEQUERY(V)                          ▷ V: set of queried values
2:      s_c ← gcd(S(V))                  ▷ greates common divider of all matching pack sizes
3:      for all v ∈ V do                                     ▷ iterate queried values
4:          s ← S(v), I_v ← I(v)
5:          for all i ∈ I_v do
6:              i ← B(F(i, s), s_c)                          ▷ align first pack number
7:              I_v ← I_v ∪ i
8:              for all c ∈ [1, s/s_c − 1] do                ▷ add following pack numbers
9:                  I_v ← I_v ∪ (i + c)
10:     I ← merge all I_v                                    ▷ merge sort pack numbers
11:     for all i ∈ I do                                     ▷ iterate pack numbers
12:         SCAN(F(i, s_c), s_c · l)                         ▷ scan pack
```

7 Evaluation

We conducted a series of experiments to evaluate the Pack Index concept. Specifically, we examined the approaches impact on query runtime and memory consumption. Further, we investigated the influence of query selectivity and data distribution. All experiments were performed on a system equipped with an Intel i5 3450 3.1GHz CPU, 16GB of DDR3 1600MHz main memory, and Windows 7 x64 Professional as operating system.

For the experiments, we prototypically implemented all Pack Indexing components for an in-memory column store. The column store uses directory compression, so that all columns store fixed-length directory keys. Read-optimized, a column is stored in one big block in memory. The directory keys are integers running from 0 to n for n distinct values. Our prototype uses bitmap indexes for the Pack Index. All bitmaps can be efficiently looked up through a pointer array that uses the directory keys as array indexes. The Pack Size Table is implemented analogously.

We used two datasets. The first dataset is the well-known Netflix dataset of movie ratings. We indexed the movie column. It contains 17,770 distinct movies for more than 100 million movie ratings which total at about 400 MB. The physical order of values remained

unchanged. We used the first dataset to investigate query runtime and index memory consumption. The second dataset is synthetic and consists of a single column containing 100 million integer values. We generated multiple versions of the dataset with varying number of distinct values and value distributions. In all versions of the second dataset, the physical order of values is random with uniform distribution. The second dataset was used to investigate the influence of query selectivity.

For all experiments, we used the same performance ratings which were measured once before the experiments. For each experiment, we created a Pack Index for a given time constraint using the Pack Index Configurator and the data distribution statistics were determined from each dataset after the load.

7.1 Query Execution Times

First, we investigated the resulting query execution times when using a Pack Index configured for varying execution time constraints. We varied the execution time constraint from 12.5% to 100% of the runtime of a column scan. For each movie m in the Netflix dataset, we measured the runtime of the point query `SELECT movie FROM Netflix WHERE movie = 'm'`.

Figure 4 shows the results. When using the Pack Index Access Path, the query runtime varies from movie to movie mainly because of the individual pack sizes configured for each movie. The distribution of query runtimes is shown in the figure by the box plot. The figure also shows the corresponding execution time constraint. As can be seen, the Pack Index Access Path did not exceed the time constraint. The only exception is the constraint of 12.5% where the three most frequent movies are too frequent to be fetched within the given constraint. Consequently, no index technique could push a query on these movies below the constraint. For comparison, the figure also shows the runtimes of a full column scan.

Further, the results show a wide spread in the execution times when using the Pack Index Access Path. Reason of this spread is the physical distribution of the values. The Pack Index Configurator assumes physically uniformly distributed data. The movies in Netflix dataset, though, are non-uniformly distributed; the ratings have chronological order and a movie is most frequently rated when it hits the Netflix' DVD racks. In consequence of the non-uniform physical distribution of the Netflix data, the Pack Index Configurator overestimates the number of packs. A smaller index without violating the execution time constraint would be possible. However, this requires detailed knowledge about the physical order of the tuples during index creation time and reduces the index' robustness against changes in the physical tuple order.

7.2 Memory Consumption

Second, we examined the memory consumption of the Pack Index configured for varying execution time constraints. Again, we varied the execution time constraint from 12.5% to

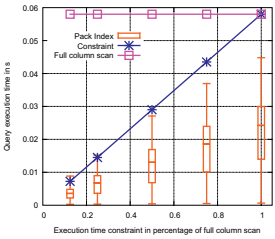


Figure 4: Query runtime of Pack Index

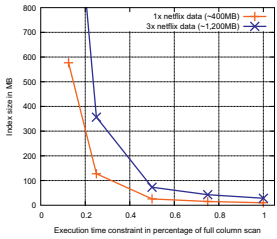


Figure 5: Memory consumption of Pack Index

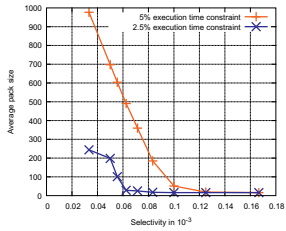


Figure 6: Selectivity of the dataset

100% of the runtime of a column scan. We used (1) the Netflix dataset in its normal size and (2) the Netflix dataset tripled in size. The tripled Netflix dataset is essentially three concatenated copies of the original dataset. For each dataset and execution time constraint, we measured the size of the Pack Index.

Figure 5 shows the results. As can be clearly seen, the size of the Pack Index decreases with an increasing execution time constraint. This clearly illustrates the trade-off between execution time and memory usage we wanted to achieve. The longer queries runtimes you are willing to accept, the less memory you have to spend on indexing. Nevertheless, the trade-off was not linear. To push all queries below 50% runtime of a full column scan, we only had to spend 26 MB which is roughly 6% of the data size. A constraint 25% already required an index as large as 29% of the data size.

The trade-off observations hold independently from the size dataset. As the figure also shows, a larger dataset exhibits the same runtime–space trade-off. However, a larger dataset requires a larger index. Provided that the logical and physical data distribution remains the same, the index grows linearly with the data.

7.3 Selectivity of the Index

The wide spread in execution times in the first experiment is caused by different query selectivities. To analyze the selectivity's impact on the time constraint, we created a synthetic dataset with 100 million tuples and varied the selectivity. With physically and logically randomly distributed data, the number of distinct values specifies the selectivity of an index. We investigated different selectivities in the range from $0.033 \cdot 10^{-3}$ (30,000 distinct values) up to $0.166 \cdot 10^{-3}$ (6,000 distinct values). These test were made for two different time constraints, 2.5% and 5% of the column scan time. For each selectivity we measured the average pack size.

Figure 6 shows the results. As you can see, with a higher selectivity, which means the column holds more distinct values for the same total number of tuples, a bigger average pack size is used. In contrast, a lower selectivity leads to smaller packs, which consume more memory. Furthermore, Figure 6 illustrates the selectivity's and time constraint's impact on the pack size. For an average pack size of 200 tuples per pack, the test with

the 2.5% time constraint requires a selectivity of $0.05 \cdot 10^{-3}$ where a system with 5% only need a selectivity of $0.08 \cdot 10^{-3}$. In real-world scenarios point queries have different selectivities, depending on the queried value. To maximize the memory savings and to exploit the differences in selectivity, each value has to be aligned independently.

The results also show, for highly selective datasets the time constraint can be decreased without fully indexing the data. For instance, the tests for a selectivity of $0.055 \cdot 10^{-3}$. The 5% time constraint had an average pack size of 600 tuples per pack whereas the other test had a average pack size of 100. For both examples no fully index was required to meet the given time constraint. However, the time constraint that can be achieved when using the Pack Index strongly depends on the selectivity of the queries.

8 Related Work

The Pack Index is a novel indexing concept and independent of the index structure. However, Pack Indexing is designed for main memory databases. Indexing structures suitable for characteristics of main memory were developed in the past decade [CK96, KCS+10, KSHL12]. Theses structures are highly optimized and speed up the MMDBMS by avoiding cache misses. Combining Pack Indexing with such optimized structures leads to a win-win situation. On the one hand the index structures benefit from using Pack Indexing while consuming less memory. On the other hand the capabilities of memory saving will be exploited, if Pack Indexing is used in combination with a highly optimized index structure.

Pack Indexing aims at time-constrained query execution times, a recurring goal in database research over the last decades. In [HOT89] a statistical method for aggregation queries is presented. A similar approach is used in [OGDH95] for non-aggregation queries. Both methods are based on statistical estimation and once the time for processing the query is exceeded, the evaluation process will be interrupted. In [RSQ+08] the authors present a system that constantly scans the partitions to answer queries. Thus, the system is able to answer a query within a constant time. By constantly scanning partitions, the query can never be answered faster than the scan execution time. Pack Indexing focuses on scenarios in which a faster execution time is required. At the same time the indexes have to consume as little memory as possible to meet the constraint.

Time-constrained systems are closely related to real-time systems, here real time database systems (RTDBS). An overview of methods and techniques is given [KGm95]. RTDBS deal with hard time constraints and are used in special domains such as monitoring. In monitoring scenarios a high update or insert ratio is very important. In contrast, the Pack Index is oriented towards read-intensive scenarios.

Partial Indexing is a useful approach to save memory and index maintenance costs, even in main memory databases [Sto89, SS95]. A partial index only indexes a subset of tuples which helps to keep the index small, but if an unindexed value is queried the execution time increases to the scan time. Thus, to apply partial indexes detailed information about the workload of the database system is required. Our approach also saves memory, but not by leaving tuples unindexed. Additionally the Pack Index is independent of any query

workload knowledge. Pack Indexing and partial indexing are complementary approaches.

A pack contains a subset of tuples of the table like a horizontal partition does. Partitioning is well-known and all major database vendors introduced database design advisors, which recommend index, views and partitioning configurations [ACK+04, ZRL+04, DDD+04]. In modern cloud systems partitioning is also used for scalability [AEAAD10]. In contrast to partitioning, which is applied physically, packs do not split the data into physical fragments.

9 Conclusions

The amount of data processed and stored in main memory databases grows rapidly, but the techniques of disc-centric database systems cannot transferred without further ado. In this paper we proposed Pack Indexing, a technique that is suitable for row-oriented as well as column-oriented storages. Pack Indexing builds on two ideas: (1) Indexing packs of records instead of individual records and (2) Configuring the size of the packs for each value in the index individually depending on the value's frequency. These techniques allow us to control the granularity of an index to create a trade-off between execution time and memory consumption.

We explained an implementation of Pack Indexing consisting of three components. The Performance Rating component measures the system's read performance. These results in combination with the time constraint are used by the Pack Configurator to determine the pack size for each value individually. Both, the Performance Rating and Pack Configurator are used during setup time. At runtime, the Pack Index Access Path provides the actual benefit to the queries. Additionaly, we evaluated our prototypical implementation in several experiments. Our tests showed the benefit of Pack Indexing in memory consumption while the system did not exceed the time constraint.

References

[ACK+04] Sanjay Agrawal, Surajit Chaudhuri, Lubor Kollár, Arunprasad P. Marathe, Vivek R. Narasayya, and Manoj Syamala. Database Tuning Advisor for Microsoft SQL Server 2005. In *VLDB'04*, 2004.

[ADHS01] Anastassia Ailamaki, David J. DeWitt, Mark D. Hill, and Marios Skounakis. Weaving Relations for Cache Performance. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, 2001.

[AEAAD10] Divyakant Agrawal, Amr El Abbadi, Shyam Antony, and Sudipto Das. Data Management Challenges in Cloud Computing Infrastructures. In *Databases in Networked Information Systems*, volume 5999 of *Lecture Notes in Computer Science*, pages 1–10. Springer Berlin / Heidelberg, 2010.

[CK85] George P. Copeland and Setrag N. Khoshafian. A decomposition storage model. In *Proceedings of the 1985 ACM SIGMOD international conference on Management of data*, SIGMOD '85, 1985.

[CK96] Kong-Rim Choi and Kyung-Chang Kim. T*-tree: a main memory database index structure for real time applications. In *Real-Time Computing Systems and Applications, 1996. Proceedings., Third International Workshop on*, 1996.

[DDD+04] Benoît Dageville, Dinesh Das, Karl Dias, Khaled Yagoub, Mohamed Zaït, and Mohamed Ziauddin. Automatic SQL Tuning in Oracle 10g. In *VLDB'04*, 2004.

[HK04] Peter J. Haas and Christian König. A bi-level Bernoulli scheme for database sampling. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, 2004.

[HOT89] Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeo K. Taneja. Processing aggregate relational queries with hard time constraints. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, SIGMOD '89, 1989.

[KCS+10] Changkyu Kim, Jatin Chhugani, Nadathur Satish, Eric Sedlar, Anthony D. Nguyen, Tim Kaldewey, Victor W. Lee, Scott A. Brandt, and Pradeep Dubey. FAST: fast architecture sensitive tree search on modern CPUs and GPUs. In *SIGMOD'10*, 2010.

[KGm95] Ben Kao and Hector Garcia-molina. An Overview of Real-Time Database Systems. In *Advances in Real-Time Systems*, pages 463–486. Springer-Verlag, 1995.

[KSHL12] Thomas Kissinger, Benjamin Schlegel, Dirk Habich, and Wolfgang Lehner. KISS-Tree: smart latch-free in-memory indexing on modern architectures. In *Proceedings of the Eighth International Workshop on Data Management on New Hardware*, DaMoN '12, 2012.

[OGDH95] G. Ozsoyoglu, S. Guruswamy, Kaizheng Du, and Wen-Chi Hou. Time-constrained query processing in CASE-DB. *Knowledge and Data Engineering, IEEE Transactions on*, 7(6):865 –884, dec 1995.

[Pla09] Hasso Plattner. A common database approach for OLTP and OLAP using an in-memory column database. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, 2009.

[RR00] Jun Rao and Kenneth A. Ross. Making B^+-Trees Cache Conscious in Main Memory. In *SIGMOD'00*, 2000.

[RSQ+08] V. Raman, G. Swart, Lin Qiao, F. Reiss, V. Dialani, D. Kossmann, I. Narang, and R. Sidle. Constant-Time Query Processing. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, 2008.

[SS95] P. Seshadri and A. Swami. Generalized partial indexes. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, pages 420 –427, mar 1995.

[Sto89] M. Stonebraker. The case for partial indexes. *SIGMOD Rec.*, 18(4):4–11, December 1989.

[ZRL+04] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy M. Lohman, Adam Storm, Christian Garcia-Arellano, and Scott Fadden. DB2 Design Advisor: Integrated Automatic Physical Database Design. In *VLDB'04*, 2004.

Duplicate Detection on GPUs

Benedikt Forchhammer[1], Thorsten Papenbrock[1], Thomas Stening[1], Sven Viehmeier[1], Uwe Draisbach[2], Felix Naumann[2]

Hasso Plattner Institute
14482 Potsdam, Germany
[1]firstname.lastname@student.hpi.uni-potsdam.de
[2]firstname.lastname@hpi.uni-potsdam.de

Abstract: With the ever increasing volume of data and the ability to integrate different data sources, data quality problems abound. Duplicate detection, as an integral part of data cleansing, is essential in modern information systems. We present a complete duplicate detection workflow that utilizes the capabilities of modern graphics processing units (GPUs) to increase the efficiency of finding duplicates in very large datasets. Our solution covers several well-known algorithms for pair selection, attribute-wise similarity comparison, record-wise similarity aggregation, and clustering. We redesigned these algorithms to run memory-efficiently and in parallel on the GPU. Our experiments demonstrate that the GPU-based workflow is able to outperform a CPU-based implementation on large, real-world datasets. For instance, the GPU-based algorithm deduplicates a dataset with 1.8m entities 10 times faster than a common CPU-based algorithm using comparably priced hardware.

1. Introduction

Duplicate detection (also known as *entity matching* or *record linkage*) is the task of identifying multiple representations of the same real-world entities [NH10]. It is an integral part of data cleansing and an important component of every ETL process. Duplicate detection is typically performed by applying similarity functions to pairs of entries in datasets: Some algorithm carefully selects promising pairs of records. If the values of two records are sufficiently similar, they are assumed to be duplicates. Due to the large number of comparisons and the ever-increasing size of many databases, duplicate detection is a problem that is hard to solve efficiently. However, in most approaches the comparisons of record pairs are independent from one another – the problem is highly parallelizable. In this paper, a selection of duplicate detection algorithms and similarity measures are described and adapted in the context of General Purpose Computation on Graphics Processing Units (GPGPUs).

General purpose GPU programming has gained much appreciation in the past few years. Unlike *Single Instruction, Single Data* (SISD) CPU architectures, *Single Instruction, Multiple Data* (SIMD) GPU computing allows the execution of one set of operations on large amounts of data in a massively parallel fashion. This parallelization can provide immense speedups in applications that focus on highly data-parallel problems.

Currently, there are only few frameworks for GPGPU development. For our prototype, we use the OpenCL 1.0 framework, as it allows development for both ATI and NVIDIA graphics cards. The framework allows the execution of so-called kernels, which are written in a variant of ISO C99. OpenCL kernels can be executed on different devices; usually the device is a graphics card, but other devices, in particular the CPU, are also possible if respective hardware drivers are available. Devices execute kernels as work items. A work item is a set of instructions that are executed on specific data by one thread. Further work items are grouped into work groups.

When developing applications for GPUs, memory management is a key factor: GPUs have four types of memory with different capacities and different access speeds: *Global memory* is slow but has the highest capacity; *local memory* is faster but has a far smaller capacity; *private memory* is only usable by one operating unit; and *constant memory* is the fastest but not writable by the graphics card. An additional difficulty lies in the fact that it is not possible to allocate memory dynamically on the GPU. We address these memory challenges and opportunities in the next sections. Concerning the execution units, the graphics card executes a number of threads (usually 32) in so-called *warps*. All threads within a warp execute the same instructions on different data. If one thread of a warp takes a longer execution time, all the others wait. Moreover, conditions in the program flow are serialized; each thread waits until the complete warp finishes an `if`-statement, before starting with an `else`-statement. After an `else`-statement the threads are synchronized as well. Hence, we avoid divergent branching as far as possible.

Our main contribution is a complete duplicate detection workflow that utilizes the resources of the GPU as much as possible. First, we describe how each algorithm can be parallelized to utilize a very high amount of GPU cores. Second, we propose algorithm specific data-partitioning structures and memory access techniques to organize data in the NUMA architecture of GPUs. Finally, we present experiments that evaluate the performance of the presented workflow based on different CPU and GPU hardware. For comparison reasons, we optimized the algorithmic parameters for high precision and recall values (not for speed) and used real world data sets as input data.

In the following Sec. 2, we highlight related work for the areas of duplicate detection and GPGPU programming. Section 3 introduces the individual components of the duplicate detection workflow. Section 4 describes our adaptations for two popular pair-selection methods for the GPU environment. In Sec. 5 we adapt algorithms for popular similarity measures, as well as for the aggregation of different result lists and clustering. Section 6 evaluates the components of the workflow on various hardware platforms. The last section summarizes our results and discusses future work.

2. Related Work

Duplicate detection has been researched extensively over the past decades. Recent surveys [EIV07,NH10] explain various techniques for duplicate detection and methods for improving effectiveness and efficiency. Common approaches to improve the efficiency of duplicate detection are blocking and windowing methods, such as the Sorted Neigh-

borhood method [HS95], which reduce the number of comparisons. Another approach to reducing execution time is parallelization, i.e., splitting the problem into smaller parts and distributing them onto multiple computing resources. Our approach combines both the Sorted Neighborhood method and parallelization.

Parallelization has been proven to be effective by various authors. One of the first approaches to parallelizing duplicate detection is the Febrl system [CCH10], which is implemented in Python and parallelized via the well-known Message Passing Interface (MPI) standard. Kim and Lee presented a match/merge algorithm for cluster computing based on distributed Matlab [KL07]. Kirsten et al. developed a parallel entity matching strategy for a service-based infrastructure [KKH10]. They evaluate both the Cartesian product as well as a blocking approach, and demonstrate that parallelization can be used to reduce execution time significantly. Kolb et al. explored *map-reduce* to bring duplicate detection onto a cloud infrastructure [KTR11]. They focus on parallelizing the Sorted Neighborhood method and their experiments show nearly linear speedup for up to 4 and 8 cores. While these papers present effective approaches to the problem of parallelizing duplicate detection, they all require multiple CPUs or PC clusters for parallelization. This limits the level of parallelization that can be achieved, e.g., Kirsten et al. use up to 4 nodes and 16 CPUs for evaluation. Compared to what is possible with GPUs, the respective level of parallelization is low.

Katz and Kider worked on parallelizing transitive closure, i.e., the step of transforming a list of duplicate pairs into duplicate clusters [KK08]. In contrast to other papers on this topic [AJ88,To91] which only use CPUs for parallelization, Katz and Kider's approach utilizes graphics cards. Their algorithm is, however, not scalable for a large number of input pairs, as it is limited by the amount of memory available on the GPU. Our prototype builds on their work and solves this scalability issue.

GPGPU programming has received an increasing amount of attention over past years. Recent surveys show that applications for GPGPU can be found in a wide area of fields including database and data mining applications [ND10,OLG07]. For duplicate detection, however, most approaches have been targeted at distributed infrastructure and do not consider the unique challenges presented by GPUs. To the best of our knowledge, we are first to evaluate a complete duplicate detection workflow on GPUs.

3. Duplicate Detection Workflow

This section presents a complete duplicate detection workflow, which combines common duplicate detection algorithms with the computation capacities of modern graphics cards. Figure 1 gives an overview of the workflow with the following steps:

Parsing converts the input data, e.g., a CSV file, into an internal character array with all values concatenated. To allow values of different lengths, an additional array containing the starting indices of the individual attribute values is needed. This format is essential, because GPU-kernels can only handle basic data types and arrays with known sizes.

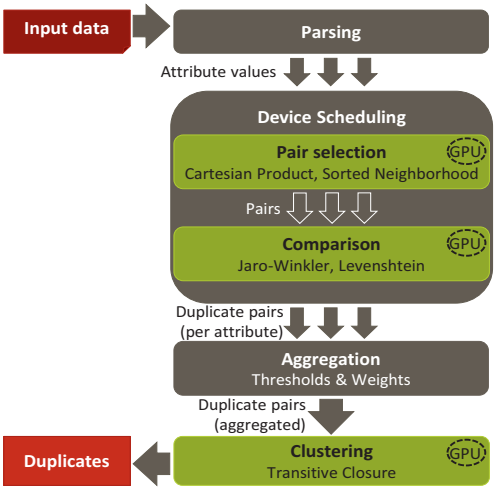


Figure 1: The duplicate detection workflow

Pair Selection selects record pairs for comparison. We adapt the *Cartesian product* and the *Sorted Neighborhood* algorithms to run on the GPU. To generate a sorting key for the *Sorted Neighborhood* algorithm, we present a simple *key-generation function* and an adapted *Soundex* algorithm, both running on the GPU.

Comparison: The selected record pairs are compared for similarity: We process each attribute value individually and return a normalized similarity value for each pair of attribute values. We describe the computation of two edit-based similarity measures on the GPU: *Levenshtein* and *Jaro-Winkler.*

Aggregation: The attribute similarities are aggregated to an overall record pair similarity, which is used to decide whether the two records are duplicates or not. We calculate a weighted average and check similarity values before and after the aggregation against predefined thresholds.

Clustering: The result of a pairwise duplicate detection process may not contain all transitively related record pairs. Thus, we calculate the transitive closure to obtain a complete list of duplicate clusters.

4. Pair Selection

Next to the Cartesian product, the literature knows several algorithms that select a subset of candidate pairs for comparison to avoid the complexity of comparing all pairs; a popular representative is the Sorted Neighborhood Method [HS95].

Regardless of the used algorithm, to completely utilize the parallel potential of GPUs, each work item compares exactly one selected pair of attribute values. This leads to a higher amount of work items than the GPU has processors, and, therefore, allows the GPU to use optimization techniques like memory latency hiding.

Since the memory of graphics cards is limited, it cannot fit all values of a large dataset. Thus, we cannot execute all comparisons at once and, instead, have to perform multiple comparison rounds. Each round consist of the following steps: Copy a subset of attribute values from the host to the GPU, execute the comparisons on those values, and finally copy the results back from the GPU to the host. We describe two approaches to divide the input values into blocks of data and select the comparisons for each round.

4.1 Cartesian product

The simplest method to select pairs is the Cartesian product. It selects every possible combination of input values. This leads to high recall, but also to a high number of comparisons. In general, the set of pairs must be split into chunks that fit into memory. This split can be performed easily with CPU and main memory due to dynamic memory allocation. But on the GPU, memory allocations must be done *before* the GPU executes the kernel code. Especially, different lengths of input values lead to different memory requirements for each comparison.

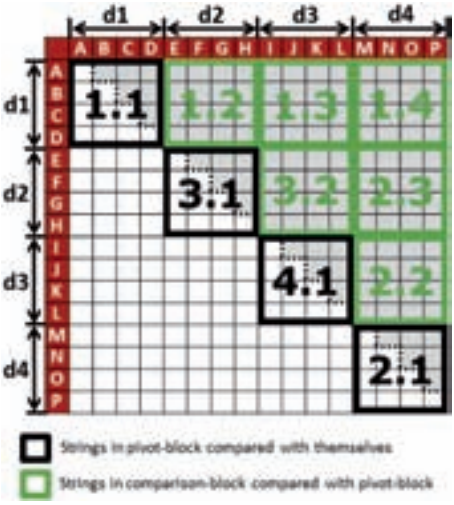


Figure 2: Cartesian product pair selection

For an optimal usage of GPU-resources two requirements must be met: First, the transfer of data between main memory and graphics cards should be minimized, i.e., data on the GPU should be reused as much as possible. Second, the entire available memory should be used to fully utilize the parallel potential of the GPU. To fulfill these goals, we establish two blocks of GPU memory of about the same size: The first block is the *pivot-block*, which is kept on the graphics card until all comparisons with its values are finished. The second block is the *comparison-block*, whose content is exchanged in each round.

Figure 2 shows which blocks of input data are compared. The x- and y-axes represent the input values; each cell represents a comparison between a value from the x- and a value from the y-axis. The comparisons under and on the diagonal (white cells) are never performed, because we assume symmetric comparison measures.

First, the pivot-block contains data `d1` and is compared with itself in round 1.1. Then in rounds 1.2 to 1.4, the comparison-block is filled with input data `d2` to `d4` and compared with the pivot-block. The data in the last comparison-block is then kept on the graphics card and used as the new pivot-block. The selection of the last comparison-block as the new pivot-block can lead to very small pivot-blocks, which in turn leads to fewer comparisons. To avoid this effect, the algorithm pre-calculates the optimal size of the new pivot-block based on the current pivot-block. We call it the *candidate-block,* and compare it with the pivot-block after all other comparison-blocks have been processed.

Assuming that the pivot-block contains p elements and the comparison-block contains c elements, we can do $p * c$ comparisons in parallel and thus maximally utilize the parallel potential of the GPU. The comparisons in rounds x.1 are exceptions, because they compare the pivot-block with itself, with $p * \frac{p-1}{2}$ comparisons in parallel. Every kernel has to calculate the memory addresses of the values that it should compare. To unify the calcu-

lation and to avoid branches, we increase the number of comparisons to $p * \left\lceil \frac{p-1}{2} \right\rceil$. Now, $p/2$ work items always compare the same string to one of the following $p/2$ strings – continuing at the beginning of the value array if its end is reached. This generates duplicate results if p is an even number, but the subsequent aggregation algorithm (see Sec. 5.4) filters them out.

We process input values with different lengths. Thus, we cannot use blocks of fixed size. Instead, the sizes of pivot- and comparison-blocks have to be determined based on input data and any additional memory required by a specific comparison algorithm. Additionally, the block-sizes are limited by the GPU-memory. This leads to the formula:

$$Memory \geq Strings + AlgorithmData + Results \qquad (1)$$

where *Strings* represents the size in bytes of the strings in both blocks (including the index arrays), *AlgorithmData* represents the individual requirements of a comparison algorithm, and *Results* is the size of the array that contains the calculated similarity values. Furthermore, the two goals of using the entire available memory and minimizing the data transfer have to be fulfilled by the value selection.

Our approach dynamically calculates the block's memory requirements depending only on the current lengths of the strings in the input data: First, it calculates the size of the pivot-block, which also depends on the strings in the comparison block, by increasing its size continuously. Since the strings of the comparison-block are not known at this time, the content of the comparison-block must be estimated. We assume that the comparison-block contains one string with average length for every string in the pivot-block. This approach fulfills the goal of maximizing the number of comparisons in each round. The pivot-block is filled with strings until the memory is too small to contain the pivot-block, the estimated comparison-block, and the additional memory for the comparison algorithm. Then the pivot-block is transferred to the GPU and compared with itself. Since the size and the content of the pivot-block are now fixed, the content of the comparison-block can be calculated based on the input data and the pivot-block. As the strings have different lengths, the comparison-block can contain more or fewer strings than the pivot-block. Our experiments show that usually both contain nearly the same number of strings, because of the average length estimation.

The Levenshtein algorithm for comparing attribute values needs additional memory: each comparison of a string from the pivot-block with a string from the comparison-block requires two times the size of the string from the pivot-block. Thus, it needs the size of the pivot-block times the number of strings in the comparison-block as additional memory (see Sec. 5.1 for more details). In the best case, the pivot-block contains many short strings while the comparison-block contains few long strings. Then, the Levenshtein algorithm will only need a small amount of memory and more strings can be placed on the graphics card. In the worst case, the pivot-block contains few long strings and the comparison-block contains many short strings. In this case, the Levenshtein algorithm needs more memory for the comparisons. To avoid the worst case, one could always compare the shorter with the longer string, but in this case the calculation of memory addresses in the kernel becomes overly complex. Jaro-Winkler does not need

additional memory for its comparisons; it uses the complete GPU-memory for attribute values. Thus, the number of comparisons does not depend on the contents of the blocks.

4.2 Sorted Neighborhood

The Sorted Neighborhood Method (SNM) [HS95] greatly reduces the number of comparisons compared to the Cartesian product. It consists of three phases: First, a sorting key is generated; then the records are sorted according to that key in the hope that duplicates have similar sorting keys and thus end up close to each other; and finally a fixed-size window is slid over the sorted records and all records within the same window are compared to each other.

For *key generation* we propose a simple hashing algorithm as well as the Soundex code (see Sec. 5.3): The simple hashing approach uses the string length and the first letter of the attribute to be compared, to create a sort key: $1{,}000 \cdot |String| + firstLetterCharValue$ results in a sorting primarily according to the length and secondly according to the first letter. Sorting by length leads to comparisons of strings of roughly same length, which reduces branch divergence – an advantage in GPU processing. Further, the first letter is often the same for duplicate strings, e.g., because spelling mistakes are less likely to be made here [YF83], and because abbreviations start with the same letter. The computation of the simple sort key requires no branching, and therefore all GPU threads can run in parallel. The downside of this simple key generation is that it produces poor results if there are many strings with the same length, which results in many similar keys. In this case, Soundex (described in Sec. 5.3) produces better sort keys, because it focuses on phonetic characteristics instead of the string length. Altogether, sorting key generation is well suited for GPUs, because each calculation only depends on one string, and therefore can be easily computed in parallel.

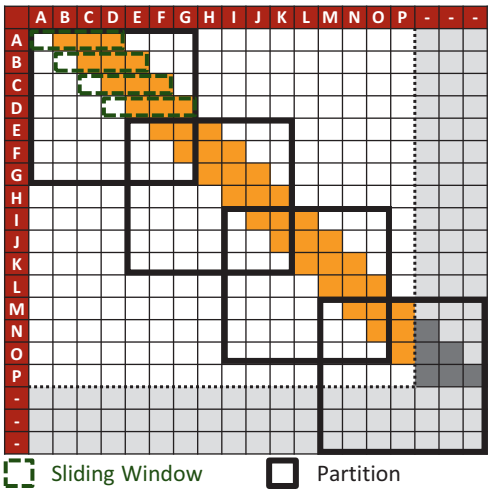


Figure 3: Sorted Neighborhood pair selection

For the sorting step, we choose the GPU-based merge sort algorithm of [SKC10]. We did not improve this sorting algorithm; therefore, we do not cover it in this paper.

Figure 3 shows the comparison matrix for SNM. The colored cells on the diagonal denote the comparisons that are actually performed by SNM. To compare the attribute values efficiently, we have to determine the maximum amount of strings that can be transferred to the GPU at a time. In Fig. 3, a partition visualizes the comparisons that can be done with the strings on the GPU in one execution. We approximate the number of the strings

that can be copied to the GPU based on the average string length. The amount of memory required for one string depends on the length of the string, the number of strings it is compared with, and the comparison algorithm.

The Sorted Neighborhood approach compares each string with the next $w-1$ strings where w is the window size (see the sliding window, Fig. 3). This leads to $\big((w-1)\cdot |AllStringsInPartition|\big) - (w-1)^2$ comparisons in one round on the GPU. In this formula, $(w-1)^2$ denotes the comparisons that are postponed to the next partition, due to the window sliding out of the partition boundaries.

With this number of comparisons and the average string length, we approximate the maximum number of strings that can be copied to the GPU. Since the input strings have different length, we iteratively calculate the required memory based on the approximation, until the maximum number of strings that can be computed on the graphics card is determined. The calculation benefits from the internal data format, produced during the parsing step (see Sec. 3). It allows the computation of the string lengths just by inspecting the index array with the starting indices of the strings.

Once the data is copied onto the GPU, each string within a partition is compared with the next $w-1$ strings. The last strings in a partition cannot be compared, because they are not followed by $w-1$ strings. Therefore, partitions have to overlap by $w-1$ strings to ensure that no comparisons are missed. To execute the comparisons in the last partition efficiently, the index array is expanded by $w-1$ dummy string entries (see header-cells labeled with "-" in Fig. 3). These dummy strings prevent branching, because the last strings of the final partition can be treated like any other string without needing conditional checks. Furthermore, the dummies are empty, so their respective comparisons can easily be omitted by the kernels. Thus, they do not negatively impact computation time.

5. Similarity Classification

This section describes implementations of methods to classify record pairs as duplicate or non-duplicate. In particular, we present two edit-based and one phonetic (Soundex) similarity measure to calculate the attribute similarity on graphics cards. Then we describe how different attribute similarities are aggregated to record similarities and how we cluster results using GPUs.

5.1 Levenshtein similarity

The Levenshtein distance is defined as the minimum number of character insertions, deletions, and replacements necessary to transform a string s_1 into another string s_2 [NH10]. To compute the Levenshtein distance $LevDist(s_1, s_2)$ on a GPU, we use a dynamic programming approach [MNU05] and extend this approach to optimize its memory usage. The comparison of two strings requires a matrix M of size $(|s_1|+1)\times(|s_2|+1)$, where $|s|$ denotes the length of string s. A value in the i-th row and j-th

column of M is defined by $M_{i,j}$, where $0 \leq i \leq |s_1|$ and $0 \leq j \leq |s_2|$. We initialize the first row $M_{0,j}$ and column $M_{i,0}$ as:

$$M_{0,j} = j \qquad M_{i,0} = i \quad (2)$$

The algorithm then iterates from the top left to the bottom right cell of the matrix. It recursively computes each value $M_{i,j}$ in the matrix as:

$$M_{i,j} = \begin{cases} M_{i-1,j-1} & \text{if } s_{1,i} = s_{2,j} \\ 1 + min(M_{i-1,j}, M_{i,j-1}, M_{i-1,j-1}) & \text{otherwise} \end{cases} \quad (3)$$

where $s_{k,i}$ denotes the i-th letter in the string s_k. In the end, matrix cell $M_{|s_1|,|s_2|}$ delivers the Levenshtein distance between s_1 and s_2.

Because dynamic memory allocation is not possible from inside a GPU kernel in OpenCL, we pre-allocate the needed memory for each comparison. To reduce memory consumption, we use only two matrix rows for each comparison, because calculation of row i depends only upon the current row i and the previous row $i-1$ (see Equation 3). Thus, we can swap the current and previous row and calculate row $i+1$ by overwriting the values of row $i-1$, without affecting performance. We analyzed that the average string length in our test collection is 14 characters, which results in an average matrix size of $(14+1) \cdot (14+1) = 225$ cells. By using only two rows, we can greatly reduce the average required cells in our test collection to $(14+1) \cdot 2 = 30$, which is only 13% of the whole matrix.

To calculate the amount of required memory for the matrix rows, the algorithm can use a simple formula that takes the arbitrary length of each string $|s_k|$ into account. Let n be the number of strings that should be compared and c be the number of strings to which each of the n strings is compared to. Then the overall memory in byte that is required for the matrix rows can be calculated as:

$$c \cdot sizeof(int) \cdot 2 \cdot \sum_{k=1}^{n} (|s_k| + 1) \quad (4)$$

Within a comparison of two strings, one string defines the length of the two matrix rows. Therefore, one comparison requires $sizeof(int) \cdot 2 \cdot (|s_k| + 1)$ bytes of memory for the matrix rows. Our pair selection algorithms are designed to compare each of the n strings to $c > 0$ other strings, so that each string defines c times the length of the matrix rows. For example, the Sorted Neighborhood algorithm sets $c = w - 1$ and the Cartesian product defines $c = \left\lceil \frac{n-1}{2} \right\rceil$. To calculate the overall amount of matrix memory, the algorithm sums up all n string specific row lengths $|s_i| + 1$.

As usual, to transform the Levenshtein distance into a normalized similarity measure, we finally normalize the distance by dividing by the length of the longer string and subtract the result from 1.

5.2 Jaro-Winkler similarity

Jaro-Winkler similarity was originally developed for the comparison of names in U.S. census data. The measure is comprised of the Jaro distance [Ja89] and additions by Winkler [WT91]. The Jaro distance $JaroDist(s_1, s_2)$ combines the number of common characters m between two strings s_1 and s_2, the number of transpositions t between the two strings of matching characters, and the lengths of both strings:

$$JaroDist(s_1, s_2) = \frac{1}{3} \cdot \left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m} \right) \quad (5)$$

Common characters are only searched for within a range of size w:

$$w = \frac{max(|s_1|, |s_2|)}{2} - 1 \quad (6)$$

Winkler's main modification of the Jaro distance is the inclusion of the length of the common prefix ℓ into the formula (see Eq. 7), which improves similarity scores for names starting with the same prefix. The common prefix is limited to ≤ 4 and is weighted by the factor p, for which Winkler's default value is 0.1.

$$JaroWinkler(s_1, s_2) = JaroDist(s_1, s_2) + \left(\ell p \left(1 - JaroDist(s_1, s_2) \right) \right) \quad (7)$$

The original algorithm calculates the number of transpositions t by first calculating the two strings of matching characters and then comparing them character by character. For each pair of strings being compared, matching characters are stored in two temporary variables of length $l_{min} = min(|s_1|, |s_2|)$. For graphics cards, these variables pose difficulties: First, the amount of fast private/local memory is limited, which restricts the amount of work items that can be executed in parallel. In order to still achieve high levels of parallelism, global memory needs to be used, which is slower to access but also several magnitudes larger. Second, since GPU memory cannot be allocated dynamically within a kernel at runtime, we would have to wastefully pre-allocate the worst-case amount of memory or perform additional comparisons as in [HYF08].

Our approach focuses on reducing memory consumption with the goal of achieving high levels of parallelism while primarily using fast private/local memory. This comes at the cost of increased kernel-time complexity. Instead of pre-computing the strings of matching characters, our algorithm (see Alg. 1) computes the number of matched characters and transpositions by iterating the input strings twice. The first iteration (lines 3-9) finds and counts the number of matching characters m. It also keeps track of which characters have been matched already (array of matched characters $mc_{1,x}$). The second iteration (l. 12-25) calculates the number of half-transpositions t and the length of the common prefix ℓ. The $find(c, s, mc)$ function (l. 4 and l. 13) tries to find a character c in the given string s, without matching any characters that were previously matched. This is done by checking that the respective position in the mc array is not set to 1, and by respectively updating the array once a matching character has been found.

The $find$ function returns a Boolean value indicating whether a match was found, and the offset at which it was found. Internally, the function also considers the window size w (see Eq. 6) to match only characters within the allowed range of $i \pm w$.

The $countUnmatched(i, mc)$ function (l. 15) counts how many characters in s_2 up to position i cannot be matched to a character in s_1 (by inspecting the mc array). A half-transposition exists if a character can be matched without an offset, while ignoring all unmatched characters (l. 16). The prefix counter ℓ is only increased for the first 4 characters, if the current character has been matched without an offset, and all characters on earlier positions do also match respectively (l. 19).

The key to memory efficiency with this algorithm lies in the arrays $mc_{1,x}$ and $mc_{2,x}$ which store only Boolean values and thus can be represented at the level of single bits. Our implementation uses two 8-byte variables allowing comparisons of strings up to length 64. Using the original approach we would need two 64-byte variables to compare strings of the same length.

```
01 m ← 0, t ← 0, ℓ ← 0
02 mc1,x ← 0
03 for i = 1 to |s1| do
04  [match,offset] ← find(s1,i,s2,mc1)
05  if match = True then
06   m ← m + 1
07   mc1,i ← 1
08  end if
09 end for
10 mc2,x ← 0
11 uc1 ← 0
12 for i = 1 to |s1| do
13  [match,offset] ← find(s1,i,s2,mc2)
14  if match = True then
15   uc2 ← countUnmatched(i+offset,mc1)
16   if offset + uc1 ≠ uc2 then
17    t ← t + 1
18   end if
19   if offset = 0 and ℓ = i - 1
        and i ∈ [1,4] then
20    ℓ ← ℓ + 1
21   end if
22  else
23   uc1 ← uc1 + 1
24  end if
25 end for
26 t ← t/2
27 return m, t, ℓ
```

Algorithm 1: Jaro-Winkler: computation of matching characters **m**, transpositions **t** and common prefix ℓ for two strings $\mathbf{s_1}$ and $\mathbf{s_2}$

5.3 Soundex

Soundex is a phonetic algorithm for identifying words that are pronounced similarly but spelled differently [USN07]. The algorithm produces 4-letter codes, which match for similar sounding words, e.g., Robert and Rupert are both represented by the code R163. Soundex is good for finding misspelled names but it produces many false positives as well as false negatives [PS01].

Our implementation consists of two kernels: One for generating Soundex codes for a set of input strings, and one for comparing pairs of Soundex codes. For comparison we minimize memory operations by leaving generated Soundex codes on the graphics card for the comparison phase. To generate Soundex codes we walk through the letters of a given term and build up the Soundex code by either coding the current letter or moving

on to the next one. The compare-kernel uses lists of previously generated Soundex codes to create pairs of terms that have the same code. Unlike other similarity measures, this results in similarity values of either 0 or 1.

5.4 Aggregation

To classify whether two records are a duplicate or not, we aggregate the attribute similarities to an overall record similarity. The comparators described in the previous sections return lists of pairs with similarity values above attribute-specific thresholds. The aggregated similarity value is a weighted average of all similarity values for the specific pair. In order to increase the precision of results, merged pairs with a similarity value below a manually defined overall threshold are removed.

For efficient aggregation each list is first sorted by a unique identifier that represents the compared data records. This approach reduces the search time for corresponding pairs in the result lists; additionally, duplicate entries that may have been produced by the Cartesian product (see Sec.4.1) can be removed easily.

While the sorting part is suited for the GPU, the merging part is not: First, merging on the CPU can be a simple Sort-Merge join that requires linear time, so the additional time required for copying the data to the GPU does not pay off (see Sec. 6.2). On the GPU, the retrieval of corresponding pairs is more complex, because each kernel instance would merge one combination of pairs and corresponding pairs in different lists cannot be found at the same defined places. Pairs can be missing in some lists, due to attribute-specific thresholds and different comparisons that are triggered by the Sorted Neighborhood method. Thus, a GPU variant would either need a complex kernel with slow branching, or additional preprocessing of all lists. When iterating all lists, the computation of the weighted average would only produce little to no computational overhead. This invalidates the point of using the GPU for merging, so we sort the lists on the GPU and merge them on the CPU.

5.5 Clustering

As we use pairwise comparisons to find duplicate records, our result may not be transitively closed (e.g. pairs ⟨A,B⟩ and ⟨B,C⟩ are classified duplicates, but not ⟨B,C⟩). We calculate the transitive closure using the tiled Floyd-Warshall (FW) algorithm by Katz and Kider [KK08] and adapt it to the specific task of clustering real-world duplicate pairs. We first present the tiled FW algorithm in a condensed form. Afterwards, we describe how the algorithm can be extended to optimize its efficiency and scalability in computing extremely large amounts of data.

The tiled FW extends the original FW [Wa62] in order to run efficiently on the GPU. It uses dynamic programming and is based on a directed graph, represented by an adjacency matrix M. In the design of the tiled FW, Katz and Kider assume that the entire matrix for n vertices can be loaded into the GPU's global memory at once, but not into local memory. Therefore, they propose to load all data into global memory first and then split

the computation of the transitive closure into many sub-tasks that can be executed sequentially using maximal local memory in each step. After loading M into global memory, the tiled FW algorithm partitions M into sub-matrices of size $s \times s$ with $s \leq n$. Size s must be chosen small enough so that three sub-matrices can be loaded into local memory at once. M then consists of $m \times m$ sub-matrices with $m = \lceil n/s \rceil$. Afterwards, the algorithm uses an iterative execution strategy for the Floyd-Warshall algorithm (see Fig. 4). It needs m stages to calculate the complete transitive closure. Each stage consists of the following three phases:

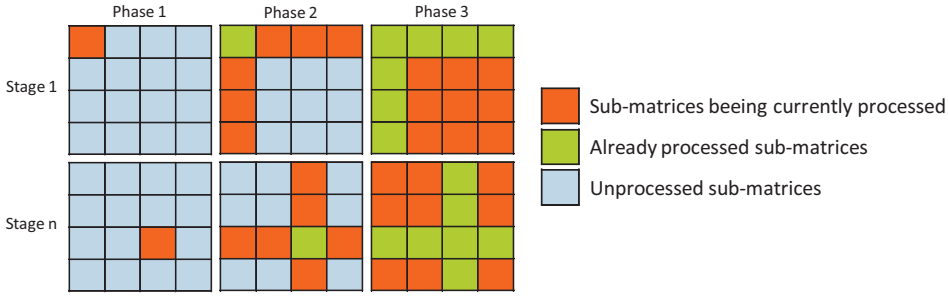


Figure 4: Stages and phases of the tiled FW algorithm introduced by Katz and Kider [KK08].

1. Start one work group: The work group loads the submatrix (i, i) as pivot matrix into local memory, where i is the current stage number. Then, only one thread in this work group calculates the transitive closure for this matrix using the original Floyd-Warshall algorithm.
2. Start $(m-1) \cdot 2$ work groups: Each work group loads the pivot matrix (i, i) and a second sub-matrix (j, i) or (i, j) into local memory. Now the second submatrix is located in the same row or column as the pivot matrix. Its calculation depends only upon itself and the pivot matrix. Within a work group, each value in the second sub-matrix can be computed in parallel by an own thread executing a part of the Floyd-Warshall algorithm (for more details see [KK08]).
3. Start $(m-1)^2$ work groups: Each work group loads two sub-matrices that have been processed in phase 2 and a third sub-matrix into local memory. The third matrix for two previously processed matrices (k, i) and (i, j) is placed at (k, j) and only depends upon their values and itself in this step. Again, all values of the third matrix can be processed in parallel by an own thread executing a part of the Floyd-Warshall algorithm.

In the following, we adapt the approach of Katz and Kider to the specific task of clustering duplicate pairs and add some modifications to improve the algorithm's efficiency and scalability.

5.5.1 Optimizing transitive closure efficiency

The adjacency matrix defines a directed graph, whereas our result graph is undirected, as we assume a symmetric duplicate relation between different records. Thus, all values in the adjacency matrix are mirrored across the matrix's diagonal axis. An obvious optimi-

zation approach is to remove redundant edges and hence reduce both the matrix size and the necessary computation steps in Phases 2 and 3 of the tiled FW algorithm. In Phase 2, for example, the algorithm could compute only the sub-matrices (i, j) with $j > i$ and (j, i) with $j < i$. Nevertheless, the overall performance would decrease for two reasons: First, the computation of the edge position in the matrix becomes more complex. Whenever the algorithm needs to read an edge value from the redundant (and therefore not existing) half of the matrix, it must mirror the edge's coordinates to find the corresponding value, which is a complex operation especially in Phases 2 and 3. Second, Warshall's algorithm might write the edges (x_i, x_j) and (x_j, x_i) at the same time. To guarantee consistent write operations, the kernels would need locking mechanisms, which decrease performance and restrains parallelism. Thus, we retain the original matrix and store each duplicate pair as two directed edges in the adjacency matrix.

The original tiled FW represents each value in the adjacency matrix as a single numerical value. To reduce the physical size of the matrix in memory, our implementation of the algorithm encodes these values as bitmasks: Each bitmask contains 32 edge values, because common GPUs address 32 bits at once. This technical optimization reduces the required memory by 1/32 compared to integers. However, this compression also impacts the structure of the algorithm: While computing the transitive closure, Floyd-Warshall's algorithm iterates over multiple rows and columns of the matrix. Each read operation returns 32 edge values. A horizontal iteration over a row containing bitmasks of edge values can be done very fast, because it needs $\lceil n/32 \rceil$ read operations to receive n edge values. In contrast, a vertical iteration over a column of the matrix still needs n read operations for n edges and returns $31 \cdot n$ not required values. This becomes a drawback for the performance, if we execute the Floyd-Warshall algorithm on a bit-compressed graph matrix. Warren's algorithm [Wa75], which extends the Floyd-Warshall algorithm, solves this problem by just iterating horizontally in the adjacency matrix. So we use this approach instead of Warshall's algorithm to calculate Phase 1 without iterating vertically. In Phases 2 and 3, the algorithm can use the redundant edges in the adjacency matrix to avoid vertical iterations. Each column c in the matrix has a corresponding row r that is mirrored across the matrix's diagonal axis and contains the same bit values. Therefore, all iterations over c can be replaced by iterations over r.

Using bitmasks to encode the matrix also affects the granularity of parallelization. In Phases 2 and 3 the algorithm can no longer compute the value of each single edge in parallel. To guarantee consistent writes, each bitmask must be processed by one GPU thread. However, by using bitwise *OR* operations for the comparison of two bitmasks, each thread computes all 32 values at once.

Figure 5 shows how all previously described modifications of the tiled FW work together in Phase 2. In this phase, each work group loads the pivot and a second submatrix into local memory. Then, all bitmasks in the second sub-matrix are computed in parallel.

Let $(x_k, y_i) - (x_k, y_j)$ be a bitmask b in the second submatrix. The thread that processes b iterates over row x_k in the pivot matrix and analyses each bit. If a bit (x_k, x_l) is 1, the thread loads the bitmask $(x_l, y_i) - (x_l, y_j)$ from the second matrix and then compares it to b using the bitwise *OR* operation. After analyzing the whole row x_k in the pivot ma-

trix, the thread writes the new values for *b* into the second matrix. This algorithm also works for Phase 3. In this phase, three sub-matrices are loaded into local memory. To compute the bitmask *b* in the third matrix, a thread iterates over the corresponding row in the horizontally deferred second matrix and loads bitmasks for the comparison from the vertically deferred second matrix.

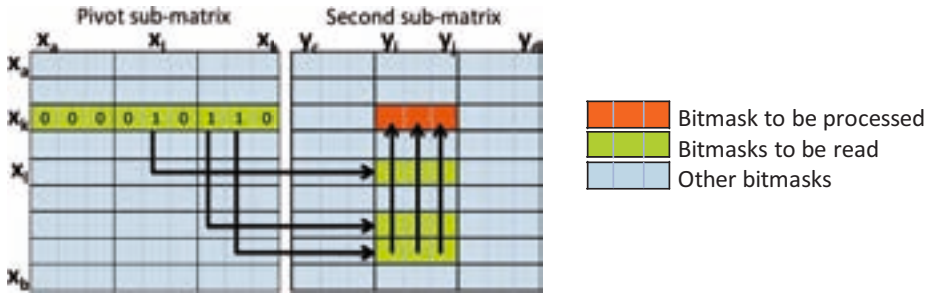


Figure 5: Optimized calculation of Phase 2 using bitmask encoding, horizontal iteration and bitwise OR comparison.

5.5.2 Achieving scalability

The algorithm of Katz and Kider assumes that the entire adjacency matrix fits into the GPU's global memory. Given a GPU with 1GB of global memory, this assumption limits the maximum number of nodes in the result graph to 92,672 even if bitwise encoding is used. Assuming 5% duplicates as result size, this is not enough to analyze datasets with 2 million records or more. Therefore, we need an additional partitioning of the matrix between the host's main memory and the GPU's global memory. We achieve this partitioning by using the same stage-wise execution strategy of the tiled FW again to pre-partition the global adjacency matrix G into smaller, quadratic matrices M_i on the host. The algorithm has to ensure that all matrices M_i are equally large and that three matrices M_i fit into the global memory at once. We call this approach the double tiled FW algorithm. It uses the same stages and phases of loading matrices M_i into global memory like the original tiled FW loads sub-matrices into the local memory. In Phase 1, only one pivot sub-matrix M_i resides in global memory. The GPU processes this matrix by executing the already known tiled FW. In Phase 2, the algorithm loads the pivot and a second sub-matrix into global memory. All bitmasks in the second sub-matrix are then processed in parallel like in Stage 2 of the tiled FW (see Fig. 5). Afterwards, the same procedure is used for Phase 3, which needs one pivot and two previously processed second sub-matrices.

6. Evaluation

We evaluated performance and accuracy of our workflow using real-world data sets. In addition, the execution time of each component is evaluated on different hardware.

6.1 Experimental setup

We evaluated on four different graphics cards, two from NVIDIA and two from ATI. As ATI's OpenCL drivers also allow the execution of OpenCL kernels on CPUs, we additionally evaluated our implementation on two Intel CPUs (see Tab. 2 for specifics of all six devices).

We used a subset of 1.792 million music CDs extracted from freedb.org for the performance evaluation of our algorithms. This dataset contains attributes artist, title, genre, year of publication, and multiple tracks. The DuDe Duplicate Detection Toolkit [DN10] provides a gold-standard for a randomly selected subset of 9,763 CDs (http://www.tinyurl.com/dude-toolkit), which we used to measure the accuracy of our results. Furthermore, we calculated the similarity of two records based on the values of four attributes that contain strings of variable length, namely Artist, Title, Track01, and Track02. This selection is based on our experience with that database.

To ensure a realistic assessment of the workflow efficiency, we first evaluated its effectiveness. We calculated precision (proportion of retrieved real duplicates), recall (proportion of identified real duplicates), and F-measure (harmonic mean of precision and recall) for different configurations: Sorted Neighborhood (SNM) and Cartesian product (CP) for pair selection combined with Levenshtein (L) and Jaro-Winkler (JW) as comparison algorithms. Table 1 lists the configuration parameters that delivered the best F-measure, showing similar results compared to other duplicate detection tools [DN10]. In Sec. 6.2, we use these configuration parameters to test the performance of our algorithm

For SNM, we tested window sizes between 10 and 500. We observed that any value above 20 has only minimal effect on the F-measure (at best 2 percentage points increase). Therefore, all experiments used a window size of 20. For the SNM's sort key generation, we tested two different generating algorithms. As already mentioned in Sec. 4.2, the Soundex algorithm generates the best sort keys for attributes whose values have similar lengths, which is true for the artist and track attributes. The values of the title attribute, however, vary considerably in length. As a result, our own key generation algorithm performs better for these attributes.

We tried multiple thresholds to determine whether a pair with a certain similarity is classified as a duplicate. The thresholds are first applied to attribute pairs during comparison and afterwards to record pairs during aggregation. The aggregation step additionally uses a set of weights to sum up the single attribute similarities. We evaluated various sets of thresholds and weights and settled on the values in Tab. 1.

Method	Thresholds				Weights			Precision	Recall	F-Measure
	Overall	Artist	Title	Tracks	Artist	Title	Tracks			
SNM + L	0.6	0.6	0.6	0.5	20%	30%	25%	95.2%	80.3%	87.1%
SNM + JW	0.66	0.6	0.67	0.87	20%	30%	25%	95.2%	79.6%	86.7%
CP + JW	0.66	0.78	0.75	0.87	20%	30%	25%	92.2%	86.6%	89.3%

Table 1: Configurations and results

ID	Type	Device Name	Clock	Memory	Cores	System	Price (August 2011, http://www.alternate.de)
G1	GPU	Nvidia GeForce GTX 570	732 MHz	1280 MB GDDR5	480 CUDA	Win64	279 Euro
G2	GPU	Nvidia Tesla C2050	1147 MHz	3071 MB GDDR5	448 CUDA	Linux64	2,149 Euro
G3	GPU	ATI Radeon HD 5700	850 MHz	1024 MB GDDR5	800 SP	Win64	91 Euro
G4	GPU	ATI Mobility Radeon HD 5650	450 MHz	1024 MB GDDR3	400 SP	Win64	*unknown*
C1	CPU	Intel Core i5 750	2.67 GHz	8192 MB DDR3	4	Win64	185 Euro
C2	CPU	Intel Core i5 M560	2.67 GHz	8192 MB DDR3	2	Win64	200 Euro

Table 2: Evaluation devices

6.2 Algorithmic complexity

To evaluate the performance of the duplicate detection workflow, we analyzed the execution times of its individual components. All tests were executed on the NVIDIA GeForce GTX 570 (G1), because our experiments in Sec. 6.3 show that this device performs best.

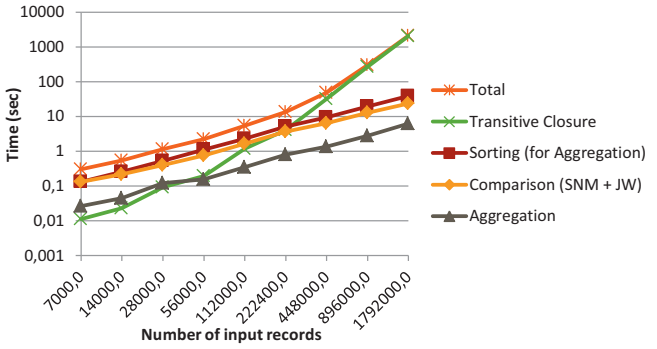


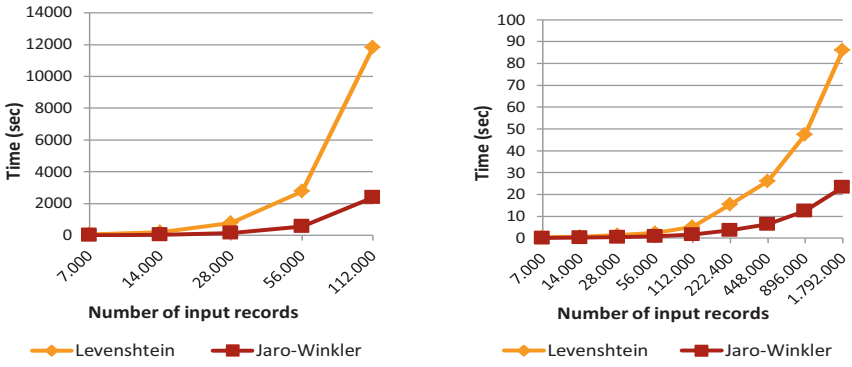
Figure 6: Execution times of different components

Figure 6 shows the execution times of the different components as parts of the complete workflow for various input sizes n. We used Jaro-Winkler for comparison and Sorted Neighborhood for pair selection. In the following, l denotes the longest list of found attribute-wise duplicates after the comparison, and m denotes the number of record-wise duplicates after the aggregation step. Since we observed that l and m increase linearly in proportion to n, the complexities of the subsequent algorithms can be defined in relation to the input size n.

The diagram shows that with an increasing amount of data, and thus an increasing amount of duplicates, the execution time of the transitive closure becomes the dominant part of the workflow. Note that for a complete result one cannot omit this last step and that its complexity is hardly dependent on the total number of previously found duplicates, but rather on the number of disjoint records in the duplicates. The execution time of the transitive closure increases fastest, because its complexity is $\mathcal{O}(n^3)$, whereas the other complexities are $\mathcal{O}(n^2)$ for the Cartesian product, $\mathcal{O}(n \log(n))$ if $w \leq \log(n)$ or otherwise $\mathcal{O}(w \cdot n)$ for the Sorted Neighborhood, and $\mathcal{O}(n \log(n))$ for the aggregation.

The sorting, as an aggregation preprocessing step, has the second highest time; more advanced algorithms [SKC10] might improve this value. The comparisons also have high execution times, because a string comparison is the most complex calculation on the GPU. The aggregation step itself has the smallest execution time and thus has only little impact on the workflow's overall execution time.

Figure 7 shows the execution times for the comparators only. We observe that Jaro-Winkler has a much lower execution time for both pair-selection algorithms for three reasons: First, Levenshtein performs more accesses to global GPU-memory. Second, Levenshtein performs more comparison rounds due to the higher memory consumption; these rounds need additional time to be triggered by the host. Third, Jaro-Winkler creates more work items allowing the GPU to use memory latency hiding to optimize the execution.



Cartesian Product Sorted Neighborhood

Figure 7: Execution times of comparison algorithms in combination with different pair selectors

6.3 Comparison of hardware

As discovered in Sec. 6.2, the most efficient configuration uses Sorted Neighborhood in combination with the Jaro-Winkler comparison algorithm. Figure 8 shows that the best results are indeed achieved on GPUs. The fastest GPU G1 (see Tab. 2) takes 35 minutes (2,095 seconds) to process 1.792 million entries; this is about 10 times faster than the fastest CPU C2, which takes 335 minutes. However, Tab. 2 shows that the CPUs in our experimental setup are cheaper than the used GPUs. To compare them in a fair way, we placed the execu-

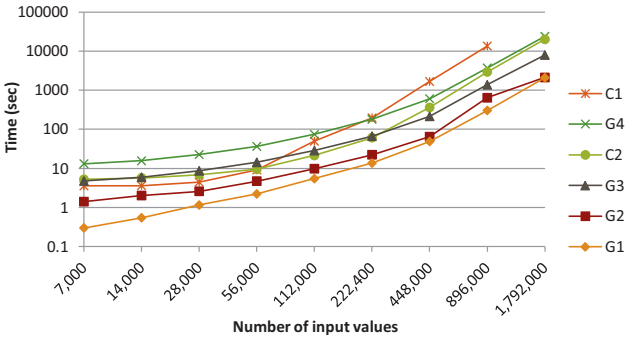


Figure 8: Performance of Sorted Neighborhood with Jaro-Winkler on different devices

tion times in relation to the prices by multiplying the price (in Euro) and the execution time (in minutes). This gives us a measure for the price-performance ratio, which assigns lower numbers to better devices. Since a GPU cannot be operated without a CPU, we add the price for the cheapest CPU. Still under this measure, the GPUs perform better than the CPUs: Again, for 1.792 million entries, G1 has the best results with a value of $(279 + 185) \cdot 35 = 16{,}240\ EuroMins$ compared to the best CPU C2 with a value of $200 \cdot 335 = 67{,}000\ EuroMins$; this is a 4-fold better price-performance ratio for the GPU.

7. Conclusion

We have presented and evaluated a complete duplicate detection workflow that uses graphics cards to speed up execution. The workflow uses either the Cartesian product or the Sorted Neighborhood approach for pair selection, and calculates the similarity of a record pair using Levenshtein, Jaro-Winkler, and Soundex. The evaluation of our workflow shows that modern GPUs can execute the duplicate detection workflow faster than modern CPUs. It has also been shown that the workflow and algorithms are scalable and can process large datasets.

The experiments also show that the access of global memory on graphics cards is indeed a bottleneck and has great impact on the performance of our algorithms. Profiling has shown that reads and writes are mostly non-coalesced and therefore very slow. To solve this problem in the future, all strings could be interlaced, which is a complicated task when using strings of variable lengths. Also, the use of local memory could further speed up execution. More optimizations concerning concrete hardware devices are possible and could be applied to a concrete usage of the workflow [FTP11]. Currently, only the comparisons of different attributes are distributed over all available devices. Thus, other algorithms, especially the computation of the transitive closure, could be further optimized to scale out on multiple devices. The implementation and evaluation of more similarity measures, e.g., token-based approaches, would allow the processing of real-world data with different properties and make the workflow more adaptable.

Acknowledgments: This research was supported by the HPI Future SOC Lab and the German Research Society (DFG grant no. NA 432). We thank Frank Feinbube (HPI) for his support.

References

[AJ88] R. Agrawal and H. V. Jagadish. Multiprocessor transitive closure algorithms. In *Proceedings of the first international symposium on Databases in parallel and distributed systems* (DPDS), 56-66, Los Alamitos, 1988.

[CCH10] P. Christen, T. Churches, and M. Hegland. Febrl - a parallel open source data linkage system. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining* (PAKDD), 638-647, Sydney, 2004.

[DN10] U. Draisbach and F. Naumann. DuDe: The duplicate detection toolkit. In *Proceedings of*

the International Workshop on Quality in Databases (QDB), Singapore, 2010.

[EIV07] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering* (TKDE), 19(1):1-16, Piscataway, 2007.

[FTP11] F. Feinbube, P. Tröger, and A. Polze. Joint Forces: From Multithreaded Programming to GPU Computing. *IEEE Software*, 28(1):51-57, 2011.

[HS95] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. *In Proceedings of the ACM International Conference on Management of Data* (SIGMOD), 127-138, San Jose, 1995.

[HYF08] B. He, K. Yang, R. Fang, M. Lu, N.K. Govindaraju, Q. Luo, P.V. Sander: Relational joins on graphics processors. *In Proceedings of the ACM International Conference on Management of Data* (SIGMOD), 511-524, Vancouver, Canada, 2008.

[Ja89] M. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):414-420, 1989.

[KK08] G. Katz and J. Kider Jr. All-pairs shortest-paths for large graphs on the GPU. In *Proceedings of the ACM Symposium on Graphics Hardware* (SIGGRAPH), 47-55, Los Angeles, 2008.

[KKH10] T. Kirsten, L. Kolb, M. Hartung, A. Groß, H. Köpcke, and E. Rahm. Data partitioning for parallel entity matching. *Proc. of the VLDB Endowment*, 3(2), Singapore, 2010.

[KL07] H. Kim and D. Lee. Parallel linkage. *In Proceedings of the International Conference on Information and Knowledge Management* (CIKM), 283-292, Lisbon, 2007.

[KTR11] L. Kolb, A. Thor, and E. Rahm. Parallel sorted neighborhood blocking with mapreduce. In *Proceedings of the Conference Datenbanksysteme in Business, Technologie und Web Technik* (BTW), 45-64, Kaiserslautern, 2011.

[MNU05] V. Makinen, G. Navarro, and E. Ukkonen. Transposition invariant string matching. *Journal of Algorithms*, 56(2):124-153, 2005.

[ND10] J. Nickolls and W. Dally. The GPU computing era. *Micro*, IEEE, 30(2):56-69, 2010.

[NH10] F. Naumann and M. Herschel. *An Introduction to Duplicate Detection*. Morgan & Claypool, 2010.

[OLG07] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80-113, 2007.

[PS01] F. Patman and L. Shaefer. Is Soundex good enough for you? On the hidden risks of Soundex-based name searching. *Language Analysis Systems*, Inc., Herndon, 2001.

[SKC10] N. Satish, C. Kim, J. Chhugani, A. D. Nguyen, V. W. Lee, D. Kim, and P. Dubey. Fast sort on CPUs and GPUs: a case for bandwidth oblivious SIMD sort. In *Proceedings of the ACM International Conference on Management of Data* (SIGMOD), 351-362, Indianapolis, 2010.

[To91] A. Toptsis. Parallel transitive closure computation in highly scalable multiprocessors. *Advances in Computing and Information* (ICCI), 197-206, Ottawa, 1991.

[USN07] The U.S. National Archives and Records Administration. The Soundex indexing system, May 2007. URL: http://www.archives.gov/research/census/soundex.html. Retrieved on Sept. 1, 2011.

[Wa62] S. Warshall. A theorem on Boolean matrices. *Journal of the ACM*, 9(1):11-12, 1962.

[Wa75] H. Warren Jr. A modification of Warshall's algorithm for the transitive closure of binary relations. *Communications of the ACM*, 18(4):218-220, 1975.

[WT91] W. E. Winkler and Y. Thibaudeau. An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census. In *U.S. Decennial Census. Technical report, US Bureau of the Census*, 11-13, 1991.

[YF83] E. J. Yannakoudakis and D. Fawthrop. The rules of spelling errors. *Information Processing and Management*, 19(2):87-99, 1983.

Experimental Evaluation of NUMA Effects on Database Management Systems

Tim Kiefer, Benjamin Schlegel, Wolfgang Lehner
Technische Universität Dresden
Database Technology Group
Dresden, Germany
{tim.kiefer, benjamin.schlegel, wolfgang.lehner}@tu-dresden.de

Abstract: NUMA systems with multiple CPUs and large main memories are common today. Consequently, database management systems (DBMSs) in data centers are deployed on NUMA systems. They serve a wide range of database use-cases, single large applications having high performance needs as well as many small applications that are consolidated on one machine to save resources and increase utilization.

Database servers often show a natural partitioning in the data that is accessed, e.g., caused by multiple applications accessing only their data. Knowledge about these partitions can be used to allocate a database's memory on the different nodes accordingly: a strategy that increases memory locality and reduces expensive communication between CPUs.

In this work, we show that partitioning a database's memory with respect to the data's access patterns can improve the query performance by as much as 75%. The allocation strategy is enabled by knowledge that is available only inside the DBMS. Additionally, we show that grouping database worker threads on CPUs, based on their data partitions, improves cache behavior, which in turn improves query performance. We use a self-developed synthetic, low-level benchmark as well as a real database benchmark executed on the MySQL DBMS to verify our hypotheses. We also give an outlook on how our findings can be used to improve future DBMS performance on NUMA systems.

1 Introduction

Servers with 2, 4, or 8 CPUs on a single board, many cores, and non-uniform memory access (NUMA systems) to 64, 128, or more gigabytes of RAM are common today. NUMA systems are scalable and offer the ease of programming with distributed memory hidden behind a global address space and a cache coherence protocol. Compared to other parallel or distributed systems, NUMA systems are tightly coupled with fast communication links. Both, bandwidth and latency, of the links that connect different CPUs have greatly improved in the last few years. However, applications with high performance needs may still benefit from careful memory and thread placement and optimized memory locality.

Database management systems in data centers are increasingly often deployed on NUMA systems where they serve different use-cases: single large applications as well as many small applications that are consolidated on one machine. The large amount of memory combined with the intensive use of compression techniques [WKHM00, ZHNB06] lead to structured data that often completely fit in memory. Therefore, main-memory database systems will gradually outnumber disk-based database systems. With more and more data in memory, there often is no need to perform expensive I/O operations to execute a query, which in turn reduces the relevance of traditional buffering mechanisms in the DBMS (buffer pools). Hence, the design and performance optimization focus shifts from disk-centric, with I/O having been the main bottleneck for decades, to memory-centric with new challenging research topics like optimal memory layout and access.

Database servers often show a natural partitioning in the data that is accessed, caused by, e.g., (a) multi-database operation, (b) private schema multi-tenancy, or (c) business or application requirements. A single database server can host different databases (a) for one or multiple applications. This can be used to allow an application access to all the databases it needs on a single machine or to introduce multi-tenancy and hence let applications with moderate performance requirements share resources [CJMB11, KL11]. Resource consolidation also motivates (b): private schema multi-tenancy. Here, different applications share a single database but operate on private tables. Some Database-as-a-Service providers use this scheme to provide scalable solutions, e.g., Microsoft SQL Azure [BCD+11]. The third case of data partitioning (c) occurs when a single large database stores data of different parts of the same business. Although, e.g., marketing and sales tables are stored in the same database, it is likely that they are never accessed together in one transaction. All cases of database partitioning have in common that parts of the database system (databases, schemas, or sets of tables) are accessed independently of one another. Transactions that spread across multiple data partitions are either impossible (private databases) or at most rare.

Knowledge about data partitions can be used to allocate a database's memory on the different CPUs' memories accordingly. This strategy can increase memory locality and reduce expensive communication between CPUs. Our goal is to evaluate the potential of thread/memory placement strategies in a set of experiments. We quantify the performance improvement achieved by co-locating threads and the memory they access. Additionally, we show that grouping database worker threads that access the same data on one CPU improves cache behavior, which in turn improves query performance. We use a thorough synthetic benchmark as well as a real database benchmark based on the TPC-H schema, executed on the MySQL database management system to verify our hypotheses. We also give an outlook on how our findings can be used to improve future DBMS performance on NUMA systems.

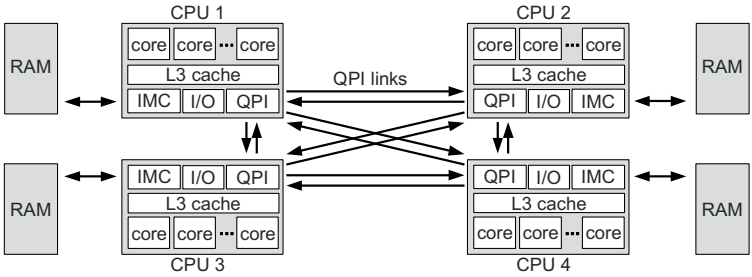


Figure 1: Overview of a NUMA system (e.g., Intel Westmere Architecture)

2 Preliminaries on NUMA Architecture

This section gives an overview on NUMA systems, main characteristics, and methods to influence application behavior on NUMA systems.

A NUMA system consists of multiple CPUs (also called sockets or nodes) that are connected via point-to-point connections. Each CPU has its own *local memory* that can be accessed with a lower latency and a higher bandwidth compared to the memories of the other CPUs (*remote memory*). Depending on the CPU vendor, the nodes are connected using different standards: Intel systems use QuickPath Interconnect[1] (QPI) whereas AMD systems use HyperTransport[2]. Figure 1 illustrates an Intel NUMA system with four sockets. Each of the CPUs has its own RAM that is connected via an *integrated memory controller* (IMC). The cores of a CPU each have a dedicated L1- and L2-cache (not shown); the L3-cache (also *last level cache* or *LLC*) is usually shared among all cores of a CPU. Each CPU has four outgoing and incoming QPI links, of which three are connected to other CPUs. Hence, four CPUs can be fully connected, i.e., each CPU can access any remote memory with only a single hop. Larger systems are usually not fully connected.

All currently available NUMA systems ensure cache coherency to keep the caches of the participating CPUs consistent (sometimes denoted as ccNUMA systems). There are several cache-coherence protocols like MESI, MOESI, and MESIF. QPI relies on the MESIF protocol, which adds a fifth state (*Forward*) to the MESI protocol, while HyperTransport uses MOESI with (*Owner*) being the fifth state.

By default, the operating system scheduler places threads based on the utilization of the CPUs. More precisely, a thread is created on the CPU that has the lowest CPU usage. For this reason, it is common that the threads of a single process are spread over all CPUs of a NUMA system. The scheduler can also place threads (possibly on a different CPU) after they were interrupted or sleeping. Moving a thread to another CPU can be quite expensive because the thread's cache state has

[1] `http://www.intel.de/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html`

[2] `http://www.hypertransport.org/docs/uploads/HT_General_Overview.pdf`

to be also moved and the thread's memory may not be local anymore. Therefore, the linux scheduler has a concept of scheduling domains that model the memory hierarchy and that reduce the likelihood of threads to migrate between nodes. To further control the risk of thread migrations, programming languages provide functionality to forbid the scheduler to move threads, i.e., threads can be bound to a specific CPU or even a specific core. Linux `sched_setaffinity` or functions of linux `libnuma` allow to do that. The linux tool `numactl` can also be used to force application-to-node bindings for all threads of an application. However, binding threads to specific CPUs or cores limits the scheduler's opportunities for balancing the load. This can lead to worse performance when some CPUs are overloaded while others are underutilized. For this reason, explicitly binding threads to single CPUs is a method that should be used carefully and only with the necessary application and context knowledge.

Besides thread placement, data placement is the second important aspect to consider on NUMA architectures. Naturally, data should be located close to the CPU that accesses it frequently. The default data placement policy of linux is called *first touch*. Newly allocated memory is placed local to the thread that actually uses (touches) it for the first time. This policy is especially advantageous when a single thread allocates large amounts of memory for multiple worker threads that are eventually executed on different nodes. The `libnuma` library provides functionality to directly specify a set of CPUs on which a thread allocates memory. Similar to the placement of threads, the `numactl` tool can be used for the placement of memory for an entire application. It provides standard placement policies like *preferred* allocation on a single CPU or *interleaved* and *local* allocation on a defined set of CPUs. Interleaved allocation allocates memory in a round-robin manner on the CPUs while local allocation is similar to the first touch strategy but with a restricted set of CPUs.

3 Synthetic Memory-Access Benchmark

To better understand the effects of the NUMA architecture on a DBMSs' query performance, we have first designed and implemented a synthetic benchmark to measure memory accesses in different situations. The benchmark's intention is to mimic a database system's memory access behavior. Therefore, our benchmark is positioned between low-level benchmarks like in [MHSM09] and full-scale application benchmarks. The results of our benchmark tool for simple setups are consistent with established memory access benchmarks. By measuring our benchmark's execution times and monitoring the operating system as well as hardware performance counters, we are able to identify, quantify, and explain effects that memory and thread placement have on the performance. As our experiments show, especially cache sharing (sharing of LLCs among readers) and cache migrations (pulling the content of the LLC to another socket after a thread has migrated) have major impacts on performance.

3.1 Benchmark Setup and Execution

The benchmark is based on a tool, developed by us and written in C++. We have made our code available to other researchers.[3] In the tool, threads access arrays of configurable size by repeatedly reading or writing random entries. A benchmark run consists of either reads or writes, mixed workloads are not supported. Multiple threads can share access to an array to simulate multiple clients that read or write the same data. The `libnuma` library is used to control where threads are executed and on which socket they allocate memory. It is also used to force threads to migrate to another socket during the test run.

The metric of the benchmark is latency, i.e., how long it takes to execute a random read/write operation. During the benchmark, we use a modified version of the Intel Performance Counter Monitor[4] (PCM) to count and log certain events like LLC hits and misses or the number of packets sent over any QPI link. All presented results are measured on a 4-Socket Intel Westmere EX machine (see Table 2 in Section 4 on page 10 for details on the *Intel machine*). For all tests, we report the average access time of four billion accesses.

3.2 Benchmark Configuration: Custom Experiments

We use our benchmark tool with different configurations to measure remote memory access costs, thread migration costs, and to isolate cache effects. All results are summarized in Table 1.

Remote Memory Access Costs (Experiment 1) In a first experiment, we compare random access latency to local and remote memory. We therefore configure our tool to spawn a single thread that allocates an array of 128MBs. The size of the array is chosen to be larger than the LLC of our machine so that we observe and measure memory access and QPI link utilization. The working thread repeatedly reads (respectively writes) random bytes in the array that is either allocated in local memory or on a remote socket. Results are shown in Table 1. It can be seen that executing the workload with remote memory access takes about 1.18 times longer compared to local access (26 nanoseconds for local access versus 30.8 nanoseconds for remote access).

Thread Migration Costs (Experiment 2) The second experiment evaluates the costs for migrating a thread from one socket to another one. The benchmark tool spawns a single thread that is either executed on one socket for the whole benchmark or migrated to another socket frequently. We investigate two different cases: 16MBs and 128MBs of memory that are accessed. In the first case, the whole

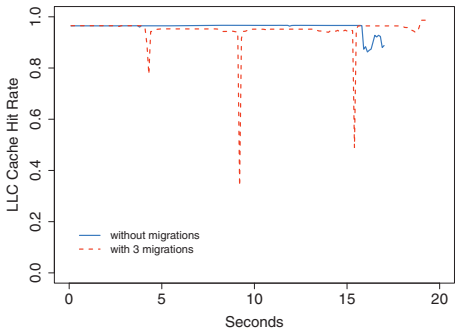
[3] `http://wwwdb.inf.tu-dresden.de/misc/user/kiefer/numa_benchmark.zip`
[4] `http://software.intel.com/en-us/articles/intel-performance-counter-monitor/`

array fits in the LLC of our machine. Hence, after the thread has migrated to each socket once, all sockets contain the data in their LLCs. Any further migration only costs a context switch and no consecutive remote memory access (we verify that with performance counters for LLC hits, which show a hit rate of almost 100% after 3 migrations). In the second case (128MBs), the memory does not fit in cache and hence each migration to a socket other than the first socket leads to consecutive remote memory accesses. The results in Table 1 show the isolated context migration costs for a small dataset (11.2 seconds versus 14.6 seconds for reads). For the large dataset, costs for remote memory accesses and costs for context switches overlay. The results show execution times comparable to Experiment 1, which suggests that for larger datasets, remote access costs dominate costs for context switches.

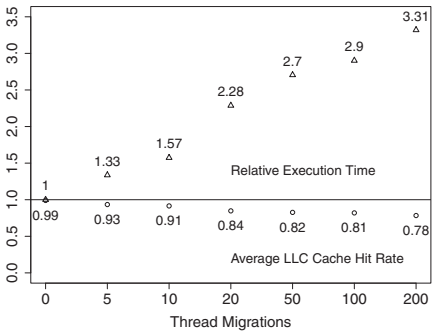
Cache Migration Costs (Experiment 3) The third experiment is intended to measure the costs for repeated migration of a thread followed by loading the data into the local last level cache. In contrast to the second experiment, a thread that migrates to a socket does not find any data in the last level cache (from a possible previous execution on that socket) because the data was evicted from the cache by

Experiment	Setup	Access Times in Nanoseconds	
		Local Access	Remote Access
Experiment 1	read 128MB	26.0	30.8
	write 128MB	26.0	31.0
		w/o Migrations	100 Migrations
Experiment 2	read 16MB	11.2	14.6
	write 16MB	12.5	15.2
	read 128MB	25.7	31.6
	write 128MB	26.5	32.3
		w/o Migrations	100 Migrations
Experiment 3	4 threads read 16MB	11.1	31.8
	4 threads write 16MB	11.4	27.0
	4 threads read 128MB	26.4	40.9
	4 threads write 128MB	27.5	50.3
		Co-located	Round-robin
Experiment 4	4 groups read 16MB	10.8	26.3
	4 groups write 16MB	11.0	21.3
	4 groups read 128MB	30.0	36.2
	4 groups write 128MB	43.5	47.5

Table 1: Synthetic benchmark results



(a) LLC hit rate with and without thread migrations



(b) Average LLC hit rate and relative execution times (normalized to execution without migrations)

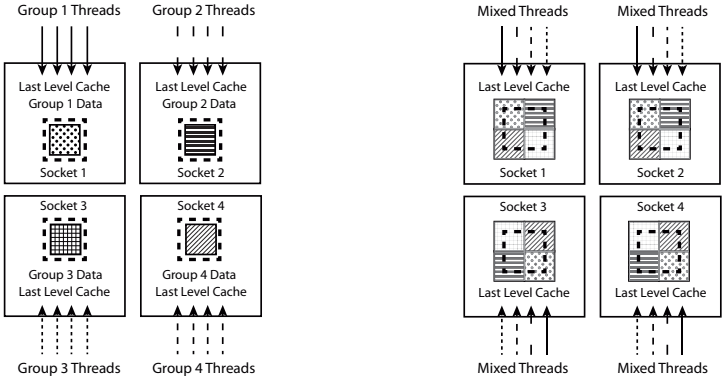
Figure 2: LLC rates for experiments with context switches

another thread in the meanwhile. The third experiment simulates the worst case costs for a thread migration.

To measure this effect, our benchmark tool spawns four threads, each reading randomly from an array of 16MBs. Without migration, each thread can fit the whole array in the local LLC and therefore read quickly from it. With thread migrations, a thread that migrates has to pull the data it reads to the socket where it migrated to. Doing so, this thread also evicts all the data that the previous thread on this socket held in cache. Hence, each migration comprises costs for the context switch and the following re-load of the last level cache. The results in Table 1 show that frequent migrations lead to about 2.8 times higher execution times compared to the experiment without migrations. The results in the table also show that for larger datasets (128MBs), the penalty for frequent thread migrations and the following re-load of the LLC is lower because most of the data does not fit in the LLC and needs to be read from memory.

We confirmed the described LLC behavior with measurements from the Intel PCM. They show that with migrations, the LLC hit rate drops for an instant after each migration because the thread has to pull the data from memory to the new LLC (shown in Figure 2a for only 3 migrations). With 200 migrations, the average hit ratio over the entire benchmark run drops to about 78% compared to 99% for the experiment without migrations. Figure 2b shows how the LLC hit rates decline while the execution times increase for a growing number of thread migrations.

Cache Concurrency Costs (Experiment 4) The fourth experiment shows and measures the effect of cache concurrency, which occurs when multiple threads access the same data but from different sockets. For this experiment, four instances of our benchmark tool are started in parallel. Each instance spawns four threads,



(a) Data layout with co-located threads (each LLC holds all data of one group)

(b) Data layout with distributed threads (each LLC holds one fourth of the data of all groups)

Figure 3: Cache concurrency effect for four groups of threads

which have access to the same array (comparable to multiple users accessing the same data in a database). We call the threads that share data a *thread group*.

Each thread in a group tries to load the data to the (thread-local) LLC. When all threads of a group are executed on the same socket, they beneficially share the LLC (see Figure 3a). When all threads are distributed across the sockets (see Figure 3b), they all try to load their respective data to the local LLC. This leads to constant cache line eviction of other groups' data and considerably lower cache hit rates. In the worst case scenario, where each group has a hot set of data that is about as large as a socket's last level cache, the effective LLC size is reduced by a factor as high as the number of sockets. The results in Table 1 show that for a 16MBs dataset, reading with distributed threads (round-robin) takes about 2.4 times longer than with co-located threads. The table also shows that for larger datasets, the effect is less important (reading only takes 1.2 times longer for 128MBs).

Our measurements with the Intel PCM verify the explained behavior. They show that the average LLC hit rate drops from about 98% when threads are co-located to as little as 33% with distributed threads (for the 16MBs experiment). They furthermore show that with distributed threads the data are replicated to all LLCs. With co-located threads, almost all cache hits (99%) are *unshared hits*, i.e., on cache lines in `modified` or `exclusive` state. With distributed threads on the other hand, cache line hits are mostly (94%) on cache lines that are either in `forward` or in `share` state and hence present in multiple LLCs.

3.3 Synthetic Benchmark Conclusion

Our results show that for certain memory access patterns such as multiple threads accessing the same data, which are common in modern database management systems, thread and memory placement can have a significant impact on performance. Our results also confirm the "common wisdom" that sharing of caches and memory with non-uniform access, although transparent to the software, can lead to performance benefits when instrumented properly or penalties when it is ignored.

Our expectation is that the demonstrated performance impact of thread and memory placement is also visible in a database management system that aims and is implemented for the highest-possible performance.

4 Impact of NUMA Architecture on DBMS Performance

In this section, we describe our experiments with a database management system on two different NUMA systems. Our goal is to transfer the results of our synthetic benchmark to the significantly more complex software of a DBMS. In our experiments, we concentrate on fixed placements of threads and memory and hence on remote access costs and cache concurrency effects. Incorporating thread migrations in a controllable fashion to also measure related effects was out of the scope of this paper and is considered future work. After detailing the experiment setup, we will show and interpret results from various runs of the chosen workload on both machines.

4.1 Experiment Setup

Metrics and Methodology We always report the average query throughput of multiple runs as the metric of our benchmarks. The MulTe database benchmark framework [KSL12] is used to execute multiple queries without think time in parallel and to measure the throughput of the database server. Additionally, we measure and log selected system parameters, e.g., hardware performance counters (Intel machine only, see next paragraph), to confirm a certain configuration or to help explaining certain effects. We are especially interested in core events like cache hits and misses and uncore events like QPI link utilization or events related to the cache coherence protocol.

Machines We use two different machines for our experiments, a 2-socket AMD architecture (*AMD machine*) and a 4-socket Intel machine (*Intel machine*). The machines were selected because they allow us to compare results for two different architectures and because they were readily available to us. An extended exper-

Table 2: Machines Used for Experiments

AMD machine	Intel machine
2x AMD Opteron (Istanbul)	4x Intel Xeon (Westmere EX)
2.6 GHz	2.13 GHz
6 cores per CPU	8 cores per CPU (2 HW threads)
16GB main memory per socket	32 GB main memory per socket
64KB L1, 512KB L2-cache per core	32KB L1, 256KB L2-cache per core
6MB LLC per socket	24MB LLC per socket
HyperTransport @ 9.6 GB/s per link	QPI @ 12.8 GB/s per link
SLES 11 (2.6.32.12-0)	Ubuntu 11.10 server (3.0.0-12)

iment with other machines, especially with more sockets, is planned as soon as respective machines are available. Table 2 summarizes both machines used for our experiments.

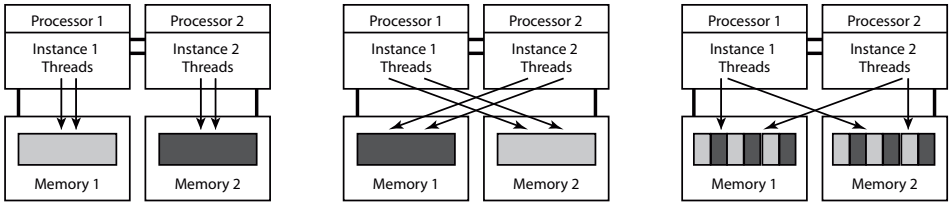
Database Management System All experiments are executed on the open-source DBMS MySQL Community Server (v5.5.17). The availability of the source code will allow us to implement future placement strategies in the DBMS (see Section 5). MySQL allows the use of different storage engines, including the MEMORY (HEAP) storage engine that holds all tables in memory. We use this in-memory engine (although it does not offer any persistence layer) because we think it is the best choice for our benchmark. It is a natural fit for data that resides in memory all the time and it scales better with the number of concurrent users compared to, e.g., the InnoDB engine.

Workload We base our experiments on the TPC-H database schema and workload [TPC12]. The scaling factor for the database is 1, i.e., 1GB of raw data. We use a subset of the official benchmark queries together with some synthetic queries that execute common data access patterns. All experiments are restricted to read-only queries.

4.2 Data Partitioning

Our experiments are motivated by the observation that in DBMS data partitions are accessed independently from one another. To mimic such partitions, we install multiple MySQL Server instances, e.g., one for each user group that accesses the data. Each instance contains a database with the TPC-H schema. To simplify our experiments, we use one instance per socket on each machine.

Each instance of MySQL runs as a separate process. Therefore, we are able to explicitly control thread placement and memory allocation on instance level, i.e.,



(a) Aligned setup (b) 1-ahead setup (c) Interleaved setup

Figure 4: Thread and memory assignment strategies

process level, with the `numactl` tool. To show the potential of the NUMA effects on database server performance, we compare three different setups with the *default* case: *aligned*, *1-ahead*, and *interleave*. The default case does not take the NUMA architecture into consideration (beyond what the operating system does). In the aligned setup—the expected best case—we bind each instance's threads explicitly to one socket and allocate all the memory on the same socket (shown in Figure 4a). In the 1-ahead setup—the expected worst case—we bind threads to one socket and allocate memory on the other one (AMD machine) or the next one[5] (Intel machine) (shown in Figure 4b). The third setup (*interleave*) binds threads to a fixed socket but interleaves memory on all sockets (shown in Figure 4c).

For each instance of MySQL, we use as many parallel connections to query the database as there are cores (AMD machine) or hardware threads (Intel machine). We compute the sum of the throughputs of all instances to get the overall system throughput. We measure a single query at a time (multiple executions of the same query with different parameter values) to minimize the side effects that may occur in a mixed workload.

4.3 Results and Interpretation

The results of our experiments are shown in Figure 5 for the AMD machine and in Figure 6 for the Intel machine. Both charts show the query throughputs of the aligned, 1-ahead, and interleave setups, normalized to the default case. Dotted lines separate TPC-H queries from the synthetic ones.

It can be seen in Figure 5 that on the AMD machine the aligned setup either is as fast as the default setup or outperforms it by as much as 18%. On average, aligning threads and memory leads to improvements between 10% and 15% which is notable, given the simple modification that is necessary to achieve the improvement. Also, one expects that the effects will be amplified in systems with more sockets. The

[5]Since all sockets are fully connected in our machine, the *next* socket, can be an arbitrary one. We use the current socket ID, incremented by one and modulo the number of sockets as the *next* socket.

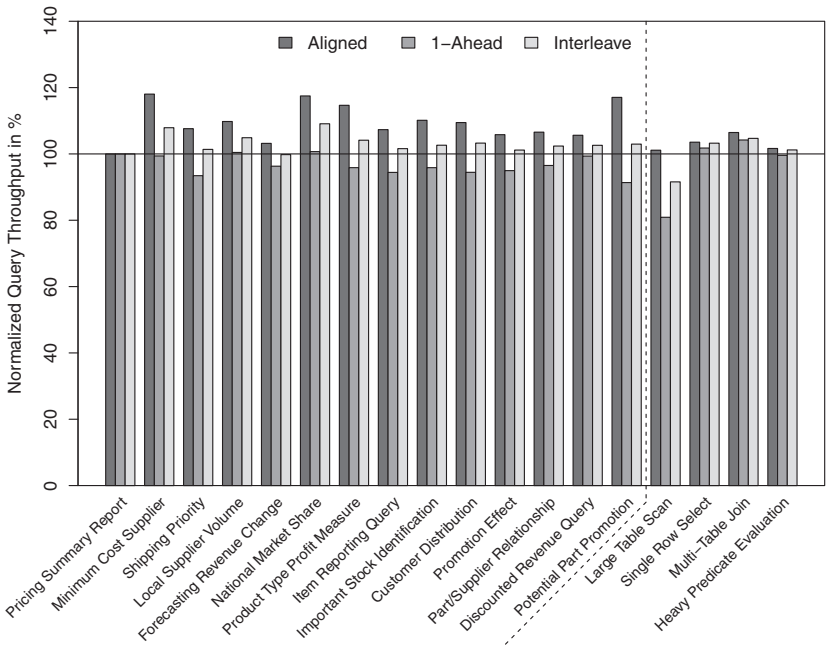


Figure 5: Normalized query throughput on the AMD machine

chart furthermore shows that placing threads and memory on different sockets (1-ahead) often leads to a degradation of the query throughput. This is to be expected, given the higher amount of communication. It is interesting to see that even the naive approach of interleaving the memory on all sockets leads to a slight improvement in some cases and no degradation for any of the queries. We account this to the threads of each instance being assigned to a fixed socket and the resulting improved LLC usage. We will revisit this effect further down when we analyze the results on the Intel machine.

Figure 6 shows the results for the Intel machine. The throughput improvement when aligning threads and memory can be as high as 75%. Interestingly, even the 1-ahead setup does not degrade query throughput, but slight improves it in most cases. The interleaved setup lies between aligned and 1-ahead. The result for 1-ahead is counter-intuitive, given the communication that is needed to access the data on the remote socket. We believe that improvements in LLC usage—caused by assigning each instances's threads to a fixed socket—lead to the better query throughput. The last result we take from both experiments is that only the first synthetic query stands out as it shows the worst performance degradation among all queries with the 1-ahead and interleaved setups. The other synthetic queries show average improvements. Finding better synthetic queries that show the connection between access pattern and performance improvements is subject to future work.

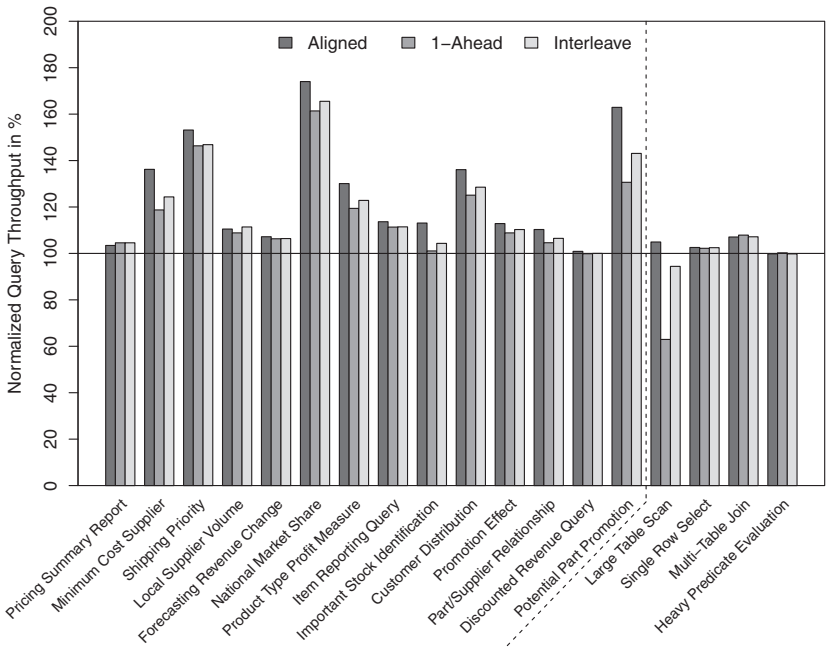
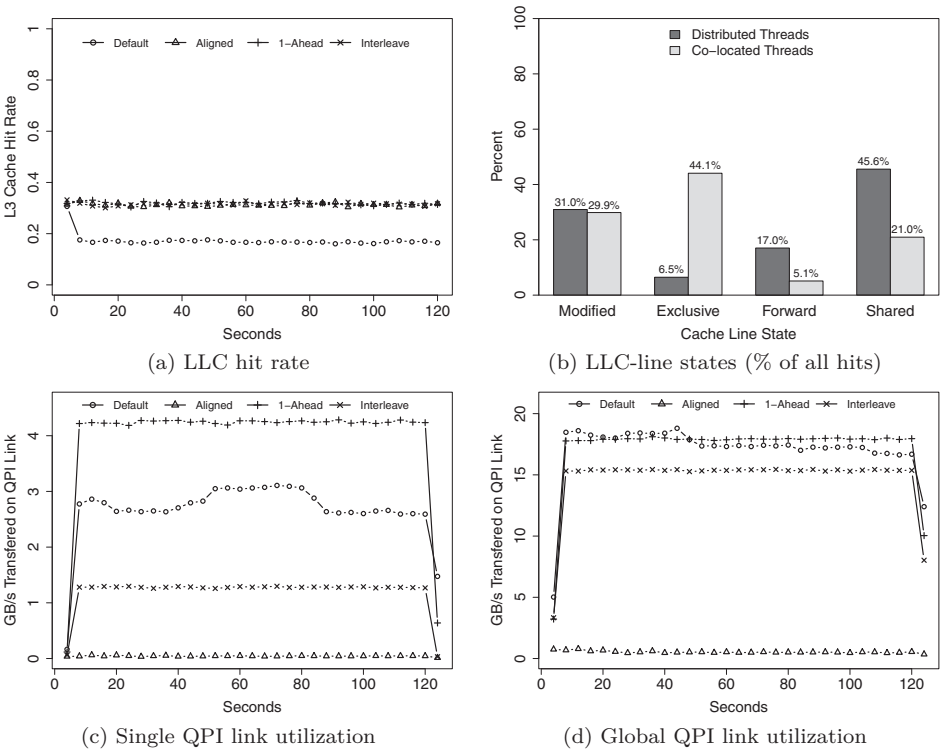


Figure 6: Normalized query throughput on the Intel machine

Cache Behavior: All three setups (aligned, 1-ahead, and interleave) have in common that all threads that access the same data are executed on the same socket. Hence, data that one thread reads is more likely to stay in the LLC for another thread that reads the same data. In the default case (without `numactl`), threads of all instances are executed on all sockets and even migrate between sockets from time to time (we have verified that with data monitored in the `/proc` pseudo-filesystem). This is comparable to the *Cache Concurrency Costs* experiment (Experiment 4) of our synthetic benchmark: multiple threads from one instance run on different sockets and therefore load the same data to their local LLCs. The effective size of the cache is considerably reduced due to data that is stored redundantly in several caches. To support this claim, we again monitored the LLC hit rate and cache line states for one query execution. Figures 7a and 7b show the respective results. The LLC hit rate for the default case is lower (about 20%) than for all other setups (about 35%). The second chart, shown in Figure 7b, confirms the cache concurrency. In the default case (distributed threads), memory accesses often hit shared cache lines, i.e., cache lines in *shared* or *forward* state (together 62.6%). There are fewer hits on unshared cache lines, i.e., in *exclusive* or *modified* state (together 37.4%). The setups with co-located threads (aligned, 1-ahead, and interleave all use `numactl` with `cpubind`) show significantly higher hit rates on unshared cache lines (together 74%) while *shared* and *forward* lines are hit only with a rate of 26%.



(a) LLC hit rate

(b) LLC-line states (% of all hits)

(c) Single QPI link utilization

(d) Global QPI link utilization

Figure 7: L3-cache usage and QPI link utilization

QPI Link Usage: All three setups (aligned, 1-ahead, and interleave) show the same LLC but still differ in the query throughput results, likely caused by differences in communication between sockets. We have analyzed the QPI link utilization and show the results for a single query and for a single QPI link (Figure 7c); and aggregated for all QPI links (Figure 7d). One can see that the throughput on a single QPI link follows the intuition from the given setups. The aligned setup does not need any inter-socket communication while the 1-ahead setup puts the most pressure on a single QPI link. The interleave setup lies somewhere in between. Interestingly, the default case that uses the first-touch policy to allocate memory needs more communication than the interleave setup. The aggregated QPI link utilization over all sockets and all links looks slightly different (Figure 7d). Interesting here, the default case causes about as much communication as the expected worst case (1-ahead). This and the improved LLC behavior can explain why on the Intel machine, even the 1-ahead setup is often slightly faster than the default setup.

5 Towards Database-Level Scheduling

Our experiments were able to show the potential of NUMA awareness in DBMSs which leads to many opportunities for future work. Control mechanisms for thread placement and memory allocation need to be implemented in the DBMS (instead of using `numactl`). Only there, the necessary knowledge about data partitions is available and only there it is possible to allocate threads and memory in a fine-grained fashion. Since DBMSs are usually deployed on dedicated machines, it is safe to assume that all important threads and memory consumers of the whole system are known to the DBMS. We have started to modify the MySQL DBMS to support thread placement based on the data that an application will access but detailed reports are subject to future work.

All data partitions (i.e., instances) were evenly queried in our experiments. A NUMA-aware DBMS will have to deal with skewed loads and dynamic changes in the query frequency of certain partitions. The workload used in the experiments contained only single queries (no mixed workload) of a certain type. Database workloads are traditionally classified as either being OLTP (online transaction processing) or OLAP (online analytical processing). OLTP workload on the one hand presents a high frequency of short queries that often access only a small amount of data. The performance of OLTP queries is dominated by the latency of the data access. OLAP queries on the other hand are usually less frequent. They access and aggregate large amounts of data and not seldom read whole tables in the process. Consequently, read operations are rather sequential and the performance is dominated by the available bandwidth. Recent trends in database management systems have weakened the separation of OLTP and OLAP systems and many systems are nowadays used to answer both types of queries. Hence, a NUMA-aware DBMS that takes care of thread and memory placement needs to take both query types and therefore very different access patterns into account.

Once a DBMS is equipped with tools to allocate memory and assign threads based on data partitioning and knowledge about the architecture, the next step will be to develop a cost model to evaluate thread and memory placement. Based on the cost model, the DBMS needs to decide where to execute threads and where to allocate memory. Using more resources on multiple sockets has to be weighted against having resources local to the execution.

Figure 8 shows a possible architecture for a NUMA-aware DBMS. The DBMS has the necessary partitioning information and sends thread scheduling and memory placement policies to the operating system. The operating system on the other hand provides information about the NUMA architecture (like the memory topology) and an interface to scheduling and memory placement decisions.

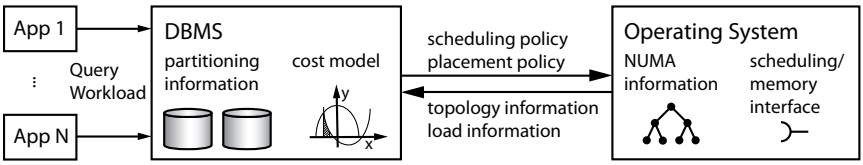


Figure 8: Architecture of a NUMA-aware DBMS

6 Related Work

Related work from different communities is relevant for the given topic. We distinguish work that concentrates on NUMA in general from papers that investigate DBMSs on modern hardware, possibly but not necessarily NUMA architectures.

6.1 Work related to NUMA architectures

The consequences of non-uniform memory access as well as thread-placement in multicore systems have been studied in the operating systems and high-performance computing communities for many years. Among the more recent results are Molka and Hackenberg et al. who have investigated the low-level memory performance and cache coherence effects at the granularity of single cache lines [HMN09, MHSM09]. McCurdy and Vetter give a general overview on finding and fixing NUMA-related performance problems [MV10] and Blagodurov et al. propose a NUMA-aware scheduler and user-level scheduling on NUMA systems, though not specific to a certain application [BZFD10, BF11]. The problem of mapping threads to cores (and possibly co-locating them) to reduce communication and increase data sharing has been investigated by, e.g., Tam et al. [TAS07] and Tang et al. [TMV+11].

6.2 Work related to DBMSs on modern hardware

There is little information available on NUMA awareness of commercial DBMSs. From blogs[6] and personal communication, we know that Microsoft SQL Server and the Oracle DBMS are NUMA-aware. In SQL Server, it is possible to direct user connections to certain nodes and the system automatically partitions buffer pools and prefers local memory for temporal and session data.

The research community has analyzed the impact of modern (multicore) processors on existing DBMSs [ADHW99, HPJ+07]. To overcome the general shortcomings of these systems, i.e., their poor scalability with the number of CPUs and cores,

[6] E.g., `http://blogs.msdn.com/b/slavao/archive/2005/08/02/446648.aspx` or `http://kevinclosson.wordpress.com/2009/05/14/you-buy-a-numa-system-oracle-says-disable-numa-what-gives-part-ii/`

different solutions have been proposed. All solutions can roughly be categorized as *distributed systems*, which treat multiple CPUs/cores as if they were a distributed system, and *shared systems*, which try to overcome scalability issues by reducing communication and contention.

Distributed system approaches Distributed database systems have been investigated first in the late 80s and early 90s [DGS+90, AvdBF+92]. Many aspects of these early distributed DBMSs can be used to improve scalability on modern systems. The Multimed system at ETH-Zürich [SSGA11] deploys multiple instances of an existing database engine (e.g., PostgreSQL or MySLQ) on non-overlapping subsets of all cores. A master database on one partition receives all updates and propagates them asynchronously to satellite databases, which in turn receive read-only load. H-Store [SMA+07] partitions the data horizontally over nodes and cores and executes transactions sequentially (single-threaded) on each core. Hence, there is no synchronization of access to each partition needed. Depending on what data a transaction accesses, multiple partitions need to communicate. The HyPer system [KN11] is an in-memory system that uses processor-inherent lazy copy-on-write and virtual memory management to support OLTP and parallel OLAP transactions. Multiple read-only queries can be executed at a time, but writing requests are executed sequentially to ensure consistency without synchronization. Also in the context of distributed (or shared-nothing) database systems are works like Schism by Curino et al. [CJZM10] that aim at reducing the need for inter-partition communication by means of workload-aware partitioning of the data.

Shared system approaches An example for a shared-everything system is the Shore-MT system by Johnson et al. [JPH+09]. After identifying bottlenecks in existing database storage managers, the authors develop a multithreaded, scalable storage manager by optimizing locks, latches, and synchronization and thereby reducing contention. Physiological partitioning, a compromise between the distributed and the shared approach, was proposed by Pandis and Tözün et al. [PTJA11, TPJA12]. While the data is still shared, a multi-rooted B+ tree is used to partition the data which avoids costly page-latches. The DORA-system, also by Pandis et al. [PJHA10], binds threads to disjoint sets of the data and decomposes transactions to smaller actions according to the data they access. Each thread has private locking mechanism to control access to the data it owns. A recent work by Albutiu et al. [AKN12] analyzes sort-merge joins in multi-core database systems. The authors recognize the NUMA characteristics of such systems and motivate their algorithm design with micro-benchmarks related to our synthetic benchmark.

A recent work that falls between the strictly shared or distributed approach and that also aims at demonstrating and utilizing the NUMA effects in a DMBS was published by Porobic et al. [PPB+12]. The authors perform a detailed analysis of different shared and distributed deployments in NUMA systems. To show the performance impact of the NUMA architecture on a DBMS, they use ShoreMT as a scalable storage manager and TPC-C as an OLTP workload.

7 Conclusion

We showed with our synthetic benchmark that remote memory access and cache usage related to thread placement and thread movement heavily influence performance on NUMA systems. We furthermore showed that database management systems—when executed on NUMA systems—can benefit from careful thread and memory placement. Executing an idealized workload showed the potential for throughput improvements of up to 75% compared to the naive execution that ignores the characteristics of the NUMA system. We had to experience that reliable experiments with a DBMS are extremely hard to conduct. MySQL, used as we did for our studies, suffers from different scalability and reliability issues so that we had to revise our experiments many times to isolate and quantify the NUMA effects. Nevertheless, our experiments have confirmed that not only the non-uniform main memory access is responsible for performance differences but also the fact that each node has its own cache hierarchy. This leads to disadvanteous cache evictions when threads migrate between nodes or when threads access the same data from different nodes.

References

[ADHW99] Anastasia Ailamaki, David J Dewitt, Mark D Hill, and David A Wood. DBMSs On A Modern Processor: Where Does Time Go? In *VLDB '99*, Edinburgh, Scottland, 1999. VLDB Endowment.

[AKN12] Martina-Cezara Albutiu, Alfons Kemper, and Thomas Neumann. Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems. In *VLDB '12*, volume 5, Istanbul, Turkey, 2012. VLDB Endowment.

[AvdBF+92] Peter MG Apers, Carel A van den Berg, Jan Flokstra, Paul WPJ Grefen, Martin L Kersten, and Annita N Wilschut. PRISMA/DB: A Parallel, Main Memory Relational DBMS. *Knowledge and Data Engineering*, 4(6):541–554, 1992.

[BCD+11] Philip A. Bernstein, Istvan Cseri, Nishant Dani, Nigel Ellis, Ajay Kalhan, Gopal Kakivaya, David B. Lomet, Ramesh Manne, Lev Novik, and Tomas Talius. Adapting Microsoft SQL Server for Cloud Computing. In *ICDE '11*, pages 1255–1263, Hannover, Germany, 2011. IEEE.

[BF11] Sergey Blagodurov and Alexandra Fedorova. User-level scheduling on NUMA multicore systems under Linux. In *Linux Symposium*, pages 81–91, Ottawa, Canada, 2011.

[BZFD10] Sergey Blagodurov, Sergey Zhuravlev, Alexandra Fedorova, and Mohammad Dashti. A Case for NUMA-aware Contention Management on Multicore Systems. In *PACT'10*, Vienna, Austria, 2010.

[CJMB11] Carlo Curino, Evan P.C. Jones, Sam Madden, and Hari Balakrishnan. Workload-Aware Database Monitoring and Consolidation. In *SIGMOD '11*, pages 313–324, Athens, Greece, 2011. ACM.

[CJZM10] Carlo Curino, Evan Jones, Y. Zhang, and Sam Madden. Schism: a Workload-Driven Approach to Database Replication and Partitioning. In *VLDB '10*, volume 3, pages 48–57, Singapore, China, 2010. VLDB Endowment.

[DGS+90] David J Dewitt, Shahram Ghandeharizadeh, Donovan Schneider, Allen Bricker, Hui-I Hsiao, and Rick Rasmussen. The Gamma database machine project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, March 1990.

[HMN09] Daniel Hackenberg, Daniel Molka, and Wolfgang E. Nagel. Comparing Cache Architectures and Coherency Protocols on x86-64 Multicore SMP Systems. In *MICRO '09*, pages 413–422, New York, New York, USA, 2009.

[HPJ+07] Nikos Hardavellas, Ippokratis Pandis, Ryan Johnson, Naju G Mancheril, Anastasia Ailamaki, and Babak Falsafi. Database Servers on Chip Multiprocessors: Limitations and Opportunities. In *CIDR '07*, pages 79–87, Asilomar, California, USA, 2007.

[JPH+09] Ryan Johnson, Ippokratis Pandis, Nikos Hardavellas, Anastasia Ailamaki, and Babak Falsafi. Shore-MT: A Scalable Storage Manager for the Multicore Era. In *EDBT '09*, pages 24–35, Saint Petersburg, Russia, 2009. ACM.

[KL11] Tim Kiefer and Wolfgang Lehner. Private Table Database Virtualization for DBaaS. In *UCC '11*, volume 1, pages 328–329, Melbourne, Australia, December 2011. IEEE.

[KN11] Alfons Kemper and Thomas Neumann. HyPer: A Hybrid OLTP&OLAP Main Memory Database System Based on Virtual Memory Snapshots. In *ICDE '11*, pages 195–206, Hannover, Germany, 2011. IEEE.

[KSL12] Tim Kiefer, Benjamin Schlegel, and Wolfgang Lehner. MulTe: A Multi-Tenancy Database Benchmark Framework. In *TPCTC '12*, Istanbul, Turkey, 2012.

[MHSM09] Daniel Molka, Daniel Hackenberg, Robert Schöne, and Matthias S. Müller. Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System. In *PACT'09*, pages 261–270, Raleigh, North Carolina, USA, September 2009. IEEE.

[MV10] Collin McCurdy and Jeffrey Vetter. Memphis: Finding and Fixing NUMA-related Performance Problems on Multi-core Platforms. In *ISPASS'10*, pages 87–96, White Plains, NY, USA, 2010. IEEE.

[PJHA10] Ippokratis Pandis, Ryan Johnson, Nikos Hardavellas, and Anastasia Ailamaki. Data-Oriented Transaction Execution. In *VLDB '10*, volume 3, Singapore, China, 2010. VLDB Endowment.

[PPB+12] Danica Porobic, Ippokratis Pandis, Miguel Branco, Pinar Tözün, and Anastasia Ailamaki. OLTP on Hardware Islands. In *VLDB '12*, pages 1447–1458, Istanbul, Turkey, 2012. VLDB Endowment.

[PTJA11] Ippokratis Pandis, Pinar Tözün, Ryan Johnson, and Anastasia Ailamaki. PLP: Page Latch-free Shared-everything OLTP. In *VLDB '11*, Seattle, Washington, USA, 2011. VLDB Endowment.

[SMA+07] Michael Stonebraker, Sam Madden, Daniel J Abadi, Stavros Harizopoulos, Nabil Hachem, and Pat Helland. The End of an Architectural Era (It's Time for a Complete Rewrite). In *VLDB '07*, Vienna, Austria, 2007. VLDB Endowment.

[SSGA11] Tudor-Ioan Salomie, Ionut Emanuel Subasu, Jana Giceva, and Gustavo Alonso. Database Engines on Multicores, Why Parallelize When You Can Distribute? In *EuroSys '11*, page 14, Salzburg, Austria, 2011. ACM Press.

[TAS07] David Tam, Reza Azimi, and Michael Stumm. Thread Clustering : Sharing-Aware Scheduling on SMP-CMP-SMT Multiprocessors. In *EuroSys '07*, Lisboa, Portugal, 2007.

[TMV+11] Lingjia Tang, Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. The Impact of Memory Subsystem Resource Sharing on Datacenter Applications. In *ISCA '11*, San Jose, California, USA, 2011.

[TPC12] TPC. Transaction Processing Performance Council, TPC-H, 2012.

[TPJA12] Pinar Tözün, Ippokratis Pandis, Ryan Johnson, and Anastasia Ailamaki. Scalable and dynamically balanced shared-everything OLTP with physiological partitioning. *The VLDB Journal*, June 2012.

[WKHM00] Till Westmann, Donald Kossmann, Sven Helmer, and Guido Moerkotte. The Implementation and Performance of Compressed Databases. In *SIGMOD '00*, pages 55–67, New York, NY, USA, 2000. ACM.

[ZHNB06] Marcin Zukowski, Sándor Héman, Niels Nes, and Peter A. Boncz. Super-Scalar RAM-CPU Cache Compression. In *ICDE '06*, page 59, Washington, DC, USA, 2006. IEEE.

Fully Parallel Inference in Markov Logic Networks

Kaustubh Beedkar, Luciano Del Corro, Rainer Gemulla

Max-Planck-Institut für Informatik
66123 Saarbrücken
{kbeedkar,corrogg,rgemulla}@mpi-inf.mpg.de

Abstract: Markov logic is a powerful tool for handling the uncertainty that arises in real-world structured data; it has been applied successfully to a number of data management problems. In practice, the resulting ground Markov logic networks can get very large, which poses challenges to scalable inference. In this paper, we present the first fully parallelized approach to inference in Markov logic networks. Inference decomposes into a grounding step and a probabilistic inference step, both of which can be cost-intensive. We propose a parallel grounding algorithm that partitions the Markov logic network based on its corresponding join graph; each partition is ground independently and in parallel. Our partitioning scheme is based on importance sampling, which we use for parallel probabilistic inference, and is also well-suited to other, more efficient parallel inference techniques. Preliminary experiments suggest that significant speedup can be gained by parallelizing both grounding and probabilistic inference.

1 Introduction

Real-world data is often inconsistent, noisy, or incomplete. Markov logic [RD06] is a recent, promising approach to handle such uncertainty in structured data. It has been employed successfully in a number of applications, including link prediction [RD06], entity resolution [SD06], information extraction [PD07], and ontology learning [PD10]. At its heart, Markov logic bridges the gap between first-order logic and probability theory: The former allows to encode and reason about deterministic information, the latter is well-suited to manage uncertainty.

A *Markov logic network* (MLN) is a set of first-order logic formulas called *rules*; each rule is associated with a numerical *weight*. For example, suppose that we are given a university database in which the `advisedBy` relation between students and professors is incomplete. MLNs allow us to use rules such as "if a student and a professor have a joint publication, then the student is advised by the professor" to approach this link prediction problem. This rule appears helpful but is inherently uncertain, i.e., it is true in many but not all cases. MLNs thus attach a weight (say, 2.5) to this rule; the weight is related to the confidence that instances of the rule are true. Formally, we obtain

$$2.5: \forall s.\forall p.\forall t.\, \texttt{student}(s) \wedge \texttt{professor}(p) \wedge \texttt{authorOf}(p,t) \wedge \texttt{authorOf}(s,t) \implies \texttt{advisedby}(s,p).$$

Inference in Markov logic is performed by grounding the network using an *evidence database* of known facts and a set of constants. The output of this grounding step is a Markov network, on which probabilistic inference is performed subsequently. The evidence database and, even more so, the ground network can be very large. A (somewhat naive) grounding of the above rule with 1000 students, 10 professors, and 100 publications results in one million instances, connecting tens of thousands of variables (such as `advisedBy(Anna, Bob)`). In real-world applications, in which the involved datasets can be much larger, both grounding and probabilistic inference pose severe challenges to the scalability of Markov logic.

In this paper, we propose a fully parallel approach to scalable inference in Markov logic. Most prior work has focused on efficient grounding methods [SN09, MR10, NRDS11] or parallel probabilistic inference [GLGG11]. In contrast, our approach is holistic in that we parallelize both grounding and probabilistic inference. In more detail, we develop a simple yet effective technique to partition an MLN in such a way that each partition can be ground independently and in parallel (using a grounding method of choice). In addition to reducing grounding time, the resulting ground network is readily partitioned and well-suited for parallel probabilistic inference (again, using a method of choice). Our approach thus completely avoids expensive network partitioning and data redistribution steps after grounding.

The contributions of this paper are as follows: (1) We propose a framework for fully parallel inference in Markov logic networks. (2) We derive and analyze a parallel inference algorithm based on importance sampling. This algorithm may not be the best-performing choice in practice, but it provides valuable insight into how to obtain a good partitioning of a ground Markov logic network over a set of compute nodes. (3) We develop and analyze a practical algorithm for partitioning a ground Markov network based on minimum graph cuts. (4) Based on the insights obtained by our analysis, we propose a novel partitioning scheme for Markov logic networks. In contrast to prior work, we partition the network before we ground it. This approach avoids the need for partitioning and redistributing the ground network. (5) We present results of a preliminary experimental study on real-world data. Our results suggest that significant speedup can be gained by both parallelizing grounding and parallelizing probabilistic inference.

2 A Primer On Markov Logic Networks

Recall that a Markov logic network is a set of weighted rules (first-order logic formulas). Fig. 1 displays an excerpt of practical MLN used for predicting the `advisedBy` relation [RD06]; the example is very simple for expository reasons. The first rule states that advisees must be students and the second rule states that advisors must be professors or senior researchers. The higher weight of the second rule indicates that its instances are more likely to be satisfied than instances of the first rule.

To understand the semantics of a Markov logic network, we need to ground the network using a specific set of *constants*, i.e., students and professors. The

Rule	Weight	Formula
1	1.7	$\forall s.\forall p.\ \neg\texttt{student}(s) \implies \neg\texttt{advisedBy}(s,p)$
2	2.5	$\forall s.\forall p.\ \texttt{advisedBy}(s,p) \implies \texttt{hasPosition}(p,\texttt{Professor})$ $\vee\,\texttt{hasPosition}(p,\texttt{Senior Researcher})$

Figure 1: Excerpt of a Markov logic network for predicting the `advisedBy` relation

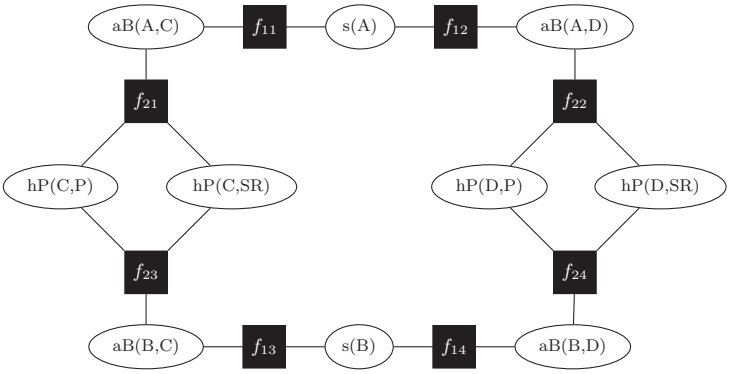


Figure 2: Factor graph representation of a ground MLN

output of the grounding process can be represented by a factor graph. A *factor graph* is a bipartite graph in which nodes correspond to either Boolean random variables (ovals) or factors (boxes). Each *Boolean variable* represents a grounding of an atom that occurs in an MLN rule. For example, suppose that Anna and Bob are students (domain of s) and that Charles and Debbie are professors (domain of p). We obtain the 10 Boolean variables shown in Fig. 2, e.g., $\texttt{s}(\texttt{A})$ and $\texttt{aB}(\texttt{A},\texttt{C})$. Informally, *factors* represent groundings of MLN rules; see [RD06] for a formal definition. In our example, there are 8 factors; the two variables mentioned above are connected by factor f_{11}. Each factor can be seen as a function over the variables that it is connected to. Given a valuation of these variables, the factor outputs the (exponential of the) weight of its corresponding rule if satisfied under the valuation; otherwise the factor outputs 1. We obtain

$$f_{11}(\texttt{s}(\texttt{A}),\texttt{aB}(\texttt{A},\texttt{C})) = \begin{cases} e^{1.7} & \text{if } \neg\texttt{s}(\texttt{A}) \implies \neg\texttt{aB}(\texttt{A},\texttt{C}), \\ 1 & \text{otherwise.} \end{cases}$$

Note that the exponentiation here implies that the value of a factor is always positive. Negative weights correspond to factor values less than one (rules that are "usually" wrong), zero weights correspond to factor value one (no influence), and positive weights correspond to factor values larger than one (usually true).

If the truth value of a ground atom is known from the evidence database, we clamp the value of the corresponding Boolean variable appropriately. These *evidence variables* form the "data" for inferring statistics of the unknown variables. If a predicate is ground under *closed-world semantics*, we additionally clamp the

values of ground atoms that do not occur in the evidence database to false; such a grounding semantic is useful for predicates that are completely known. Note that the factor graph can be simplified by eliminating evidence variables; this simplification is usually performed directly during grounding for efficiency reasons. Under the alternate *open-world semantics*, the value of the ground atoms that do not occur in the database are not clamped. In our ongoing example, we use the open-world semantics and an empty evidence database for simplicity.

After grounding, we perform probabilistic inference to make statements about the probability distribution of the unknown Boolean variables. We refer to predicates that we are ultimately interested in as *query predicates* (here `advisedBy`); the corresponding Boolean variables are referred to as *query variables*. The factor graph defines a probability distribution that assigns a probability $P(\boldsymbol{x})$ to each world $\boldsymbol{x}$, i.e., to each distinct assignment of values to the unknown variables $\boldsymbol{X}$:

$$P(\boldsymbol{X} = \boldsymbol{x}) = \frac{1}{Z_P} \prod_{f \in F} f(\boldsymbol{x}_f). \tag{1}$$

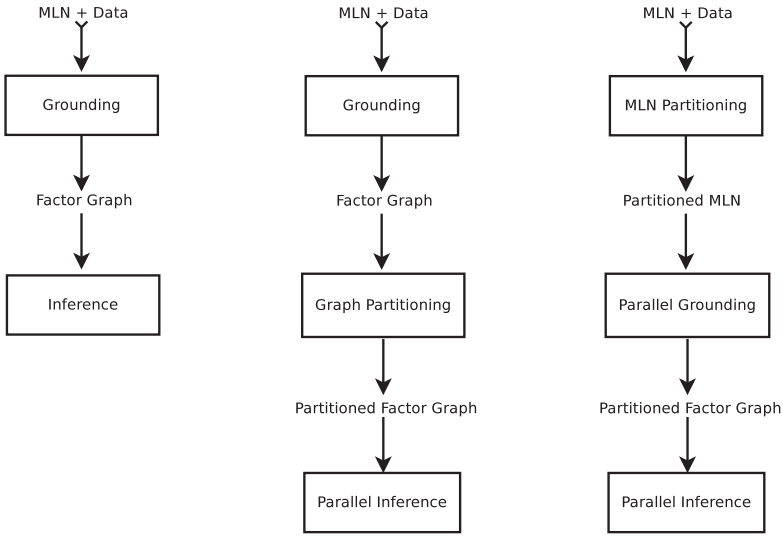
Here Z_P is a normalization constant that ensures summation to one, F denotes the set of all factors, and $\boldsymbol{x}_f$ denotes the values in $\boldsymbol{x}$ of the variables connected to factor f. Observe that the probability of a given world depends on both the number and the weights of the ground rules that are satisfied. If we increase the weight of a specific rule, then worlds that satisfy that rule become more likely while other worlds become less likely. In this paper, we focus on *marginal inference*, i.e., we want to infer the marginal distribution of each query variable (e.g., the probability that $\texttt{aB}(\texttt{A}, \texttt{B})$ is true).

The meaningfulness of the result of probabilistic inference depends on the information captured in the MLN. In general, rules are created by domain experts or learned from training data. Weight assignment is very difficult for humans because different formulas correlate with each other; weights are thus almost always learned from training data. In this paper, we assume that we are given an already learned MLN. Nevertheless, our methods and techniques are also helpful for scaling up the learning process, which makes repeated use of an inference component.

A thorough and accessible treatment of Markov logic can be found in [RD06]; factor graphs and techniques for probabilistic inference are discussed in [KF09].

3 Related Work

A well-known implementation of Markov logic is Alchemy [KSRD05], which includes both learning and inference components. Inference is performed as described above: the MLN is grounded using the evidence database and a probabilistic inference algorithm is run on the resulting factor graph. This traditional approach is illustrated in Figure 3(a). A number of techniques have been proposed to speed up inference, including clustering of query literals [MR10], reducing the size of the ground network [SN09], incremental grounding [Rie08], in-database grounding [NRDS11], and task-specific probabilistic MAP inference [NZRS11]. In this



(a) Sequential (b) Partly parallel (c) Fully parallel

Figure 3: Comparison of approaches to parallel inference in Markov logic

paper, we propose a fully parallel framework for speeding up inference in Markov logic. Since our framework is oblivious to the actual grounding algorithm and (to a lesser extent) to the probabilistic inference algorithms being run at worker nodes, most of the techniques mentioned above still apply.

Parallel probabilistic inference has received a lot of attention in the literature [DVKMG09, ASW08, NASW07, NRDS11, GLGG11]. In the context of Markov logic, Tuffy [NRDS11] implements a partly parallel approach to inference. In contrast to the sequential approach taken by Alchemy, Tuffy runs a graph partitioning algorithm on the factor graph obtained by the grounding algorithms, and subsequently runs a simple parallel algorithm for probabilistic inference on the partitions; see Figure 3(b). Since Tuffy also uses efficient in-database grounding, it is highly efficient and scalable. A potential bottleneck of Tuffy is the graph partitioning step; finding a minimum-cost balanced partition is NP-hard even for quite simple MLNs [NRDS11]. In fact, Tuffy resorts to a simple heuristic algorithm because even state-of-the-art graph partitioners are too expensive in practice.

Our work is inspired by Tuffy but also parallelizes the grounding and graph partitioning steps. As shown in Fig. 3(c), we partition the Markov logic network *before grounding*. A key advantage of this approach is that the expensive graph partitioning step is performed on the small MLN (data independent) instead of on the large factor graph (data dependent). Each partition of the resulting partitioned MLN is ground independently and in parallel; the output of the grounding step is thus a readily partitioned factor graph. Our MLN partitioning is designed such that the partitioned factor graph is suitable for parallel probabilistic inference; our experiments suggest that this approach can outperform state-of-the-art graph

partitioners in both speed and quality.

4 Parallel Probabilistic Inference

In general, exact probabilistic inference in ground Markov logic networks is intractable so that all existing implementations use some form of approximate inference; e.g., local search, belief propagation, or Markov Chain Monte Carlo (MCMC) sampling. In large applications, where factor graphs consist of millions of variables and factors, even approximate inference can be prohibitively expensive.

In this section, we show how to parallelize an arbitrary MCMC sampling technique via importance sampling. Our main objective is to gain understanding in how to partition a factor graph in a way that allows for efficient parallel inference. Our results are the basis for the MLN partitioning methods described in Sec. 5. Note that, in general, the MLN partitioning method needs to be designed for the specific probabilistic inference algorithm being used. As discussed later on, our framework is flexible enough to allow for a number of existing, potentially more efficient parallel inference methods [GLGG11].

4.1 Importance Sampling

In general, MCMC sampling techniques [KF09] explore the probability distribution of $\boldsymbol{X}$ by analyzing a subset $\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(n)}$ of the possible worlds; these subsets are referred to as *samples*. MCMC methods differ in how these samples are generated; in general, these methods construct a Markov chain that consists of one state for each possible world and has a stationary distribution of (exactly or approximately) $P(\boldsymbol{X})$. Samples from the Markov chain are generally dependent, but the dependencies decrease the further apart the samples are taken. We thus assume that $\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(n)}$ are independent samples from $\boldsymbol{X}$.

Given samples $\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(n)}$, we can estimate the expected value of any function $h(\boldsymbol{X})$ over the possible worlds as follows:

$$\mu = \mathrm{E}\,[\,h(\mathbf{X})\,] \approx \frac{1}{n}\sum_{i=1}^{n} h(\boldsymbol{X}^{(i)}) = \hat{\mu}.$$

In our setting of marginal estimation, we use functions of form $h(\boldsymbol{X}) = I_X$ for some query variable $X \in \boldsymbol{X}$. Here, indicator I_X takes value 1 if X is true; otherwise it takes value 0. For example, if $h(\boldsymbol{X}) = I_{\mathtt{aB(A,C)}}$, then μ is equal to the probability that Anna is advised by Charlie. Our estimate $\hat{\mu}$ of μ is simply the fraction of samples in which Charlie advised Anna. Under our assumptions, $\hat{\mu}$ has variance

$$\mathrm{Var}\,[\,\hat{\mu}\,] = \mathrm{Var}\Big[\,\frac{1}{n}\sum_{i=1}^{n} h(\mathbf{X}^{(\mathbf{i})})\,\Big] = \frac{\mathrm{Var}\,[\,h(\mathbf{X})]\,]}{n}, \tag{2}$$

which decreases linearly in the number n of samples.

Two aspects play a key role in selecting a sampling method: the time to obtain a sample $\boldsymbol{X}^{(i)}$ and the number of required samples to reach the desired accuracy.

Importance sampling (IS) is a method to increase efficiency by replacing the *target distribution* $P(\boldsymbol{X})$ by a new *proposal distribution* $Q(\boldsymbol{X})$. The increase in efficiency is obtained by either reducing the variance of the estimator (numerator of (2)) or by increasing the number of samples that can be obtained in a given amount of time (denominator). Since samples are taken from the "wrong" distribution $Q(\boldsymbol{X})$, we assign an *importance weight* $w(\boldsymbol{X}^{(i)}) = P(\boldsymbol{X}^{(i)})/Q(\boldsymbol{X}^{(i)})$ to each sample from the proposal distribution. We have

$$\mathrm{E}_P\,[\,h(\mathbf{X})\,] = \int h(\mathbf{x})P(\mathbf{x})\;\mathrm{d}\mathbf{x} = \int h(\mathbf{x})w(\mathbf{x})Q(\mathbf{x})\;\mathrm{d}\mathbf{x} = \mathrm{E}_Q\,[\,h(\mathbf{X})w(\mathbf{X})\,].$$

Here we use subscripts P and Q to indicate the distribution w.r.t. which the expectation is taken. Note that $\mathrm{E}_Q\,[\,w(\boldsymbol{X})\,] = 1$ and that, for example, $w(\boldsymbol{X}^{(i)}) > 1$ if possible world $\boldsymbol{X}^{(i)}$ is more likely under P than under Q. Thus the importance of sample $\boldsymbol{X}^{(i)}$ (when sampled from Q) is higher than average because we see the sample with lower probability than required.

The previous discussion assumes that the target distribution $P(\mathbf{X})$ is known. In the case of factor graphs, however, computing $P(\boldsymbol{x})$ for some possible world $\boldsymbol{x}$ is intractable in general because we do not know the normalization constant Z_P of Eq. (1). Instead, we make use of an *unnormalized distribution* $\tilde{P}(\boldsymbol{X}) = Z_P P(\boldsymbol{X}) = \prod_{f\in F} f(\boldsymbol{X}_f)$ (similarly $\tilde{Q}$). We obtain the *normalized IS estimator*

$$\hat{\mu}_{\mathrm{IS}} = \frac{\sum_{i=1}^{n} \tilde{w}(\mathbf{X}^{(i)})h(\mathbf{X}^{(i)})}{\sum_{i=1}^{n} \tilde{w}(\mathbf{X}^{(i)})} \approx \mathbb{E}_Q[h(\mathbf{X})], \tag{3}$$

where $\tilde{w}(\boldsymbol{X}^{(i)}) = \tilde{P}(\boldsymbol{X}^{(i)})/\tilde{Q}(\boldsymbol{X}^{(i)})$ are *unnormalized importance weights*. Estimator $\hat{\mu}_{\mathrm{IS}}$ is consistent,[1] asymptotically unbiased, and has approximate variance [KF09]

$$\mathrm{Var}_Q(\hat{\mu}_{\mathrm{IS}}) \approx \frac{(1 + \mathrm{Var}_Q[w(\mathbf{X})])\,\mathrm{Var}_P[h(\mathbf{X})]}{n}. \tag{4}$$

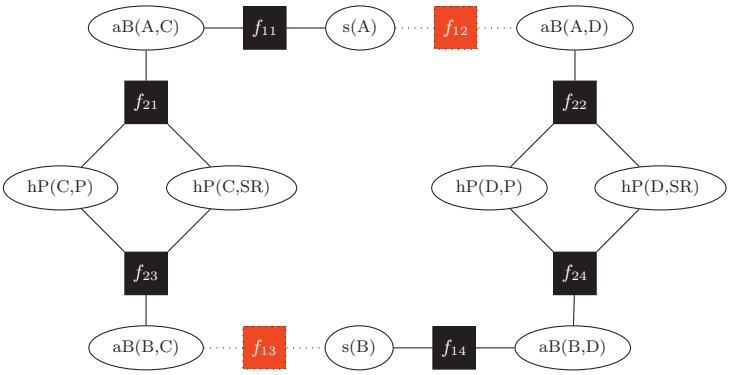
Thus the performance of estimator $\hat{\mu}_{\mathrm{IS}}$ is governed by (i) properties of the proposal and target distributions and (ii) the number of samples that can be taken in a given amount of time. Note that if $P = Q$, we obtain the standard Monte Carlo estimator and variance given in Eq. (2).

See [KF09] for a more thorough treatment of importance sampling.

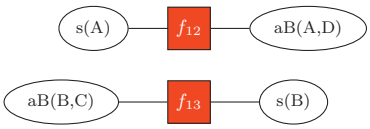
4.2 Parallel Probabilistic Inference with Importance Sampling

Given a *target factor graph* $G_P = (V, F_P)$ obtained from grounding the MLN, we parallelize probabilistic inference by constructing a *proposal factor graph* $G_Q = (V, F_Q)$ from G_P in such a way that sampling from G_Q can be easily parallelized. Here G_P represents the target distribution P while G_Q represents the proposal distribution Q. Since G_P and G_Q can represent different distributions, we use

[1] Here we assume that $Q(\mathbf{x}) \neq 0$ whenever $P(\mathbf{x})h(\mathbf{x}) \neq 0$, which holds for our choice of Q.



(a) Proposal distribution (black nodes and edges only)



(b) Importance weight (cut)

Figure 4: Partitioning of the factor graph for the university network

importance sampling to ensure that our estimates of marginal probabilities are (asymptotically) correct.

Denote by k the number of available processors. We obtain G_Q by partitioning the set of variables V into k partitions $V_1, \ldots, V_k$ such that $\bigcup_i V_i = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$, $1 \leq i, j \leq k$; we discuss how to actually obtain these partitions in subsequent sections. During the sampling process, processor i will be responsible for sampling the variables in partition V_i. For example, consider the ground university network reproduced in Fig. 4(a). Setting $k = 2$, a potential partitioning is given by setting

$$V_1 = \{\, \mathtt{s(A)},\ \mathtt{aB(A,C)},\ \mathtt{aB(B,C)},\ \mathtt{hP(C,P)},\ \mathtt{hP(C,SR)} \,\}$$

and

$$V_2 = \{\, \mathtt{s(B)},\ \mathtt{aB(A,D)},\ \mathtt{aB(B,D)},\ \mathtt{hP(D,P)},\ \mathtt{hP(D,SR)} \,\}\,.$$

We can now construct a factor graph $G_i = (V_i, F_i)$ for each partition by considering only a "local" subset of the factors from G_P, i.e., by setting

$$F_i = \{\, f \in F_P : \text{all neighbors of } f \text{ are contained in } V_i \,\}\,.$$

In our example, we have $F_1 = \{\, f_{11}, f_{21}, f_{23} \,\}$ and $F_2 = \{\, f_{22}, f_{24}, f_{14} \,\}$. The proposal factor graph G_Q is given by the union of the factor graphs for each partition, i.e., $F_Q = \bigcup_i F_i$. In our ongoing example, the proposal factor graph consists of the black nodes and edges in Fig. 4(a).

The partitioning approach described above ensures that in G_Q, there are no connections in between the set of variables in V_i and the set of variables in V_j,

$i \neq j$. One can show that this implies that the variables in V_i are probabilistically independent from the variables in V_j. This allows us to parallelize sampling from G_Q: Each processor i runs an arbitrary MCMC sampling scheme on factor graph G_i. To obtain a sample from the entire proposal distribution G_Q, we take the union of the samples produced by each processor (conceptually, see below).

We can view G_Q as a factor graph obtained from G_P by "dropping" some of the factors. Since these factors have no influence on the samples obtained from G_Q, we need to account for their effect via the importance weights. Denote by $G_{\text{cut}} = (V_{\text{cut}}, F_{\text{cut}})$ a *cut factor graph* that consists of the factors dropped from G_P and the variables directly connected to these factors; i.e.,

$$F_{\text{cut}} = F_P \setminus F_Q$$

and

$$V_{\text{cut}} = \{\, v \in V : \text{v is connected to at least one factor in } F_{\text{cut}} \,\}.$$

In our example, we have $F_{\text{cut}} = \{ f_{12}, f_{13} \}$ and $V_{\text{cut}} = \{ \texttt{s(A)}, \texttt{s(B)}, \texttt{aB(B,C)}, \texttt{aB(A,D)} \}$. The corresponding factor graph is shown in Fig. 4(b). To obtain the unnormalized importance weights, observe that

$$\begin{aligned}\tilde{w}(\boldsymbol{X}) = \frac{\tilde{P}(\boldsymbol{X})}{\tilde{Q}(\boldsymbol{X})} &= \frac{\prod_{f \in F_P} f(\boldsymbol{X}_f)}{\prod_{f \in F_Q} f(\boldsymbol{X}_f)} = \frac{\left(\prod_{f \in F_Q} f(\boldsymbol{X}_f)\right)\left(\prod_{f \in F_{\text{cut}}} f(\boldsymbol{X}_f)\right)}{\prod_{f \in F_Q} f(\boldsymbol{X}_f)} \\ &= \prod_{f \in F_{\text{cut}}} f(\boldsymbol{X}_f).\end{aligned}$$

Thus G_{cut} can be used to compute the importance weights using the values of only the variables in V_{cut}.

To summarize, we parallelize probabilistic inference as follows: We first partition the factor graph and distribute the partitions across the set of compute nodes. Each node computes a sample of its local factor graph using an arbitrary MCMC sampling method. After the samples have been obtained, we communicate the values of the cut variables (i.e., V_{cut}) to a coordinator node, which subsequently computes and broadcasts the unnormalized importance weight. Finally, each node updates the marginals of its local variables using Eq. (3).

4.3 The Optimal Partitioning

Recall Eq. (4), which defines the variance of the normalized IS estimator. For a fixed number n of samples, the variance depends on (1) the target distribution and (2) the variance of the importance weights $w(\boldsymbol{X})$. Since (1) is not affected by the partitioning, we subsequently focus on how to minimize (2). In our setting, we have

$$w(\boldsymbol{X}) = \frac{Z_Q}{Z_P} \tilde{w}(\boldsymbol{X}),$$

where Z_P and Z_Q are the normalization constants of G_P and G_Q, respectively. We obtain

$$\mathrm{Var}_Q[w(\boldsymbol{X})] = \frac{Z_Q^2}{Z_P^2}\,\mathrm{Var}_Q[\tilde{w}(\boldsymbol{X})] = \frac{\mathrm{Var}_Q[\tilde{w}(\boldsymbol{X})]}{E_Q^2[\tilde{w}(\boldsymbol{X})]} = \mathrm{CV}_Q^2(\tilde{w}(\boldsymbol{X})),$$

where $\mathrm{CV}_Q(\tilde{w}(\boldsymbol{X})) = \sqrt{\mathrm{Var}_Q[\tilde{w}(\boldsymbol{X})]}/E_Q[\tilde{w}(\boldsymbol{X})]$ denotes the *coefficient of variation* (CV) of $\tilde{w}(\boldsymbol{X})$. In the derivation, we used the fact that $E[w(\boldsymbol{X})] = 1$ and thus $E[\tilde{w}(\boldsymbol{X})] = Z_P/Z_Q$. To minimize the variance of the IS estimator $\hat{\mu}_{\mathrm{IS}}$, we thus need to minimize the CV of $\tilde{w}(\boldsymbol{X})$ under proposal Q. To do this, we need to infer the ratio Z_Q/Z_P and the joint distribution of the variables in V_{cut} (both of which depend on our choice of proposal), i.e., we need to run inference. Since our ultimate goal is to actually parallelize inference, such an approach is impractical.

The situation is further complicated by the fact that the partitioning that minimizes the CV is not necessarily the best performing one in practice. Recall that in order to compute the unnormalized importance weight, we need to obtain a sample from each of the partitions. If the partitions are balanced (e.g., equal number of variables and/or factors), parallelization is effective because every processor has roughly the same amount of work. If partitions are imbalanced, parallalization may not be effective since one of the processors may require significantly more time to obtain its sample than the other processors. Thus with k processors, we obtain a speedup somewhere in between 1 (no balancing) and k (perfect balancing).

We conclude that the optimal partitioning trades off statistical efficiency and parallelization benefits. In principle, we can determine this optimal partitioning given an appropriate model of sampling cost. However, such an approach is impractical: the benefits of parallelization are outweighed by the cost of determining the partitioning. For this reason, we settle for a "good" partitioning instead of the optimal one.

4.4 A Good Partitioning

To obtain a practicable procedure that finds a good but not necessarily the best partitioning, we (1) do not minimize but bound the CV of the unnormalized importance weights and (2) always create balanced partitions (i.e., we do not trade off variance and computational cost). With these relaxations, we are able to apply existing hypergraph partitioning algorithms to our problem. Moreover, these relaxations form the basis for the parallel grounding method described in Sec. 5.

Recall that the unnormalized importance weight is given by $\tilde{w}(\boldsymbol{X}) = \prod_{f \in F_{\mathrm{cut}}} f(\boldsymbol{X}_f)$. From our definition of factor functions, we know that f can only take two possible values: e^{w_f} (corresponding formula satisfied) or 1 (otherwise). Denote by $f_{\min}$ ($f_{\max}$) the smaller (larger) of the two values. Given a cut F_{cut}, we obtain

$$\tilde{w}_{\min} = \prod_{f \in F_{\mathrm{cut}}} f_{\min} \leq \tilde{w}(\boldsymbol{X}) \leq \prod_{f \in F_{\mathrm{cut}}} f_{\max} = \tilde{w}_{\max}.$$

To bound the CV of $\tilde{w}(\boldsymbol{X})$, we compute the largest CV realizable by any distribution on the interval $[\tilde{w}_{\min}, \tilde{w}_{\max}]$. We thus replace the actual distribution of

$\tilde{w}(\boldsymbol{X})$ by its worst-case distribution; this trick allows us to avoid running inference while still obtaining guarantees on the quality of estimation. One can show that the largest CV is realized by a distribution on $\{\, \tilde{w}_{\min}, \tilde{w}_{\max} \,\}$. Such a distribution is parameterized by a parameter $p_{\max}$ for the probability of observing $\tilde{w}_{\max}$; set $p_{\min} = 1 - p_{\max}$ and $\bar{w} = p_{\min}\tilde{w}_{\min} + p_{\max}\tilde{w}_{\max}$. We obtain

$$\mathrm{CV}_Q[\tilde{w}] \leq \max_{0 \leq p_{\max} \leq 1} \frac{\sqrt{p_{\min}(\tilde{w}_{\min} - \bar{w})^2 + p_{\max}(\tilde{w}_{\max} - \bar{w})^2}}{\bar{w}}. \tag{5}$$

One way to compute the above bound is to (conceptually) "normalize" the factor graph before running inference. The *normalized factor graph* can be obtained by replacing every factor function $f \in F$ by $f'(\boldsymbol{X}) = f(\boldsymbol{X})/f_{\min}$; the distribution represented by the factor graph is not affected by such a scaling. After normalization, we have $\tilde{w}'_{\min} = 1$ so that the bound of (5) depends only on $\tilde{w}'_{\max}$. In fact, the *normalized bound* is monotonically increasing in $\tilde{w}'_{\max}$ so that we pick the partitioning that minimizes $\tilde{w}'_{\max} = \prod_{f' \in F'_{\text{cut}}} f'_{\max}$. If the so-obtained bound on the CV is close to 0, importance sampling is guaranteed to be statistically efficient. In contrast, if the bound is large, the performance of IS depends on the distribution of the cut variables; IS may or may not be effective.

We can use existing weighted hypergraph partitioning algorithms to find the partitioning that minimizes $\tilde{w}'_{\max}$. A hypergraph partitioning algorithm [KL70] finds a partitioning of the variables such that the sum of the weights of the hyperedges that cross partitions is minimized. Note that minimizing $\tilde{w}'_{\max}$ is equivalent to minimizing $\log(\tilde{w}'_{\max}) = \sum_{f' \in F'_{\text{cut}}} \log(f'_{\max})$, which is in summation form. To construct the *weighted hypergraph* for $G = (V, F)$, replace each factor $f \in F$ by a hyperedge that connects the variables in $\boldsymbol{X}_f$; the hyperedge is assigned weight $\log(f'_{\max})$. The minimum cost hypergraph partitioning is exactly the partitioning with the smallest possible bound on the CV. Note that the use of hypergraph partitioning for probabilistic inference is not novel (e.g., see [NRDS11]). To the best of our knowledge, however, the connection between hypergraph partitioning and the effectiveness of parallel probabilistic inference has not been established before.

In order to facilitate parallel processing, we modify the above approach and search for a *balanced* partitioning instead of an arbitrary partitioning. The perhaps simplest possible approach—which we also used in our experiments—is to require that each partition contains roughly the same number of variables. The hope is that the cost of obtaining a sample is then roughly the same for each partition. A key advantage of this particular balancing condition is that it is directly supported by most hypergraph partitioners. In fact, our experiments suggest that the so-obtained partitions work well in practice.

5 Parallel Grounding

Since both grounding and graph partitioning can be expensive, we develop an MLN partitioning technique that significantly reduces partitioning cost and also parallelizes the grounding step. The key idea of our approach is to partition

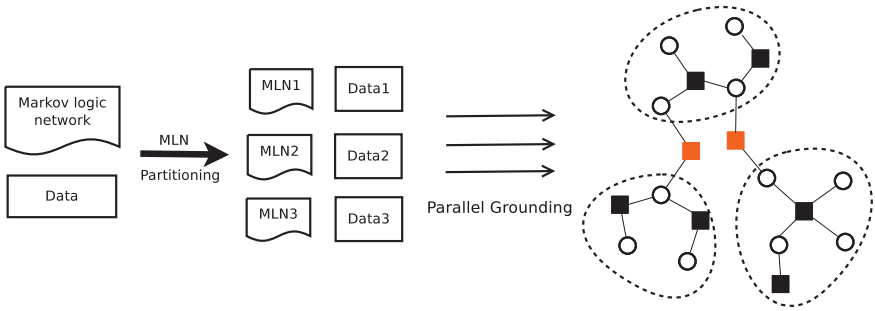


Figure 5: MLN partitioning and parallel grounding ($k = 3$).

the MLN directly—i.e., before grounding—and to ground the resulting partitioned MLNs in parallel. The MLN partitioning step draws from ideas from the parallel databases; it exploits the connection between grounding an MLN and computing a set of database joins [NRDS11]. Fig. 5 illustrates our approach.

5.1 Grounding via Database Queries

To establish the connection between grounding and database joins, recall the university MLN of Fig. 1 as well as its grounding shown in Fig. 2. The first rule of the MLN states that advisees must be students and involves predicates $\texttt{student}(s)$ and $\texttt{advisedBy}(s, p)$. Suppose that we create a *hypothetical database relation* for each predicate that occurs in the MLN; a predicate's relation contains a tuple for each ground instance of the predicate as well as a globally unique *identifier*. For students Anna and Bob, and professors Charles and Debbie, we obtain

$$\begin{aligned}\texttt{Student}(\underline{sid}, s_1) &= \{\,(1, \texttt{Anna}),\ (2, \texttt{Bob})\,\},\\ \texttt{AdvisedBy}(\underline{aBid}, s_2, p_1) &= \{\,(3, \texttt{Anna}, \texttt{Charles}),\ (4, \texttt{Anna}, \texttt{Debbie}),\ (5, \texttt{Bob}, \texttt{Charles}),\\ &\qquad (6, \texttt{Bob}, \texttt{Debbie})\,\}.\end{aligned}$$

Observe that these relations precisely capture the set of variables in the ground MLN corresponding to the `student` and `advisedBy` predicates, and that each variable is assigned a unique identifier. To additionally capture the factors that connect these variables, we perform a database join for each clause in the MLN and project to the variable identifiers. The join condition consists of the set of equality predicates induced by the MLN rule. For example, the first rule of Fig. 1 is given by $\texttt{s}(s) \vee \neg\texttt{aB}(s)$ (here in CNF). The corresponding database query and result is

$$\begin{aligned}\texttt{R1} = \pi_{sid,aBid}(\texttt{Student} \bowtie_{s_1=s_2} \texttt{AdvisedBy}) &= \{\,(1,3),\ (1,4),\ (2,5),\ (2,6)\,\}\\ &= \{\,f_{11},\ f_{12},\ f_{13},\ f_{14}\,\}.\end{aligned}$$

Here `R1` contains a single tuple for each factor obtained from grounding the first rule; a factor's tuple consists of the variable identifiers of its arguments.

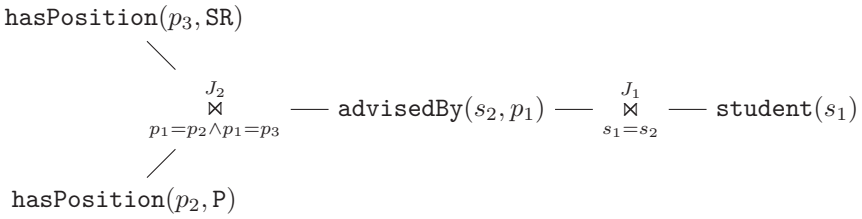


Figure 6: Join graph for the MLN of Fig. 1.

To summarize, we (conceptually) ground an MLN by instantiating a set of relations (corresponding to the ground variables) and computing a database join for each clause of the MLN (corresponding to the factors). This grounding process can be described with a *join graph* $G = (\mathscr{R}, \mathscr{J})$, where $\mathscr{R} = \{ R_1, \ldots, R_n \}$ denotes a set of relations and $\mathscr{J} = \{ J_1, \ldots, J_m \}$ denotes a set of joins (factors). Each $J \in \mathscr{J}$ is a hyperedge annotated with a join condition; the hyperedge connects to the relations occurring in the join condition. Fig. 6 shows the join graph for our running example; here we use predicates instead of relations for brevity. Note that in the foregoing discussion, we have ignored the evidence database and assumed open-world semantics. When an evidence database is given and grounding is performed under closed-world semantics, we directly use the relations from the evidence database (which have an additional Boolean `Value` attribute) instead of the hypothetical relations. To simplify exposition, however, we continue to focus on open-world grounding without an evidence database unless stated otherwise.

5.2 Partitioning a Markov Logic Network

Parallel database systems make use of vertical and horizontal partitioning techniques to parallelize query processing. In the following, we discuss how these techniques can be applied to the (hypothetical) relations of an MLN to allow for parallel grounding. We focus on horizontal partitioning throughout; similar techniques can be used to handle vertical partitioning.

A *horizontal partitioning* of a relation R w.r.t. a set A of its attributes is a set $R_1, \ldots, R_k$ of relations such that $\bigcup_i R_i = R$ and $\pi_A(R_i) \cap \pi_A(R_j) = \emptyset$ whenever $i \neq j$, where $1 \leq i, j \leq k$. For example, when $k = 2$, a horizontal partitioning of the `AdvisedBy` relation w.r.t. attribute p is given by $\texttt{AdvisedBy}_1 = \{ (3, \texttt{A}, \texttt{C}), (5, \texttt{B}, \texttt{C}) \}$ and $\texttt{AdvisedBy}_2 = \{ (4, \texttt{A}, \texttt{D}), (6, \texttt{B}, \texttt{D}) \}$. Such a partitioning can be performed at the MLN level, i.e., before grounding. Continuing the example, we obtain two new predicates $\texttt{advisedBy}_1(s, p_{11})$ and $\texttt{advisedBy}_2(s, p_{12})$ with domains $\mathrm{dom}(p_{11}) = \{ C \}$ and $\mathrm{dom}(p_{12}) = \{ D \}$. Horizontal partitioning allows us to spread the ground variables of a single relation across multiple compute nodes (partition i is stored on node i). In fact, if the partitioning is performed at the MLN level, we can ground each partition in parallel directly at its respective compute node.

Our aim is to partition the relations in a way amenable to parallel probabilis-

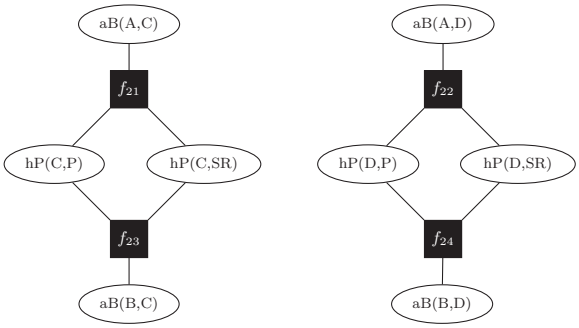


Figure 7: Partial factor graph obtained from grounding rule 2 of Fig. 1

tic inference, i.e., to minimize the number of factors spanning multiple partitions in addition to balancing the number of variables per partition. For example, consider the factor graph obtained when grounding only the second rule of our example MLN of Fig. 1. As can be seen in Fig. 7, this factor graph consists of two connected components, i.e., two subgraphs that do not share any factors. Using horizontal partitioning, we can identify and exploit the existence of multiple connected components directly at the rule level. The key idea is to make use of the co-partitioning techniques used in parallel databases: Two relations R_1 and R_2 are *co-partitioned* with respect to a set of attributes A_1 and A_2 (of equal domains), respectively, if R_l is horizontally partitioned w.r.t. A_l into $R_{l1}, \ldots, R_{lk}$, $l \in \{1, 2\}$, and $\pi_{A_1}(R_{1i}) \cap \pi_{A_2}(R_{2j}) = \emptyset$ for $1 \leq i, j \leq k$ and $i \neq j$. For example, suppose that we partition relation

$$\texttt{HasPosition}(hPid, p_2, v) = \{ (7, \texttt{C}, \texttt{P}),\ (8, \texttt{C}, \texttt{SR}),\ (9, \texttt{D}, \texttt{P}),\ (10, \texttt{D}, \texttt{SR}) \}$$

on attribute p to obtain relations $\texttt{HasPosition}_1 = \{ (7, \texttt{C}, \texttt{P}), (8, \texttt{C}, \texttt{SR}) \}$ and $\texttt{HasPosition}_2 = \{ (9, \texttt{D}, \texttt{P}), (10, \texttt{D}, \texttt{SR}) \}$. Then relations `AdvisedBy` (see above) and `HasPosition` are co-partitioned w.r.t. attributes p_1 and p_2, respectively.

If R_1 and R_2 are co-partitioned on A_1 and A_2, we can perform join $J = R_1 \bowtie_{A_1=A_2} R_2$ *locally* at each compute node: Each node i computes $R_{1i} \bowtie_{A_1=A_2} R_{2i}$. This fact is heavily exploited for efficient join processing in parallel database systems. In our setting, horizontal partitions consist of sets of ground variables and join results of sets of factors and their arguments. Suppose that join J above corresponds to some MLN rule. Since R_1 and R_2 are co-partitioned w.r.t. to the join attributes, we can not only compute join J locally at each compute node, but also guarantee that the local join results contain only local variables. In other words, all factors produced by a co-partitioned join will be local.

Our MLN partitioning technique aims to co-partition the relations in an MLN such that the number (or sum of weights) of the factors that span partitions is minimized. Since there are generally several formulas in which each relation occurs, and since we cannot partition a relation on multiple sets of attributes, we may not be able to find a co-partitioning that localizes all joins. We thus aim to find a good partitioning, i.e., one in which "important" joins are localized. Our partitioning

technique is based on the following insight: The number of ground variables or factors of a predicate or rule, respectively, is given by the size of its corresponding relation or join. In our example, there are $|\texttt{Student}| = 2$ ground variables for `student` predicate, $|\texttt{AdvisedBy}| = 4$ ground variables for the `advisedBy` predicate, and $|\texttt{R1}| = 4$ factors for the first rule. Cardinality estimation techniques from the database literature allow us to estimate these quantities accurately; we can thus estimate the size of a factor graph for a given MLN and, more importantly, for a given horizontal partitioning on an MLN.

Revisiting the example of Fig. 6, the total number of factors in the factor graph is given by $|J_1| + |J_2|$. If we co-partition `aB` and `hP` on attribute p, then J_2 becomes local. In doing so, however, we cannot co-partition `aB` and `s` so that join J_1 is not local. Vice versa, if we choose a co-partitioning that localizes J_1, join J_2 will be non-local. The optimum choice of partitioning depends on the join sizes $|J_1|$ and $|J_2|$, as well as on the weights associated with the corresponding MLN rules. Intuitively, we want to localize joins with high cardinality and high weight; joins with low cardinality and low weight do not incur a high cost when not localized.

As indicated above, our partitioning method (described in detail in the next section) relies on accurate join size estimates. There exists a vast amount of literature on join size estimation [SS94], which can be readily exploited. When we use open-world grounding, however, join size estimation is particularly simple. Recall that under open-world grounding, all relations are complete, i.e., they consist of every possible tuple. The size of a complete relation R with attributes $A_1, \ldots, A_n$ is given by $|\text{dom}(A_1)||\text{dom}(A_2)| \cdots |\text{dom}(A_n)|$. When we compute the join $J = R_1 \bowtie_{R_1.A=R_2.A} R_2$ of two complete relations R_1 and R_2, the resulting join size is given by $|J| = |R_1||R_2|/|\text{dom}(A)|$. When R_1 and R_2 are both partitioned into k equally-sized partitions, and at least one of the two relations is not partitioned on A, then the number of local factors is given by $|J|/k$ and consequently the number of non-local factors by $|J|(1 - 1/k)$. Of course, join size estimation is much more involved when grounding is performed under closed-world semantics. The simplest way to handle closed-world grounding is to use open-world join size estimation as described above, but to use closed-world semantics when actually grounding each partition. We use this simple approach in our experiments; more elaborate join size estimation techniques are likely to further improve our results.

5.3 Finding a Good Partitioning

To find a good MLN partitioning, we encode the partitioning problem as a 0/1 integer linear program (ILP), which we solve using a standard ILP solver.

In what follows, we use index i for the join edges, index j for attributes, and index l for relations of the join graph. Denote by $A(R_l)$ the set of attributes in relation R_l and by $R(J_i)$ the multiset of relations occuring in join J_i. We assume that $A(R_{l_1}) \cap A(R_{l_2}) = \emptyset$ whenever $l_1 \neq l_2$. We encode each join edge J_i via a set C_i as follows: For each pair of R_{l_1} and R_{l_2} in $R(J_i)$, C_i contains the set E_{l_1,l_2} of pairs of attributes that are equalized by the join condition. Each pair contains one attribute from R_{l_1} and one from R_{l_2}. In our running example, we have $R(J_2) =$

$\{\, \texttt{hP}(p_2, \texttt{SR}), \texttt{hP}(p_3, \texttt{P}), \texttt{aB}(s_2, p_1) \,\}$ and $C_2 = \{\, \{\, (p_2, p_3) \,\}, \{\, (p_1, p_3) \,\}, \{\, (p_2, p_3) \,\} \,\}$. Note that we treat $\texttt{hP}(p_2, \texttt{SR})$ and $\texttt{hP}(p_2, \texttt{P})$ as different relations since they are disjoint. Denote by x_{A_j} a 0/1 variable that takes a value 1 if the relation R_l that contains attribute A_j (i.e., $A_j \in A(R_l)$) is going to be partitioned on A_j. Furthermore, denote by y_{J_i} a 0/1 variable that takes a value 1 if J_i is localized by the partitioning $\{\, x_{A_j} \,\}$. In our example, we have

$$y_{J_2} = [(x_{p_2} \wedge x_{p_3}) \wedge (x_{p_1} \wedge x_{p_3}) \wedge (x_{p_2} \wedge x_{p_3})].$$

Generally, denote by $\mathscr{P}_i$ a set of sets of attribute pairs, where each set of attribute pairs is obtained by selecting a single element of each $E_{l_1,l_2} \in C_i$ (in our example, $\mathscr{P}_2 = C_2$). Only sets of pairs that mutually share at least one attribute are included in $\mathscr{P}_i$. Then the optimal partitioning can be obtained by solving the ILP

$$\max \quad \sum_{J_i} |J_i| \cdot w_{J_i} \cdot y_{J_i} \tag{1}$$

$$\text{s.t.} \quad \sum_{A_j \in A(R_l)} x_{A_j} \leq 1 \qquad \text{for all } R_l, \tag{2}$$

$$y_{J_i} = \bigvee_{P \in \mathscr{P}_i} \Big[\bigwedge_{(A_{j_1}, A_{j_2}) \in P} (x_{A_{j_1}} \wedge x_{A_{j_2}}) \Big] \qquad \text{for all } J_i, \tag{3}$$

$$x_{A_j} \in \{\, 0, 1 \,\} \qquad \text{for all } A_j.$$

The objective function (1) maximizes the weighted sum of the sizes of local joins; weight w_{J_i} is taken from the MLN rule corresponding to join J_i. One can show that this objective function is equivalent to minimizing the number of non-local factors. Depending on the particular probabilistic inference algorithm being used, other objective functions may be more suitable (e.g., sum of joins sizes or number of variables with at least one non-local factor). Condition (2) ensures that every relation is partitioned on only one attribute (the ILP can be extended to support partitioning on multiple attributes). Finally, condition (3) computes the y_{J_i} variables using Boolean formulas. The formulas can be converted to a pure ILP by introducing appropriate helper variables.

Once the ILP has been solved, we horizontally partition the relations based on the solution. In more detail, we split the domain of each partitioning attribute into k equally sized parts; e.g., using range-based or hash partitioning. Relations that are not partitioned are randomly distributed across the compute cluster; this ensures that the number of query variables is the same across nodes so that partitions are balanced.[2] Note that the cost of our MLN partitioning strategy is virtually independent of both k and the domain size, which ensures good scalability.

6 Experimental Evaluation

We conducted a number of preliminary experiments to gauge the viability of our fully parallel approach to MLN inference. In particular, we (1) compared

[2] An alternative approach is to extend the ILP with support for vertical partitioning.

existing graph partitioning algorithms with our MLN partitioning method in terms of both computational cost and result quality, (2) investigated whether bounding the CV of the unnormalized importance weights indeed leads to good partitionings, and (3) whether parallel probabilistic inference reduces overall cost. Our results provide initial justification for our approach.

6.1 Experimental Setup

We implemented a prototype system that consists of components for MLN partitioning, grounding, and parallel probabilistic inference. All algorithms were implemented in C++ using pthreads for parallel processing; during probabilistic inference, the partitioned factor graph resided in main memory and each partition was processed by a single core. We used the GNU Linear Programming Kit (GLPK) to solve the ILP. All experiments were run on a machine with an 2.4GHz Intel Xeon CPU E5530 (8 cores) and 48GB of main memory.

Most of our experiments were conducted on the UW-CSE dataset.[3] This real-word dataset contains information about the Department of Computer Science and Engineering of the University of Washington, including relationships between professors, students, courses, and publications. The dataset is associated with an MLN that aims to predict the `advisedBy` and `tempAdvisedBy` relationship; the MLN consists of 22 predicates and 94 clauses. From the MLN, we removed 6 clauses because they contain an existential quantifier; these clauses are not yet supported by our prototype. We learned the weights of the rules in the modified MLN using Alchemy as described in [RD06]. After grounding (AI dataset), the factor graph consists of roughly 9000 query variables and 1M factors. Although the UW-CSE dataset is relatively small, the benefits of parallel processing did materialize in our experiments.

6.2 MLN Partitioning

In our first experiment, we evaluated the computational cost and quality of partitioning of both traditional ground-first methods and our proposed partition-first approach. For the former, we used two different hypergraph partitioners: (1) PaToH,[4] a state-of-the-art hypergraph partitioner and (2) the graph partitioning heuristic used by Tuffy [NRDS11].[5] The experiments were conducted on the UW-CSE dataset with $k = 2$ and $k = 4$ partitions.

Quality. We used a number of metrics to evaluate the quality of the partitioned factor graph: (i) the number $|F_{\text{cut}}|$ of factors in the cut (measure of the computational cost for weight computation), (ii) the sum $\log(\tilde{w}_{\max})$ of the loga-

[3] `http://alchemy.cs.washington.edu/data/uw-cse`

[4] `http://bmi.osu.edu/~umit/software.html#patoh`

[5] The Tuffy heuristic sorts all factors by descending weight. It then scans the factors sequentially and tries to assign the variables of the current factor to the current partition (if not yet assigned). If the partition size would exceed some threshold by doing so, Tuffy finalizes the current partition and starts a new, empty partition.

k	Approach	Factors in cut	Weight of cut	Balancing	Runtime
$k = 2$	PaToH	4678	1109.04	0.000	948.288s
	Tuffy	4686	1108.66	0.000	1.092s
	MLN part.	4690	1108.47	0.000	0.003s
$k = 4$	PaToH	63001	64500.40	0.012	952.254s
	Tuffy	7040	1662.46	0.000	1.288s
	MLN part.	7023	1662.84	0.000	0.003s

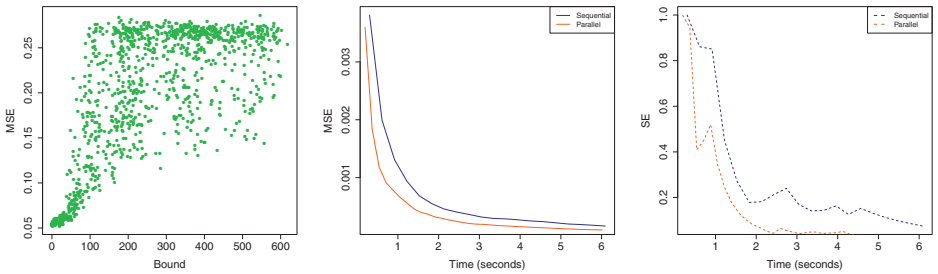
Table 1: Comparison of various graph partitioning approaches on the UW-CSE dataset

rithmic weights of these factors (measure of effectiveness of importance sampling), and (iii) the CV of the partition sizes in terms of number of variables (measure of balancing, 0 indicates perfect balancing). Our results are shown in Table 1. For $k = 2$ partition, PaToH, Tuffy, and our MLN partitioning approach give similar results. For $k = 4$, however, PaToH does not find a good partitioning, whereas Tuffy and our proposed MLN approach perform much better. Again, Tuffy and MLN partitioning perform comparably. Perhaps surprisingly, the simple Tuffy heuristic worked extremely well on the UW-CSE dataset. We conclude that MLN partitioning can indeed find good graph partitionings and may even outperform state-of-the-art partitioners.

Computational cost. We focus on the computational cost of computing the factor graph partitioning in terms of wall-clock time, i.e., we exclude the time for actually grounding the network. Table 1 shows our results. We found that PaToH is two orders of magnitude slower than Tuffy, which in turn is orders of magnitude slower than MLN partitioning. The excessive runtime of PaToH is due to the fact that PaToH is a generic hypergraph partitioner; it is unaware of and thus cannot exploit the structure of the ground factor graph. Tuffy is significantly faster because it uses an inexpensive heuristic partitioning algorithm. It is a viable option on datasets for which this heuristic works well. In both approaches, we need to distribute the partitions across the compute cluster after grounding to perform parallel probabilistic inference. This data distribution step is avoided by our MLN partitioning approach because partitions are grounded directly at their respective compute nodes. Moreover, since MLN partitioning is performed on the MLN level, it is independent of the size of the evidence database and thus much faster than the ground-first methods.

6.3 Parallel Probabilistic Inference

Quality. We conducted a simple experiment to investigate whether our bounding strategy for the CV of the unnormalized importance weights is effective. We generated a set of 1100 random graphs with 1000 variables and 1000 factors each. Each factor was connected to between 1 and 3 variables picked at random; the weights were sampled from a Normal$(0.67, 25)$ distribution. We ran a



(a) Effectiveness of bound (synthetic data) (b) Average MSE (UW-CSE) (c) Maximum SE (UW-CSE)

Figure 8: Sequential and parallel probabilistic inference (4 threads)

large number of sampling steps (10000) of a Gibbs sampler [KF09]; the resulting marginal probabilities were taken as ground truth. We then randomly partitioned the variables into two equally-sized sets and computed (1) the corresponding bound on the CV and (2) the mean square error (MSE) of the estimated marginal probabilities obtained by running a fixed number of 10000 importance sampling steps. Fig. 8(a) plots the bound vs. the MSE; each data point corresponds to one random partitioning. As can be seen, the MSE is small when the CV bound is small, which indicates that our bounding strategy is effective. Moreover, the MSE varies significantly when the CV bound is large. The reason for this behavior is that the bound is based on a worst-case distribution; when the actual distribution is far from worst-case, we may obtain better results than suggested by the bound.

Computational cost. We evaluated parallel probabilistic inference using importance sampling on $k = 4$ partitions; the partitions have been created using MLN partitioning as described in the previous section. As before, we obtained the ground truth by running a large number of Gibbs sampling steps (250k steps on each variable) on the unpartitioned factor graph. Fig. 8 plots the average MSE as well as the highest square error (SE) on an individual variable for both a sequential sampler and a parallel sampler with 4 threads. In both cases, the parallel inference method converged much faster than the sequential method so that importance sampling was effective.[6]

7 Conclusion and Future Work

We proposed a fully parallel approach to inference in Markov logic networks. In contrast to prior work, we parallelized not just the final probabilistic inference step, but also the intermediate grounding and graph partitioning steps. In more detail, we described and analyzed a simple parallel probabilistic inference algorithm based on importance sampling. Our analysis clarifies the connection between importance

[6] Note that this particular dataset is characterized by fast mixing times, which makes even the sequential sampler unusually fast.

sampling on factor graphs and graph partitioning. Since graph partitioning can be expensive, we leveraged ideas from parallel database systems to partition the Markov logic network itself, i.e., before grounding the network. Our MLN partitioning technique reduces partitioning time by multiple orders of magnitude, while producing partitionings competitive to state-of-the-art graph partitioners. Our experiments give initial evidence that both MLN partitioning and parallel probabilistic inference can speed up inference in Markov logic networks significantly. Future work includes better handling of closed-world semantics, better sampling methods, and a fully distributed implementation of our approach.

References

[ASW08] A. Asuncion, P. Smyth, and M. Welling. Asynchronous Distributed Learning of Topic Models. In *NIPS*, 2008.

[DVKMG09] F. Doshi-Velez, D. Knowles, S. Mohamed, and Z. Ghahramani. Large Scale Nonparametric Bayesian Inference: Data Parallelisation in the Indian Buffet Process. In *NIPS*, 2009.

[GLGG11] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin. Parallel Gibbs Sampling: From Colored Fields to Thin Junction Trees. *Journal of Machine Learning Research*, 2011.

[KF09] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[KL70] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell system technical journal*, 1970.

[KSRD05] S. Kok, P. Singla, M. Richardson, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, 2005.

[MR10] L. Mihalkova and M. Richardson. Speeding up inference in statistical relational learning by clustering similar query literals. In *ILP*, 2010.

[NASW07] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed Inference for Latent Dirichlet Allocation. In *NIPS*, 2007.

[NRDS11] F. Niu, C. Ré, A. Doan, and J. Shavlik. Tuffy: scaling up statistical inference in Markov logic networks using an RDBMS. *PVLDB*, 2011.

[NZRS11] F. Niu, C. Zhang, C. Ré, and J. Shavlik. Felix: Scaling Inference for Markov Logic with an Operator-based Approach. *CoRR*, 2011.

[PD07] H. Poon and P. Domingos. Joint Inference in Information Extraction. In *AAAI*, 2007.

[PD10] H. Poon and P. Domingos. Unsupervised ontology induction from text. In *ACL*, 2010.

[RD06] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 2006.

[Rie08] S. Riedel. Improving the Accuracy and Efficiency of MAP Inference for Markov Logic. In *UAI*, 2008.

[SD06] P. Singla and P. Domingos. Entity Resolution with Markov Logic. In *ICDM*, 2006.

[SN09] J. Shavlik and S. Natarajan. Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In *IJCAI*, 2009.

[SS94] Arun N. Swami and K. Bernhard Schiefer. On the Estimation of Join Result Sizes. In *EDBT*, 1994.

Experiences from Developing the Domain-Specific Entity Search Engine GeneView

Philippe Thomas, Johannes Starlinger, Ulf Leser

Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin
{thomas,starlin,leser}leser@informatik.hu-berlin.de

Abstract: GeneView is a semantic search engine for the Life Sciences. Unlike traditional search engines, GeneView analyzes texts upon import to recognize and properly handle biomedical entities, relationships between those entities, and the structure of documents. This allows for a number of advanced features required to work effectively with scientific texts, such as entity disambiguation, ranking of documents by entity content, linking to structured knowledge about entities, user-friendly highlighting of entities etc. As of now, GeneView indexes approximately ~21,4M abstracts and ~358K full texts with more than 200M entities of 11 different types and more than 100K relationships. In this paper, we describe the architecture underlying the system with a focus on the complex pipeline of advanced NLP and information extraction tools necessary for achieving the above functionality. We also discuss open challenges in developing and maintaining a semantic search engine over a large (though not web-scale) corpus.

1. Introduction

The vast majority of novel findings in Life Science research are first presented in the scientific literature. Over the years, the amount of texts in this domain has grown enormously and has reached a point where finding specific information becomes troublesome. In 2011 alone, MEDLINE archived more than 800,000 new articles, which corresponds to an increase of more than one article per minute. Besides the rapidly growing sheer number of articles, also the length of available texts is growing, as more and more articles become freely available as full text. Simple and fast access to the scientific literature is enormously important for researchers to keep up-to-date with their field. In the life sciences, researchers typically (but not always) search for information about some specific biomedical entity, like genes, diseases, mutations etc. Such a search is very difficult for a number of reasons, which we explain using genes as an example. Firstly, genes usually have many synonyms: on average, Entrez gene gives 2.2 synonyms for each human gene, with a maximum of 31 synonyms for the gene OR4H6P (Entrez gene Id 26322). In addition to synonyms, morphological variations are very frequent in scientific articles (e.g. BRCA1 or BRCA-1). Secondly, gene names are highly ambiguous, both with other genes or other biological entities (like diseases), and

with common English words. For instance, many genes are named after the phenotype they are associated with, leading to names such as „white“ or „hedgehog“. On the other hand, evolutionary related genes in different species often have the same name although they should be considered as different entities in most applications. Thirdly, single genes are studied from very different viewpoints, often leading to the invention of slight variations of names (like the mRNA created from a gene being named slightly different than the gene itself). Which of these variations are relevant for a given search is difficult to express. Finally, gene names follow no regular structure but can appear as anything from a three letter acronym to a multi-token complex name. Similar problems exist for other entity types, such as diseases (whose names often contain ordinary persons' names, like „Wilsons disease“), or medical symptoms (whose names can be used in many different contexts not related to diseases, e.g. "shiver" or "cold"). The situation becomes worse when not only information about a single entity is searched, but about relations between entities, like genes associated to a disease or mutations associated with metabolization rates of a drug. Finding all genes targeted by a drug X is impossible with conventional retrieval engines.

As a result, searches often lead to unsatisfactory results. For instance, [DMNL09] reported that over one third of all 58 million PubMed queries collected for March 2008 result in hundreds or even thousands of results. It also directly impedes research: [OW04] pointed out that ambiguous nomenclature led to multiple discoveries of the same mutation. Consequently, there is a growing body of research trying to provide improved search for scientific texts [Lu11]. A pre-requisite for such advanced features is the high-quality recognition of entities (also called named entity recognition, NER) and relationships between entities in a given text (also called relationship extraction, RE). This area has seen intensive research over the last decade [ZDF+08]. In contrast to other domains, where especially NER seems to be considered as an essentially solved problem [Bal12], in biomedicine both problems are far from having been solved in a satisfying manner. For instance, the best gene recognition systems to-date achieve an F-measure of roughly 85% [KMS+08]; the best chemical taggers reach less than 70% F-Measure [RWL12]; the best tools for recognizing disease names reach around 80% F-measure [CL10]. The situation is worse when it comes to RE. The currently best systems for recognizing drug-drug interactions reach an F-Measure of 65% [TNS+11]; recognition of protein-protein-interactions, despite that literally hundreds of papers have been devoted to this topic, cannot be performed with more than ~60% F-measure [BKS10].

State-of-the-art tools usually are the result of long-standing projects and require considerable experience, effort, and time. Still, many are freely available, but implementations differ in terms of programming language, required libraries, dependencies from other tools, configuration etc. Especially the dependency of NER and RE on specific NLP tools sometimes makes it necessary to process the same text multiple times with essentially the same goal (like POS tagging), but using different

tools. Building a high-quality entity search engine thus requires bundling the best available algorithms into complex pipelines of different algorithms processing the same text with a different purpose. Each algorithm produces specific annotations, which often need to be transformed into different formats to be read by the next algorithms.

In this paper, we describe GeneView, a full-fledged entity search engine for biomedical publications. It currently identifies and normalizes ten different entity types (chemicals, cell-types, diseases, drugs, enzymes, genes, histone modifications, single nucleotide polymorphisms (SNP), species, and tissues) and three relationship types (protein-protein-interactions, regulatory relationships, and drug-drug-interactions) and indexes app. 21.4M abstracts and almost 360K full texts. Compared to other entity search engines in the field, it is either more comprehensive in terms of coverage of entities/relationships or provides information of higher quality (and in most cases both). For instance, our previous system Alibaba [PSP+06] had a similar coverage, but performed NER using dictionaries and RE using co-occurrence, both of which achieve suboptimal results. The system probably most similar to GeneView from an IE point-of-view is BioContext [GSBN12], which indexes only three different entity types. Furthermore, its IE capabilities are not integrated into a search engine. GeneView also has a number of features, which to our knowledge are not available in any other (biomedical) search engine. For instance, we support ranking of search results by entity counts. A user interested in mutations of a specific gene may search for this gene and then ranks the (probably many) results by the number of mutations they describe. Another unique feature is personalized ranking: Users may define their own gene lists and use the number of occurrences of genes from this list as criterion during search.

A general overview of GeneView including an intensive discussion of biological applications has been published elsewhere [TSV+12]. Here, we focus on the engineering challenges behind a search engine of the coverage, quality, and depth of GeneView. We believe that these challenges also are present in other domains and thus hope that sharing our experiences proves useful for many other researchers. A specific intention of this paper is to re-emphasize the complexity of high-quality information extraction, in contrast to many recent works which essentially consider IE problems (in their domain) to be solved and focus on merging, using, or querying extracted information.

2. User Interface

GeneView provides a user-friendly web-interface to make the extracted entity data searchable and accessible (see Figure 1). GeneViews search bar, which is provided at the top of every page, allows users to issue keyword queries on all available text documents. This includes entity-specific search for recognized entities using standard identifiers, e.g., Entrez gene ids for gene identification. The search bar offers an auto-completion

function to make it easier to find specific identifiers. For instance, typing BRCA1 into the search bar will bring up suggestions for several, species-dependent Entrez gene identifiers this short gene name corresponds to. To provide this functionality, GeneView uses a dictionary containing all entity mentions found in PubMed, each associated with their corresponding identifier. Additionally, the search form provides various options for result ranking and filtering. For instance, the user can choose to only include publications in the search result, which have been found to include certain types of entities (e.g., genes, SNPs, or chemicals). Figure 1 shows the result listing for a search for publications containing two specific genes identified by their Entrez gene id. The result is sorted by date of publication and has been filtered to only contain articles that also contain at least one SNP.

Figure 1: Result of a search for texts mentioning two specific genes, filtered for SNP content, sorted by date of publication.

Clicking on a search result shows the selected article together with all annotations (see Figure 2). Recognized entities are visualized by type-specific color highlighting. All entities are clickable to provide additional information such as link-outs to external reference databases. These pop-ups also provide links to search for content related to the selected entity. GeneView also provides an overview of all entities found in the article (Figure 2, left-hand bar). This is particularly helpful when dealing with full text papers containing multiple mentions for various entity types.

The above example of SNP-filtered searching for specific genes demonstrates one important use case of GeneView: The ability to use information about several types of biological entities in a single query both for ranking and for defining what constitutes the primary search result. With the given example, a user can easily retrieve all publications, which mention a mutation in the context of the given gene of interest. GeneView makes

such complex cross-entity searches a convenience. While GeneView extracts information about several types of entities to enable this type of multi-entity search, it does have special support for genes/proteins, where the on-click information contains links to several external reference databases of genes, pathways and protein-protein interactions. The pop-up also provides the option to search GeneView for articles describing PPIs in which the given gene/protein is found.

Figure 2: GeneViews single article view of PubMed ID abstract 21344391. Inline entity highlighting is complemented by an overview of entities found in the text (left-hand bar). Highlighted entities provide pop-ups with additional information from external databases.

3. Architecture and Pipeline

GeneView indexes all available articles from PubMed and PubMed Centrals open access set. Together with each articles text we store metadata such as authors, journal, MeSH terms, and figure/table captions that can be extracted by XML parsing from the original NCBI files. All texts are imported into Lucene (http://lucene.apache.org/core/), serving as storage, query, and ranking engine. Metadata and information about all recognized entities, especially type and Id of the entity and the exact position in the text, are stored in a relational database to allow structured retrieval (see Figure 3). Upon import, texts are processed by a custom text-mining pipeline that incorporates a multitude of tools for pre- and post-processing and for the entity-specific steps of NER, NEN and RE (see below). We decided not to use frameworks like UIMA, as most of our incorporated tools are not provided as UIMA components and would have required developing a proper wrapper. Furthermore, testing components inside of UIMA is, in our experience, extremely difficult.

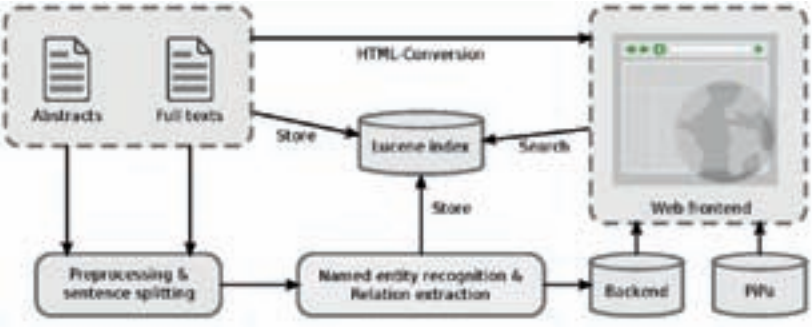


Figure 3. Architecture of GeneView.

Document preprocessing

All texts are downloaded from the National Library of Medicine (NLM) as XML. Available full text articles are converted into HTML for display in the GeneView web interface using XSLT scripts provided by NLM[1]. This transformation generates HTML representations resembling the PubMed Central visualization and thus enables a similar user experience. Subsequently, the articles' plain text is extracted: HTML specific characters like "&amp" are replaced with the corresponding UTF-8 symbol. HTML elements (e.g. </p> or <body>) are ignored and references at the end of the document are removed. Similarly, HTML tables are ignored. This extraction is necessary, as all text-mining steps require such clean text; in effect, we need to store each text twice, once for web display, and once for internal processing. This duplication seems to be inevitable, but generates additional problems when it comes to exactly addressing text snippets for syntax highlighting. Essentially, we need to maintain an exhaustive mapping of character positions from the cleansed text back to the HTML file. For articles without full text, i.e., usually PubMed abstracts, HTML is generated on the fly from the information stored in the Lucene index. Before starting the core information extraction pipeline, we detect sentence boundaries, section names, and abbreviations/long form mappings using the algorithm from [SH03]. Section names are identified using an approximate dictionary covering the 200 most often occurring section names[2]. This allows us to recognize 99.7% of all occurring section headings. We use this information for weighting search terms differently depending on the section of a document they appear in, a method which has proven highly effective in several works [DWH10; HRL05].

Named Entity Recognition and PPI extraction

The pre-processed texts are piped through a series of NER and RE tools (see Figure 4). These tools were selected using a best-of-breed strategy; some of them were developed

[1] ftp://ftp.ncbi.nih.gov/pub/archive_dtd/archiving/

[2]Note that section names in biomedical papers, in contrast to computer science, are highly standardized.

in house, some are external. We do not discuss those tools in detail here but refer to the original publications. The most important ones are (1) GNAT for gene and protein names [HGH+11], (2) MutationFinder for detecting SNPs [CBR+07], ChemSpot for chemicals [RWL12], and (4) Linnaeus for species names [GNB10]. Most of these tools use mixtures of machine learning algorithms (mostly CRFs) trained on gold standard corpora and exhaustive dictionaries of the respective entity type.

Figure 4. Pipeline of information extraction and NLP tools for creating the GeneView index.

The next step in the pipeline is relationship extraction. For this purpose, we use the freely available framework by Tikk et al. [TTP+10] which combines necessary NLP tools and a set of 13 different kernel-based RE methods. Of those, we use the two best performing algorithms (according to [TTP+10]), i.e., APG [APB+08] and SL [GLR06]. SL uses a SVM for classifying pairs of entities found in a sentence based on large bag-of-word-style feature vectors of the text surrounding the entities. APG applies a similar method, but uses a far larger vector including features derived from the dependency parse trees of the sentences. Therefore, sentences have to be parsed prior to the application of APG.

A persistent problem with using tools developed independently is that they require different input. Tools may require tokenized text, or may depend on unprocessed text because they perform their own tokenization. Similarly, some tools require text to be tagged with part-of-speech tags (POS), while others perform POS tagging themselves. Relationship extraction depends on results from sentence boundary detection, gene name recognition, part of speech recognition, and possibly constituent tree parsing and dependency parsing. Also, simple steps like abbreviation detection depend on preprocessing steps like sentence detection. On the other hand, tools also create different types of output which all need to be parsed and transformed into a uniform

representation. For instance, some NER tools create inline annotations, i.e., they output a new version of the input text with tags assigned to tokens, while others only create lists of detected entities with references into the text. These references may count tokens or characters; and may refer to different tokenizations and different treatment of special characters, which often requires a complicated re-mapping of detected entities.

Another problem in the application of text mining tools to large collections is their instability in terms of achieved performance. NER (and RE) tools typically are evaluated on small gold standard corpora (GSC) only, which are also used to train the systems. Accordingly, the obtained measures are only valid for these GSC. However, if a GSC has properties deviating substantially from the texts a tool is applied to, very different accuracies may be observed. When building a system like GeneView which annotates millions of texts, one immediately runs into this problem when inspecting some of the results. For instance, RE algorithms often are developed with GSC that contain a substantially higher fraction of true relationships than ordinary texts; this creates a tendency in classification-based methods to overestimate the a-priori probability of observing a relationship when judging an entity pair, which in turn leads to many false positives. We experimented with simply increasing the confidence threshold for PPI to reduce this problem, but yet did not find a satisfying solution.

In NER, this problem appears in two flavors. First, GSC often contain sets of sentences stemming from different abstracts. Second, most GSC draw their sentences only from abstracts and not from full text. As a consequence, effects of abbreviations are not properly represented (abbreviations are usually defined only once in a text and then used consistently), and the "one-sense-per-discourse" rule is not implemented in NER tools (meaning that a given, generally ambiguous, name usually is used in only one of its senses in a given text). We counteract this effect by two measures. First, when a NER tool tags a long (short) form of an abbreviation and we have detected the abbreviation itself, we also tag the respective short (long) form. Notably, this simple method adds 2.1 million additional gene terms. Second, when a NER tool tags a given token (or set of tokens) and we detect this token again in the same text, we also tag it. The effect of this trick is even more pronounced, as it adds 16.7 million additional gene annotations. These two post-processing steps together are responsible for 50.7% of all visualized gene mentions and have an enormous effect on the user-perceived recall and subsequent relationship extraction – yet a negligible effect when applied to an evaluation on GSC. However, the propagation again is not as simple as it appears, as one has to carefully decide when a subsequent match in a text is "good enough" for receiving an annotation. This is non-trivial, as, on one hand, names for the same gene may differ slightly (e.g. ABC-2 and ABC2 (Entrez Id 20) or TGD and TgD (Entrez Id 19)), while, on the other hand, slight variations in gene names may be decisive (e.g. "Fas" and "Fas-L").

Another problem of large-scale text mining is that some errors are only observed on a small subset of articles, which makes detecting them very hard. Examples are the following. (1) Our abbreviation detection algorithm has problems with different character encodings within the same article, a situation occurring extremely infrequently in PubMed. (2) Some of the NER tools occasionally tag trailing spaces, leading to inconsistencies in visualization. The XML format of PubMed is continuously modified, leading to unexpected parser break-downs (which are spotted immediately) or scrambled visualization (which we cannot detect automatically). (3) For full texts, we keep the XML provided by the publishers to support a journal-specific visualization, leading to diversity in, for instance, the way formulas are represented: Some journals integrate formulas as figures, whereas others enforce the use of MathML, which is removed by our parser in the cleansing step. (4) For dependency parsing, we apply the Charniak Lease parser [LC05] using the McClowsky reranking model [McCH06] which is unable to parse 14,618 out of the total number of 8,131,441 sentences. The reasons for its problems are not clear, yet; it is, however, noteworthy that the large majority (14,546) of problematic sentences came from full-text articles, although the majority of sentences are from abstracts. Again, the original parser is trained on sentences derived from abstracts, which are known to be different from full-text sentences [CJV+10]. This problem required changes in the source code, as the parser stopped after seeing a problematic sentence and did not continue parsing.

Processing step	**Time [Min]**	**Size [MB]**
Text indexing	1,211	77,855
HTML conversion	528	24,576
Gene NER	24,012	5,266
SNP NER	14,745	1,986
Histone modification NER	8,090	1,437
Chemical NER	1,272	16,539
Parsing	100,437	44,521
RE detection	11,520	29,483
DB import	3,858	-
Lookup information	1,849	53

Table 1. Requirements (single core) to create the main portions of the GeneView index.

Computational Requirements

GeneView is regularly updated using a server with 24 cores at 2.6 GHz and 256 GB main memory. Time intensive tasks, especially XML parsing, NER, syntactic parsing, and PPI extraction, are performed in parallel on chunks of the corpus. The computational requirements it takes to rebuild GeneView on a single core are shown in Table 1. Overall, running the entire pipeline in this mode would require an estimated time of 120 days. The by far most time intensive task is syntactic and dependency parsing, although we actually only parse those sentences which mention at least two genes. Of all our NER

tools, gene NER is the most time intensive due to its sophisticated disambiguation strategy responsible for mapping a gene mention to its correct database identifier (especially to the correct species). Overall disc space requirement is about 77GB for the Lucene index and 63GB for the metadata and result database.

4. Indexing Text and Entities

GeneView uses different technologies to store and index its content and to process queries: Lucene is used as a keyword search index and ranking engine; a relational database stores the structured annotation produced by the information extraction pipeline; and a web application interfaces the stored content to the user using the Catalyst MVC framework (http://www.catalystframework.org). While much of the functionality is provided off-the-shelve by the underlying systems, some features require special attention. An overview of all entities and relationships indexed in GeneView is given in Table 2.

Entity type	Entities	Distinct entities	Number of articles
Cell-type	18,891	231	5,622
Chemical	77,606,023	47,905	9,851,536
Disease	145,001	4,643	74,583
Drugs	47,113,224	3,061	6,246,067
Enzyme	894,895	2,298	590,301
Genes	37,080,749	83,705	2,959,439
Histone-mod	77,210	575	7,673
SNP	1,078,640	42,505	192,544
Species	44,808,988	115,966	9,119,134
Tissue	239	31	222
Overall	209,788,411	304,565	13,463,850

Table 2. Overview of detected entities in GeneView.

Document indexing and ranking

Ranking and filtering functions generally are implemented using Lucene. However, Lucene in the first place is not aware of the counts of detected entities and relationships within a document. Furthermore, ranking by entity-content is not a native feature of Lucene. To achieve this functionality, aggregated text-mining results for each article have to be propagated into the Lucene index and represented properly to integrate them into the customizable ranking mechanism. This encompasses the number of recognized distinct entities for each type as well as identifiers of recognized entities for each article section. The number of distinct entities of a specific type is used to filter articles without any entity of interest and to rank results by the number of distinct entities. The list of identifiers found in a specific article enables users to search for articles containing specifically this entity of interest (regardless of homonyms and synonyms).

For gene queries, the query relevance ranking is modified and a section specific ranking is applied. Optimal section weights have been determined using PubMed's gene2pubmed. Gene2pubmed provides manually curated links between PubMed articles and the genes contained in them. Using this data, we set weights as Lucene boost parameters such that a query for a curated gene in gene2pubmed ranks the corresponding articles in gene2pubmed highest. This strategy allows us to estimate and improve the mean average precision of gene queries. The automatically derived section weights meet general expectations in that, for instance, sections like *Title* are highly ranked, while *Materials and Methods* receive low weights. Technically, it would also be possible to extend this functionality to other types of entities; however, we currently see no sensible method to obtain rational weights for entities other than genes. Furthermore, the corresponding boosts would either interfere with each other or be provided separately at the user interface, which again would make it more complicated.

To allow users to focus on their particular set of genes, GeneView allows the definition of individual gene lists which later can be used to filter/rank articles of any query. In such cases, the query is expanded with the members of the gene set; implementing this feature therefore only requires functionality for storing and managing personalized gene lists, while their integration into the ranking can be achieved with standard Lucene methods. Note that achieving this functionality manually would be hard, as such gene lists often contain dozens or even hundreds of genes (in case of genetically complex diseases such as cancer or diabetes). It would be conceptually straight-forward to expand this feature to types of entities other than genes, but therein one carefully has to balance functionality and simplicity of the user interface.

Another feature of GeneView important for users is "rank by entity count". To this end, we extract aggregated counts from the database and store them as additional metadata in a proper Lucene field attached to each document. At query time, one can tell Lucene to use the information in this field for ranking and/or filtering. This solution works equally well for all types of counts; however, for usability reasons we currently expose this functionality only for SNPs and genes at the web interface.

Annotation indexing

All entities and relationships extracted by the extraction pipeline are stored in a relational database. Information stored for each entity mention includes the article id, normalized entity id, concrete annotated text span, start- and end-character position in the cleansed text. The article Id, which is the PubMed article identifier (PMID), links each mention to the corresponding document in the Lucene index. The normalized entity Id links a mention to additional information in external, type-specific data sources (e.g., Entrez gene Id for genes or Chebi Id for chemicals). The annotated text span and the start and end positions precisely define the actual occurrence of the entity in the inspected document. This information is used for entity highlighting when visualizing

single articles, which requires an additional step of mapping character positions as stored in the database to the HTML representation of the text created from the original XML files. Due to the multiple text manipulations that take place in-between, those mappings cannot be computed automatically; instead, we have to retain a positional mapping table for each inspected document (see Section 3). For all relationship types, we store links to the two linked entities, classifier confidence, and associated sentence.

Document specific aggregated information for each entity type is injected into the Lucene index once the NER/NEN/RE pipeline has finished. Thereby, Lucene can handle all ranking issues without a need to get back to the databases; the database is only accessed for highlighting during web display (see above) and for assisting users in formulating queries. Here, GeneView provides on-the-fly lookup functionality which suggests auto-completions if entered tokens match an entity name. This lookup issues one query to the database for each keystroke the user makes, which in turn requires a carefully indexed lookup table. We realize this lookup as a materialized view over the entity-specific annotation tables storing the original mention, its normalized representation and its corresponding identifier. Additionally, each entry contains the overall number of occurrences in the corpus.

Visualization in the Web Interface

For single article visualization, entities and their spans are requested from the relational database. For each type of entity found, a separate instance of the articles HTML representation is enriched with highlighting in a type-specific color. When displayed in the browser, these instances are overlaid to appear as a single document. The objective of this multi-layered approach is to allow collision free multi-entity annotation. For instance, a single entity may be (correctly) identified as both a drug and a chemical, causing two overlapping annotations. As GeneViews highlighting are semi-transparent, the resulting overlap of layers will appear to the user in a different, mixed color, indicating the detected ambiguity. A drawback is the need to transfer each text to the user, i.e., from server to client, multiple times within a single HTML document. While this is less problematic for abstracts, it does raise scalability issues for lengthy full texts in terms of the number of different entity types which can be included. For instance, GeneViews web page of a full text including five different types of entity mentions can reach a size of around 1MB.

5. Conclusions

We presented GeneView, an entity-centric search engine for the biomedical literature. To achieve its functionality, the system encompasses over two dozens of external NLP

and information extraction tools whose output are stored in a classical information retrieval engine (Lucene) and in a relational database (MySQL).

This paper gave an account of the many smaller and larger problems that emerge during the construction of systems like GeneView. Many of these problems stem from the fact that we follow a best-of-breed strategy, i.e., we use the best available tools for each of the different entity classes and relationship types that are indexed, which comes along with heterogeneous requirements in terms of execution environment, different data formats, multiple runtime dependencies, and continuous problems with version incompatibilities. In particular, the lack of standards for representing annotated texts, which gives rise to many different ways to link annotations with text spans, creates the need to perform repeated format conversions and to keep multiple copies of the text, along with brute-force mapping tables. Almost every tool in our pipeline has a different format for the input text and the positional annotations it returns. We currently see little hope that these problems will go away in the near future, unless efforts such as [HLAN12] succeed in defining standards for the community. As a positive message, we experienced that the basic infrastructures, especially Lucene, are able to provide stable, flexible and scalable search performance, although their usage for advanced features such as entity-based ranking requires some thought and effort.

However, we also see that a project like GeneView poses considerable challenges to current methods in terms of scalability, flexibility, and maintenance cost. For instance, the workflow depicted in Figure 4 can be executed in various orders, each of which will take different time depending on the selectivity of the contained filter operations, the time required to execute the various tools on input of varying size, the available hardware, etc. There have been first attempts to optimize such complex IE workflows mostly consisting of non-standard operations [RRK+08], but these focus on comparably simple operations like regular expression matching and co-occurrence. We believe that advanced methods like the ones implemented in GeneView have distinct properties calling for specific optimization techniques. We have started work in this direction [HRL+12] in the course of the Stratosphere project (http://www.stratosphere.eu/).

Another challenge is flexibility in executing an IE pipeline. Very often, only parts of the entire workflow have to be run, for instance if new versions of individual tools are available. In such cases, running the entire workflow would imply a great deal of unnecessary computations, but running only specific parts of it is not easily achieved, given that the workflow technically consists of a series of intertwined scripts in different languages. But because implementing sub-workflows is costly in terms of manpower, we often run the entire workflow despite the waste in compute power. A proper support for specifying and executing such pipelines should also support data incremental execution, as pipelines often break unexpectedly due to format problems in the input or bugs in the IE tools. Restarting the pipeline should not imply re-annotating texts that had already been finished in the previously though finally failed run. There exist some suggestions

towards this problem [KSB+10], but these, to the best of our knowledge, haven't been integrated into real dataflow languages yet.

Acknowledgements

We thank A. Rheinländer, S. Arzt, M. Neves, A. Vowinkel, and T. Rocktäschel for contributions to GeneView. We acknowledge funding from DFG GRK1651 (SOAMED) DFG LE 1428/4-1 (Stratosphere), and BMBF 0315417B (ColoNet).

References

[APB+08] Airola, A., Pyysalo, S., Bjorne, J., Pahikkala, T., Ginter, F. and Salakoski, T. (2008). "All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning." BMC Bioinformatics 9 Suppl 11: S2.

[Bal12] Balke, W.-T. (2012). "Introduction to Information Extraction: Basic Notions and Current Trends." Datenbank-Spektrum 12(2).

[BKS10] Bui, Q. C., Katrenko, S. and Sloot, P. M. (2010). "A hybrid approach to extract protein-protein interactions." Bioinformatics.

[CBR+07] Caporaso, J. G., Baumgartner, W. A., Randolph, D. A., Cohen, K. B. and Hunter, L. (2007). "MutationFinder: a high-performance system for extracting point mutation mentions from text." Bioinformatics 23(14): 1862-1865.

[CL10] Chowdhury, F. M. and Lavelli, A. (2010). "Disease Mention Recognition with Specific Features". Workshop on Biomedical Natural Language Processing, Uppsala, Sweden.

[CJV+10] Cohen, K. B., Johnson, H. L., Verspoor, K., Roeder, C. and Hunter, L. E. (2010). "The structural and content aspects of abstracts versus bodies of full text journal articles are different." BMC Bioinformatics 11: 492.

[DWH10] Divoli, A., Wooldridge, M. A. and Hearst, M. A. (2010). "Full text and figure display improves bioscience literature search." PLoS One 5(4): e9619.

[DMNL09] Dogan, R. I., Murray, G. C., Névéol, A. and Lu, Z. (2009). "Understanding PubMed® user search behavior through log analysis." Database (Oxford).

[GNB10] Gerner, M., Nenadic, G. and Bergman, C. M. (2010). "LINNAEUS: a species name identification system for biomedical literature." BMC Bioinformatics 11: 85.

[GSBN12] Gerner, M., Sarafraz, F., Bergman, C. M. and Nenadic, G. (2012). "BioContext: an integrated text mining system for large-scale extraction and contextualisation of biomolecular events." Bioinformatics 28(16): 2154-2161.

[GLR06] Giuliano, C., Lavelli, A. and Romano, L. (2006). "Exploiting shallow linguistic information for relation extraction from biomedical literature". European Chapter of the Association for Computational Linguistics Trento, Italy. pp 401–408.

[HGH+11] Hakenberg, J., Gerner, M., Haeussler, M., Solt, I., Plake, C., Schroeder, M., Gonzalez, G., Nenadic, G. and Bergman, C. M. (2011). "The GNAT library for local and remote gene mention normalization." Bioinformatics 27(19): 2769-2771.

[HRL05] Hakenberg, J., Rutsch, J. and Leser, U. (2005). "Tuning text classification for hereditary diseases with section weighting". Symposium on Semantic Mining in Biomedicine (SMBM), Hinxton, UK. pp 34-39.

[HRL+12] Heise, A., Rheinländer, A., Leicht, M., Leser, U. and Naumann, F. (2012). "Meteor/Sopremo: An Extensible Query Language and Operator Model". Workshop on End-to-end Management of Big Data, Istanbul, Turkey.

[HLAN12] Hellmann, S., Lehmann, J., Auer, S. and Nitzschke, M., Eds. (2012). "NIF Combinator: Combining NLP Tool Output". EKAW Galway City, Ireland.

[KSB+10] Koop, D., Santos, E., Bauer, B., Troyer, M., Freire, J. and Silva, C., T. (2010). "Bridging Workflow and Data Provenance Using Strong Link". Int. Conf. on Scientific and Statistical Database Management Systems, Heidelberg, Germany.

[KMS+08] Krallinger, M., Morgan, A., Smith, L., Leitner, F., Tanabe, L., Wilbur, J., Hirschman, L. and Valencia, A. (2008). "Evaluation of text-mining systems for biology: overview of the Second BioCreative community challenge." Genome Biol 9 Suppl 2: S1.

[LC05] Lease, M. and Charniak, E. (2005). "Parsing Biomedical Literature". Second International Joint Conference on Natural Language Processing (IJCNLP'05).

[Lu11] Lu, Z. (2011). "PubMed and beyond: a survey of web tools for searching biomedical literature." Database (Oxford) 2011: baq036.

[McCH06] McClosky, D., Charniak, E. and Johnson, M. (2006). "Reranking and self-training for parser adaptation". Int. Conf. on Computational Linguistics, Stroudsburg, USA. pp 337-344.

[OW04] Ogino, S. and Wilson, R. B. (2004). "Importance of standard nomenclature for SMN1 small intragenic ("subtle") mutations." Human Mutation 23(4): 392-393.

[PSP+06] Plake, C., Schiemann, T., Pankalla, M., Hakenberg, J. and Leser, U. (2006). "AliBaba: PubMed as a graph." Bioinformatics 22(19): 2444-5.

[RRK+08] Reiss, F., Raghavan, S., Krishnamurthy, R., Zhu, H. and Vaithyanathan, S. (2008). "An Algebraic Approach to Rule-Based Information Extraction". 24th International Conference on Data Engineering, Cancun, Mexico. pp 933-942.

[RWL12] Rocktäschel, T., Weidlich, M. and Leser, U. (2012). "ChemSpot: A Hybrid System for Chemical Named Entity Recognition." Bioinformatics 28(12): 1633-1640.

[SH03] Schwartz, A. S. and Hearst, M. A. (2003). "A simple algorithm for identifying abbreviation definitions in biomedical text". Pacific Symposium on Biocomputing, Hawaii, US.

[TNS+11] Thomas, P., Neves, M. L., Solt, I., Tikk, D. and Leser, U. (2011). "Relation Extraction for Drug-Drug Interactions using Ensemble Learning". DDIExtraction Workshop, Spain.

[TSV+12] Thomas, P., Starlinger, J., Vowinkel, A., Arzt, S. and Leser, U. (2012). "GeneView: A comprehensive semantic search engine for PubMed." Nucleic Acids Res 40(Web Server issue): 585-591.

[TTP+10] Tikk, D., Thomas, P., Palaga, P., Hakenberg, J. and Leser, U. (2010). "A comprehensive benchmark of kernel methods to extract protein-protein interactions from literature." PLOS Computational Biology 6(7).

[ZDF+08] Zweigenbaum, P., Demner-Fushman, D., Yu, H. and Cohen, K. B. (2007). "Frontiers of biomedical text mining: current progress." Brief Bioinform 8(5): 358-75.

Detecting Plagiarism in Text Documents through Grammar-Analysis of Authors

Michael Tschuggnall, Günther Specht

Institute of Computer Science
Databases and Information Systems
Technikerstraße 21a
6020 Innsbruck
michael.tschuggnall@uibk.ac.at
guenther.specht@uibk.ac.at

Abstract: The task of intrinsic plagiarism detection is to find plagiarized sections within text documents without using a reference corpus. In this paper, the intrinsic detection approach Plag-Inn is presented which is based on the assumption that authors use a recognizable and distinguishable grammar to construct sentences. The main idea is to analyze the grammar of text documents and to find irregularities within the syntax of sentences, regardless of the usage of concrete words. If suspicious sentences are found by computing the pq-gram distance of grammar trees and by utilizing a Gaussian normal distribution, the algorithm tries to select and combine those sentences into potentially plagiarized sections. The parameters and thresholds needed by the algorithm are optimized by using genetic algorithms. Finally, the approach is evaluated against a large test corpus consisting of English documents, showing promising results.

1 Introduction

1.1 Plagiarism Detection

Today more and more text documents are made publicly available through large text collections or literary databases. As recent events show, the detection of plagiarism in such systems becomes considerably more important as it is very easy for a plagiarist to find an appropriate text fragment that can be copied, where on the other side it becomes increasingly harder to correctly identify plagiarized sections due to the huge amount of possible sources. In this paper we present the Plag-Inn algorithm, a novel approach to detect plagiarism in text documents that circumvents large data comparisons by performing intrinsic data anaylsis.

The two main approaches for identifying plagiarism in text documents are known as *external* and *intrinsic* algorithms [PEBC+11], where external algorithms compare a suspicious document against a given, unrestricted set of source documents like the world wide web, and intrinsic methods inspect the suspicious document only. Often applied techniques

used in external approaches include n-grams [OLRV10] or word-n-grams [Bal09] comparisons, standard IR techniques like common subsequences [Got10] or machine learning techniques [BSL+04]. On the other side, intrinsic approaches have to comprise the writing style of an author in some way and use other features like the frequency of words from predefined word-classes [OLRV11], complexity analysis [SM09] or n-grams [Sta09, KLD11] as well to find plagiarized sections.

Although the majority of external algorithms perform significantly better than intrinsic algorithms by using the advantage of a huge data set gained from the Internet, intrinsic methods are useful when such a data set is not available. For example, in scientific documents that use information mainly from books which are not digitally available, a proof of authenticity is nearly impossible for a computer system to make. Moreover, authors may modify the source text in such a way that even advanced, fault-tolerant text comparison algorithms like the longest common subsequence [BHR00] cannot detect similarities. In addition, intrinsic approaches can be used as a preceding technique to help reduce the set of source documents for CPU- and/or memory-intensive external procedures.

In this paper the intrinsic plagiarism detection approach *Plag-Inn* (standing for *Plag*iarism Detection *Inn*sbruck) is described, which tries to find plagiarized paragraphs by analyzing the grammar of authors. The main assumption is that authors have a certain *style* in terms of the grammar used, and that plagiarized sections can be found by detecting irregularities in their style. Therefore each sentence of a text document is parsed by its syntax, which results in a set of grammar trees. These trees are then compared against each other, and by using a Gaussian normal distribution function, sentences that differ significantly according to its building syntax are marked as suspicious.

The rest of this paper is organized as follows: the subsequent paragraph explains and summarizes the intrinsic plagiarism detection algorithm *Plag-Inn*, whereby Section 2 describes in detail how suspicious sentences are selected. The optimization of the parameters used in the algorithm is shown in Section 3 and an extensive evaluation of the approach is depicted in Section 4. Finally, Sections 5 and 6 discuss related work and conclude with the Plag-Inn algorithm by summarizing the results and showing future work, respectively.

1.2 The Plag-Inn Algorithm

The *Plag-Inn* algorithm is a novel approach (the main idea and a preliminary evaluation has been sketched in [TS12]) in the field of intrinsic plagiarism detection systems which tries to find differences in a document based on the stylistic changes of text segments. Based on the assumption that different authors use different grammar rules to build their sentences it compares the grammar of each sentence and tries to expose suspicious ones. For example, the sentence[1]

(1) *The strongest rain ever recorded in India shut down the financial hub of Mumbai, officials said today.*

[1] example taken and modified from the Stanford Parser website [Sta12]

could also be formulated as

(2) *Today, officials said that the strongest Indian rain which was ever recorded forced Mumbai's financial hub to shut down.*

which is semantically equivalent but differs significantly according to its syntax. The grammar trees produced by these two sentences are shown in Figure 1 and Figure 2, respectively. It can be seen that there is a significant difference in the building structure of each sentence. The main idea of the approach is to quantify those differences and to find outstanding sentences or paragraphs which are assumed to have a different author and thus may be plagiarized.

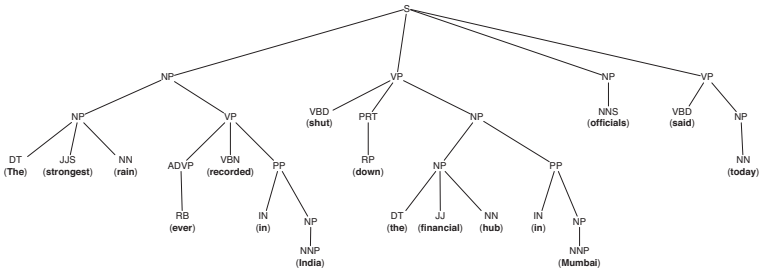


Figure 1: Grammar Tree Resulting From Sentence (1).

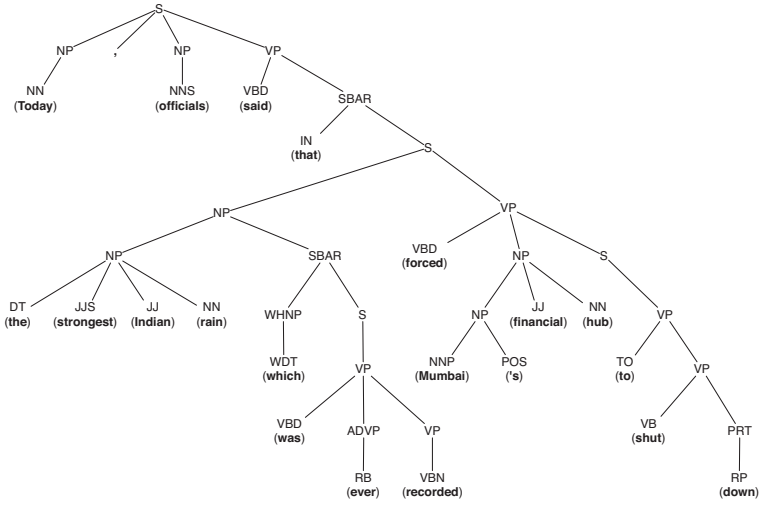


Figure 2: Grammar Tree Resulting From Sentence (2).

The Plag-Inn algorithm consists of five basic steps:

1. At first the given text document is parsed and split into single sentences by using Sentence Boundary Detection algorithms [SG00].

2. Then, the grammar is parsed for each sentence, i.e. the syntax of how the sentence was built is extracted. With the use of the open source tool *Stanford Parser* [KM03] each word is labelled with Penn Treebank tags [MMS93], that for example correspond to word-level classifiers like verbs (VB), nouns (NN) or adjectives (JJ), or phrase-level classifiers like noun phrases (NP) or adverbial phrases (ADVP). Finally, the parser generates a grammar syntax tree as it can be seen in e.g. Figure 1. Since the actual words in a sentence are irrelevant according to the grammatical structure, the leaves of each tree (i.e. the words) are dismissed.

3. Now, having all grammar trees of all sentences, the distance between each pair of trees is calculated and stored into a triangular distance matrix D:

$$D_n = \begin{pmatrix} d_{1,1} & d_{1,2} & d_{1,3} & \cdots & d_{1,n} \\ d_{1,2} & d_{2,2} & d_{2,3} & \cdots & d_{2,n} \\ d_{1,3} & d_{2,3} & d_{3,3} & \cdots & d_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{1,n} & d_{2,n} & d_{3,n} & \cdots & d_{n,n} \end{pmatrix} = \begin{pmatrix} 0 & d_{1,2} & d_{1,3} & \cdots & d_{1,n} \\ * & 0 & d_{2,3} & \cdots & d_{2,n} \\ * & * & 0 & \cdots & d_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \cdots & 0 \end{pmatrix}$$

   Thereby, each distance $d_{i,j}$ corresponds to the distance of the grammar trees between sentence i and j, where $d_{i,j} = d_{j,i}$. The distance itself is calculated using the pq-gram distance [ABG10], which is shown to be a lower bound of the more costly, fanout weighted tree edit distance [Bil05]. A pq-gram is defined through the base p and the stem q, which define the number of nodes taken into account vertically (p) and horizontally (q). Missing nodes, e.g. when there are less than q horizontal neighbors, are marked with $*$. For example, using $p = 2$ and $q = 3$, valid pq-grams of the grammar tree shown in Figure 1 would be {*S-NP-NP-VP-**} or {*S-VP-VBD-PRT-NP*} among many others. The pq-gram distance is finally calculated by comparing the sets PQ_i and PQ_j which correspond to the set of pq-grams of the grammar trees of the sentences i and j, respectively.

   The distance matrix of a document consisting of 1500 sentences is visualized in Figure 3, whereby the triangular character of the matrix is ignored in this case for better visibility. The z-axis represents the pq-gram-distance between the sentences on the x- and y-axis, and it can be seen that there are significant differences in the style of sentences around number 100 and 800, respectively.

4. Significant differences which are already visible to a human eye in the distance matrix plot are now examined through statistical methods. To find significantly outstanding sentences, i.e. sentences that might have been plagiarized, the median distance for each row in D is calculated. The resulting vector

$$\bar{d} = (\bar{d}_1, \bar{d}_2, \bar{d}_3, \ldots, \bar{d}_n)$$

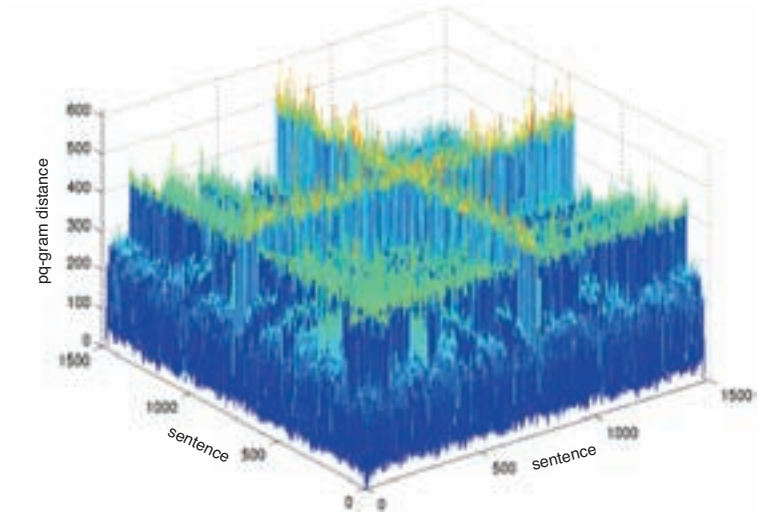


Figure 3: Distance Matrix of a Sample Document Consisting of about 1500 Sentences.

is then fitted to a Gaussian normal distribution which estimates the mean value μ and the standard deviation σ. The two Gaussian values can thereby be interpreted as a common variation of how the author of the document builds his sentences grammatically.

Finally, all sentences that have a higher difference than a predefined threshold δ_{susp} are marked as suspicious. The definement and optimization of δ_{susp} (where $\delta_{susp} \gg \mu + \sigma$) is shown in Section 3. Figure 4 depicts the mean distances resulting from averaging the distances for each sentence in the distance matrix D. After fitting the data to a Gaussian normal distribution, the resulting mean μ and standard deviation σ are marked in the plot. The threshold δ_{susp} that splits ordinary from suspicious sentences can also be seen, and all sentences exceeding this threshold are marked.

5. The last step of the algorithm is to smooth the results coming from the mean distances and the Gaussian fit algorithm. At first, suspicious sentences that are close together with respect to their occurence in the document are grouped into paragraphs. Secondly, standalone suspicious sentences might be dropped because it is unlikely in many cases that just one sentence has been plagiarized. Details on how sentences are selected for the final result are presented in Section 2.

2 Selecting Suspicious Sentences

The result of steps 1-3 of the Plag-Inn algorithm is a vector $\bar{d}$ of size n which holds the average distances of each sentence to all other sentences. After fitting this vector to a

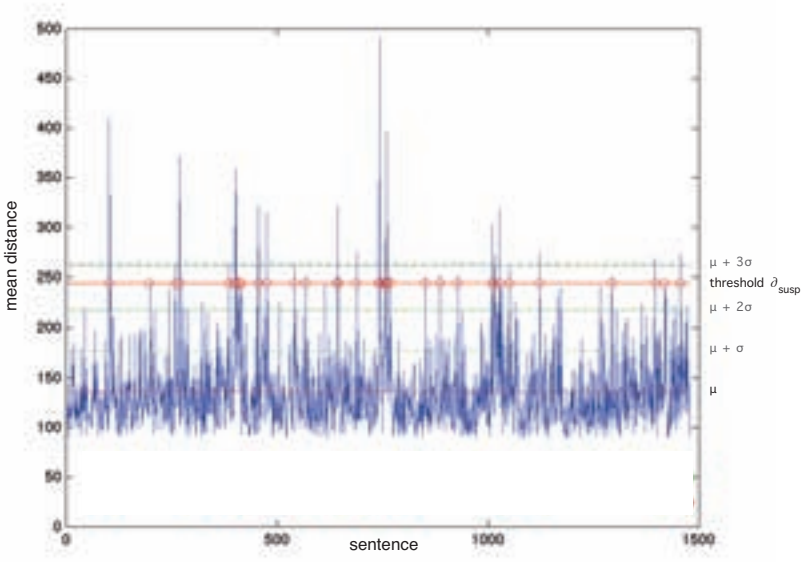


Figure 4: Mean Distances Including the Gaussian-Fit Values μ and σ.

Gaussian normal distribution as it is shown in Section 1.1, all sentences having a higher average distance than the predefined threshold δ_{susp} are marked as suspicious. This is the first step as can be seen in Algorithm 1.

The main objective of the further procedure of the sentence-selection algorithm is to group together sentences into plagiarized paragraphs and to eliminate standalone suspicious sentences. As the Plag-Inn algorithm is based on the grammatical structure of sentences, short instances like *"I like tennis."* or *"At what time?"* carry too less information and are most often not marked as suspicious as their building structure is too simple. Nevertheless, such sentences may be part of a plagiarized section and should therefore be detected. For example, if eight sentences in a row have found to be suspicious except one in the middle, it is intuitively very likely that it should be marked as suspicious as well.

To group together sentences, the algorithm shown in Algorithm 1 traverses all sentences in sequential order. If it finds a sentence that is marked suspicious, it first creates a new plagiarized section and adds this sentence. As long as ongoing suspicious sentences are found they are added to the section. When a sentence is not suspicious, the global idea is to use a lookahead variable ($curLookahead$) to step over non-suspicious sentences and check if there is a suspicious sentence *nearby*. If a sentence is then found to be suspicious within a predefined maximum ($maxLookahead$), this sentence and all non-suspicious sentences in between are added to the plagiarized section, and the lookahead variable is reset. Otherwise if this maximum is exceeded and no suspicious sentences are found, the current section is closed and added to the final result.

After all sentences are traversed, plagiarized sections in the final set R that contain only one sentence are checked to be filtered out. Intuitively this step makes sense as it can be assumed that authors do not copy only one sentence in a large paragraph. Within the evaluation of the algorithm described in Section 4 it could additionally be observed that single-sentence sections of plagiarism are often the result of wrongly parsed sentences coming from noisy data. To ensure that these sentences are filtered out, but *strongly* plagiarized single sentences remain in the result set, another threshold is introduced. In this sense, δ_{single} defines the average distance threshold that has to be exceeded by sections that contain only one sentence in order to remain in the result set R.

As the optimization of parameters (Section 3) showed, the best results can be achieved when choosing $\delta_{single} > \delta_{susp}$, which strengthens the intuitive assumption that a single-sentence section has to be *really* different. Controversially, genetic algorithms described in Section 3.2 also generated parameter optimization results that recommend to not filter single-sentence sections at all.

An example on how the algorithm works can be seen in Figure 5. Diagram (a) shows all sentences, where all instances with a higher mean distance than δ_{susp} have been marked as suspicious previously. When reaching suspicious sentence 5, it is added to a newly created section S. After adding sentence 6 to S, the lookahead variable is incremented as sentence 7 is not suspicious. Reaching sentence 8 which is suspicious again, both sentences are added to S as can be seen in Diagram (b). This procedure continues until the maximum lookahead is reached, which can be seen in Diagram (c). As the last sentence is not suspicious, S is closed and added to the result set R. Finally, Diagram (d) shows the final result set after eliminating single-sentence sections. As can be seen, sentence 14 has been filtered out as its mean difference is less than δ_{single}, whereas sentence 20 remains in the final result R.

3 Parameter Optimization

To evaluate and optimize the Plag-Inn algorithm including the sentence-selection algorithm, the PAN 2011 test corpus [PSBCR10] has been used which contains over 4000 English documents. The documents consist of a various number of sentences, starting from short texts from e.g. 50 sentences up to novel-length documents of about 7000 sentences. About 50% of the documents contain plagiarism, varying the amount of plagiarized sections per document, while the other 50% are left originally and contain no plagiarized paragraphs.

Most of the plagiarism cases are built by copying text fragments from other documents and subsequently inserting them in the suspicious document, while manual obfuscation of the inserted text is done additionally in some cases. Also, some plagiarism cases have been built using copying and translating from other source-languages like Spanish or German. Finally, for every document there exists a corresponding annotation file which can be consulted for an extensive evaluation.

As depicted in the previous section the sentence-selection algorithm relies on various input

Algorithm 1 Sentence Selection Algorithm

input:

d_i	mean distance of sentence i
δ_{susp}	suspicious sentence threshold
δ_{single}	single suspicious sentence threshold
$maxLookahead$	maximum lookahead for combining non-suspicious sentences
$filterSingles$	indicates whether sections containing only one sentence should be filtered out

variables:

$susp_i$	indicates whether sentence i is suspicious or not
R	final set of suspicious sections
S	set of sentences belonging to a suspicious section
T	temporary set of sentences
$curLookahead$	used lookaheads

set $susp_i \leftarrow false$ for all i, $R \leftarrow \emptyset, S \leftarrow \emptyset, T \leftarrow \emptyset$, $curLookahead \leftarrow 0$
for i from 1 to n **do**
    if $d_i > \delta_{susp}$ **then** $susp_i \leftarrow true$
end for

for i from 1 to number of sentences **do** ▷ traverse sentences
    if $susp_i = true$ **then**
        if $T \neq \emptyset$ **then**
            $S \leftarrow S \cup T$ ▷ add all non-suspicious sentences in between
            $T \leftarrow \emptyset$
        end if
        $S \leftarrow S \cup \{i\}$, $curLookahead \leftarrow 0$
    else if $S \neq \emptyset$ **then**
        $curLookahead \leftarrow curLookahead + 1$
        if $curLookahead \leq maxLookahead$ **then**
            $T \leftarrow T \cup \{i\}$ ▷ add non-suspicious sentence i to temporary set T
        else
            $R \leftarrow R \cup \{S\}$ ▷ finish section S and add it to the final set R
            $S \leftarrow \emptyset, T \leftarrow \emptyset$
        end if
    end if
end for
if $filterSingles = true$ **then** ▷ filter single-sentence sections
    for all plagiarized sections S of R **do**
        if $|S| = 1$ **then**
            $i \leftarrow$ the (only) element of S
            if $d_i < \delta_{single}$ **then** $R \leftarrow R \setminus \{S\}$
        end if
    end for
end if

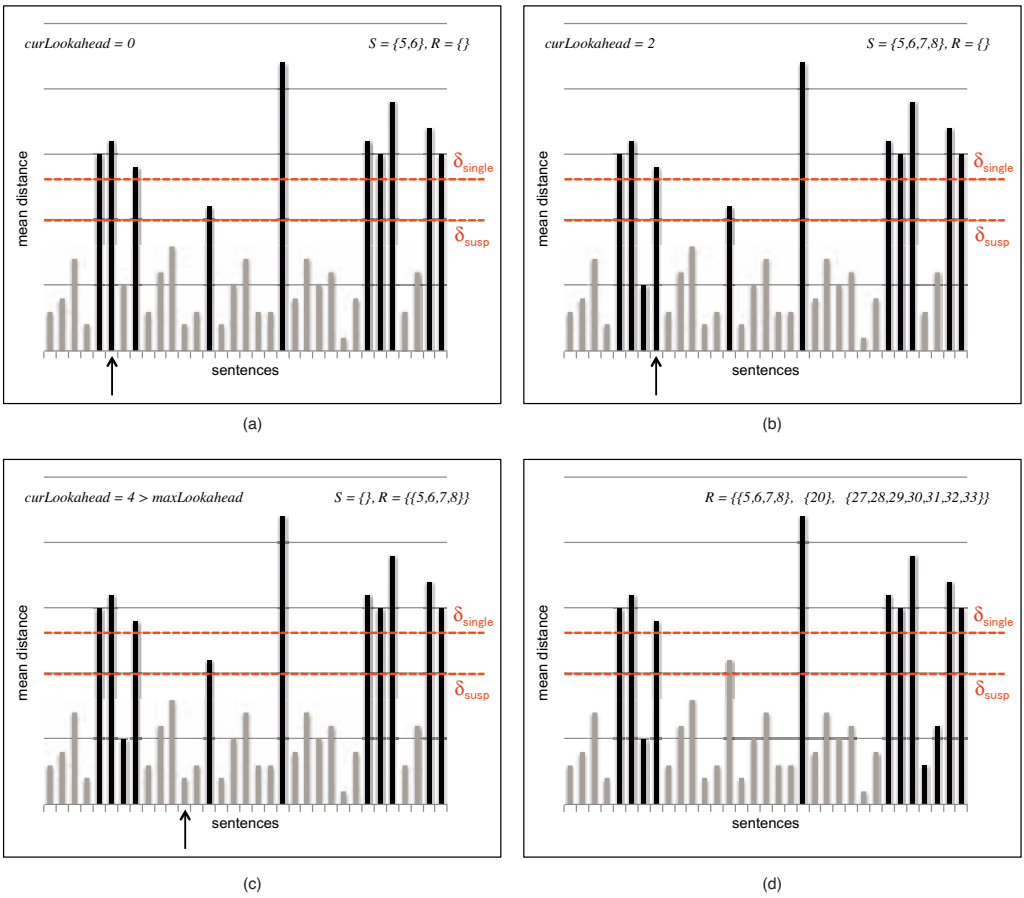


Figure 5: Example of the sentence-selection algorithm.

variables:

- δ'_{susp}: suspicious sentence threshold. Every sentence that has a higher mean distance is marked as suspicious.
- δ'_{single}: single suspicious sentence threshold. Every sentence in a single-sentence plagiarized section that is below this threshold is unmarked in the final step of the algorithm.
- $maxLookahead$: maximum lookahead. Defines the maximum value of checking if there is a suspicious sentence occuring after non-suspicious sentences that can be included into the current plagiarized section.
- $filterSingles$: boolean switch that indicates whether sections containing only one sentence should be filtered out. If $filterSingles = true$, the single suspicious

sentence threshold δ_{single} is used to determine whether a section should be dropped or not.

Thereby, the values for the thresholds δ'_{susp} and δ'_{single}, respectively, represent the inverse probability range of the Gaussian curve that include sentences with a mean distance that is not marked as suspicious. For example, $\delta'_{susp} = 0.9973$ would imply that $\delta_{susp} = \mu + 3\sigma$, meaning that all sentences having a higher average distance than $\mu + 3\sigma$ are marked as suspicious. In other words, one would find 99.73% of the values of the Gaussian normal distribution within a range of $\mu \pm 3\sigma$. In Figure 4, δ_{susp} resides between $\mu + 2\sigma$ and $\mu + 3\sigma$.

In the following the optimization techniques are described which should help finding the parameter configuration that produces the best result. To achieve this, predefined static configurations have been tested as well as genetic algorithms have been used. Additionally, the latter have been applied to find optimal configurations on two distinct document subsets that have been splitted by the number of sentences (see Section 3.3).

All configurations have been evaluated using the common IR-measures *recall*, *precision* and the resulting harmonic mean *F-measure*. In this case recall represents the percentage of plagiarized sections found, and precision the percentage of correct matches, respectively. In order to compare this approach to others, the algorithm defined by the PAN workshop [PSBCR10] has been used to calculate the according values[2].

3.1 Static Configurations

As a first attempt, 450 predefined static configurations have been created by building all[3] permutations of the values in Table 1.

Parameter	Range
δ'_{susp}	$[0.994, 0.995, \ldots, 0.999]$
δ'_{single}	$[0.9980, 0.9985, \ldots, 0.9995]$
$maxLookahead$	$[2, 3, \ldots, 16]$
$filterSingles$	$[\texttt{yes}, \texttt{no}]$

Table 1: Configuration Ranges for Static Parameter Optimization.

The five best evaluation results using the static configurations are shown in Table 2. It can be seen that all of the configurations make use of the single-sentence filtering, using almost the same threshold of $\delta'_{single} = 0.9995$. Surprisingly, the maximum lookahead with values

[2]Note that the PAN algorithms of calculating recall and precision, respectively, are based on plagiarized sections rather than plagiarized characters, meaning that if an algorithm detects 100% of a long section but fails to detect a second short section, the F-measure can never exceed 50%. Calculating the F-measure character-based throughout the Plag-Inn evaluation resulted in an increase of about 5% in all cases.

[3]In configurations where single-sentence sections are not filtered, i.e. $filterSingles = no$, permutations originating from the values of δ'_{single} have been ignored.

from 13 to 16 are quite high. Transformed to the problem definition and the sentence-selection algorithm, this means that the best results are achieved when sentences can be grouped together in a plagiarized section while stepping over up to 16 non-suspicious sentences.

As it is shown in the following section, genetic algorithms produced a better parameter configuration using a much lower maximum lookahead.

δ_{susp}	$maxLookahead$	$filterSingles$	δ_{single}	**Recall**	**Precision**	**F-Measure**
0.995	16	yes	0.9995	0.159	0.150	**0.155**
0.994	16	yes	0.9995	0.161	0.148	**0.155**
0.996	16	yes	0.9995	0.154	0.151	**0.153**
0.997	15	yes	0.9995	0.150	0.152	**0.152**
0.995	13	yes	0.9990	0.147	0,147	**0.147**

Table 2: Best Evaluation Results using Static Configuration Parameters.

3.2 Genetic Algorithms

Since the evaluation of the documents of the PAN corpus is computationally intensive, a better way than just evaluating fixed static configurations is to use genetic algorithms [Gol89] to find optimal parameter assignments.

Genetic algorithms emulate biological evolutions, implementing the principle of the *"Survival of the fittest"*[4]. In this sense genetic algorithms are based on *chromosomes* which consist of several *genes*. A gene can thereby be seen as a parameter, whereas a chromosome represents a set of genes, i.e. the parameter configuration. Basically, a genetic algorithm consists of the following steps:

1. Create a ground-population of chromosomes of size p.
2. Randomly assign values to the genes of each chromosome.
3. Evaluate the *fitness* of each chromosome, i.e. evaluate all documents of the corpus using the parameter configuration resulting from the individual genes of the chromosome.
4. Keep the fittest 50% of the chromosomes and alter their genes, i.e. alter the parameter assignments so that the population size is p again. Thereby the algorithms recognize whether a change in any direction lead to a fitter gene and takes it into account when altering the genes [Gol89].
5. If the predifined number of evolutions e is reached then stop the algorithm, otherwise repeat from step 3.

[4]The phrase was originally stated by the British philosopher Herbert Spencer.

With the use of genetic algorithms significantly more parameter configurations could be evaluated against the test corpus. Using the JGAP-library[5], which implements genetic programming algorithms, the parameters of the sentence-selection algorithm have been optimized. As the algorithm needs a high amount of computational effort and to avoid overfitting, random subsets of 1000 to 2000 documents have been used to evaluate each chromosome, whereby these subsets have been randomized and renewed for each evolution. As can be seen in Table 3 the results outperform the best static configuration with an F-measure of about 23%.

What can be seen in addition is that the best configuration gained from using a population size of $p = 400$ recommends to not filter out single-sentence plagiarized sections, but to rather keep them.

p	δ_{susp}	$maxLook.$	$filterSingles$	δ_{single}	**Recall**	**Precision**	**F**
400	0.999	4	no	-	0.211	0.257	**0.232**
200	0.999	13	yes	0.99998	0.213	0.209	**0.211**

Table 3: Parameter Optimization Using Genetic Algorithms.

3.3 Genetic Algorithms On Document Subsets

By a manual inspection of the individual results for each document of the test corpus it could be seen that in some configurations the algorithm produced very good results on short documents, while on the other hand it produced poor results on longer, novel-length documents. Additionally when using other configurations, the F-measure results of longer documents were significantly better.

To make use of the assumption that different length documents should be treated differently, the test corpus has been split by the number of sentences in a document. For example, when using 150 as splitting number, the subsets $S_{<150}$ and $S_{\geq 150}$ have been created, containing all documents that have less than 150 sentences and containing all documents that have more than or exactly 150 sentences, respectively. Then, for each of the two subsets, the optimal parameter configuration has been evaluated using genetic algorithms as it is described in Section 3.2. Like before, a random number of documents from 1000 to 2000 has been used to evaluate a chromosome.

Table 4 shows the best configurations produced by genetic algorithms using the sentence-split document subsets. For the dividing number 100, 150 and 200 sentences per document have been choosen as this seemed to be the optimal range found by manual inspection (as discussed in Section 6 this could be improved in future work).

With an F-measure of about 50% on the short-documents subset and 21% on the long-documents subset the sentence-split evaluation worked best with a splitting number of 100 and a resulting overall F-measure of 35.7%. All configurations significantly outperform

[5]http://jgap.sourceforge.net, visited October 2012

the genetic algorithm optimization described earlier, in the best case over 12%. Moreover, what can be seen in all configurations is that the short-documents subsets could be optimized significantly better, resulting in F-values of about 45% to 50%. As discussed later, this already indicates that the algorithm is well suited for short documents.

subset	δ_{susp}	$maxLook.$	$filterSingles$	δ_{single}	**Recall**	**Precision**	**F**
$S_{\geq 100}$	0.998	6	yes	0.9887	0.205	0.216	**0.210**
$S_{<100}$	0.999	1	no	-	0.501	0.508	**0.504**
					0.353	0.362	**0.357**
$S_{\geq 150}$	0.963	9	yes	0.9993	0.118	0.109	**0.113**
$S_{<150}$	0.999	4	no	-	0.494	0.478	**0.486**
					0.306	0.294	**0.300**
$S_{\geq 200}$	0.963	10	yes	0.9998	0.108	0.115	**0.111**
$S_{<200}$	0.999	2	yes	0.9999	0.441	0.457	**0.449**
					0.275	0.286	**0.280**

Table 4: Parameter Optimization Using Genetic Algorithms on Document Subsets Splitted by the Number of Sentences.

4 Evaluation

The Plag-Inn algorithm including the sentence-selection algorithm have been evaluated using the PAN 2011 test corpus which has been described in more detail in Section 3. It consists of more than 4000 English documents that randomly contain plagiarism cases.

The following section shows an extensive evaluation of the Plag-Inn approach using the sentence selection algorithm. All results are based on the optimal configuration gained from using the genetic algorithms over the whole test corpus as described in Section 3.2. Although the genetic algorithms optimized for the document subsets split by text length produced significant better results, it is more realistic to use just one parameter configuration in order to avoid overfitted results (manual inspection showed that small documents contained significantly less plagiarism cases in the test corpus).

Figure 6 shows the overall evaluation result with an F-measure of 23%[6]. It can be seen that plagiarism cases created by translation could be detected better than those created artificially[7]. This result is as expected because the grammar of another language is always different than the target language (English), and translation - which is done by automatic algorithms in most cases - produces changes in grammar that can be detected more easily.

The fourth column shows the results for documents where at least 75% of the plagiarism

[6]The currently (2012) best intrinsic plagiarism detector achieves an F-measure of about 33% over the same test corpus [OLRV11]. As shown, using the best parameter configuration on document subsets we achieve even 35%.

[7]*artificially* means that a source fragment has been copied and modified by computer algorithms.

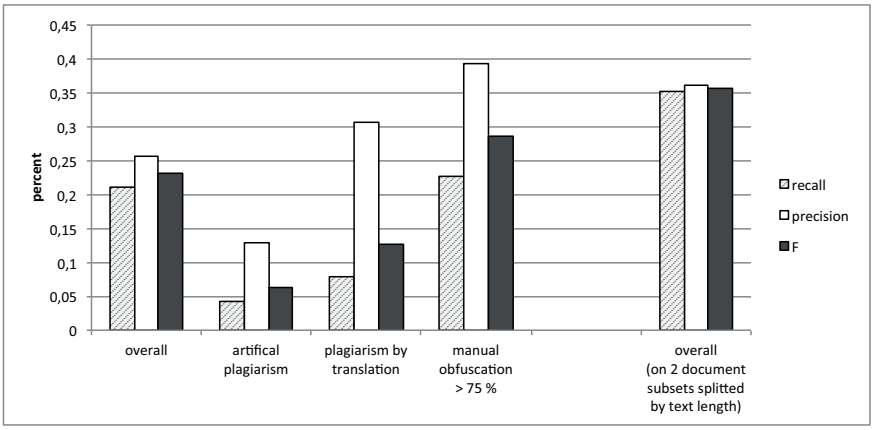


Figure 6: Overall Evaluation Results.

cases have been built by copying and subsequently doing manual obfuscation by hand. The F-measure for those cases almost reaches 30%, which indicates that the approach works very well for *real* plagiarism cases, i.e. cases that were created by humans rather than by computer algorithms. Finally, for comparison the results gained from the best parameter configurations on two document subsets splitted by the number of sentences contained is shown in the last column. As stated before, the best result of about 35% could be achieved by using a splitting number of 100.

In all results the precision is higher than the recall, meaning that the approach is better in interpreting suspicious passages than in finding them.

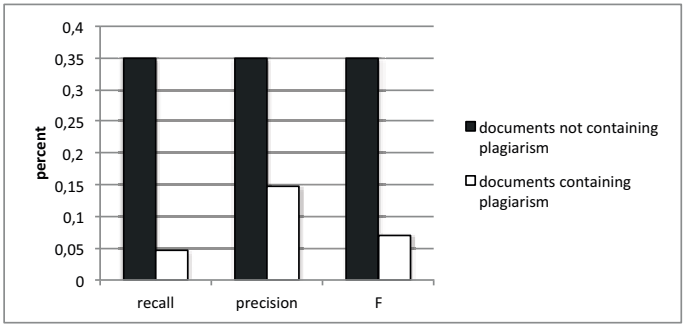


Figure 7: Evaluation Results for Documents Containing and Not Containing Plagiarism Cases.

The Plag-Inn approach achieves significantly different results on documents that contain plagiarism and on documents that do not contain plagiarism, as it is shown in Figure 7. Thereby, *clean* documents are processed very well and with balanced values for recall, precision and the according F-value of about 35%, respectively. On the other side, documents

containing plagiarized sections are obviously more difficult to detect for the algorithm, while the precision is consistently higher than the recall as experienced before.

Evaluations show that, the shorter documents get the better the algorithm works. Like it is illustrated in Figure 8, an F-measure of over 40% could be achieved on documents containing less than 300 sentences. Moreover as stated before, the algorithm performs steadily better when documents get shorter, reaching an F-measure of nearly 70% on very short documents containing less than 50 sentences. As the algorithm uses grammatical inconsistencies to find plagiarized sections, it might be that the variety of sentence syntaxes is too high in long documents, such that the algorithm fails frequently and produces the overall result of only 23%.

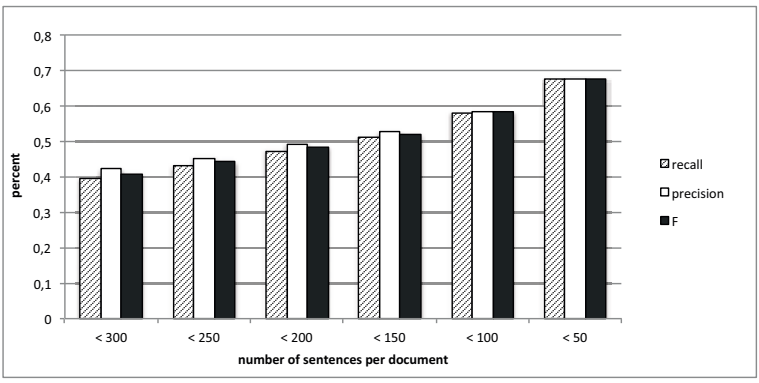


Figure 8: Evaluation Results for Short Documents.

What could be found in addition is that the approach is sensitive to the number of plagiarized sections per document as it is shown in Figure 9. Here, all documents that contain plagiarism have been inspected concerning the concrete number of plagiarism cases per document. It can be seen that the more sections of a document have been plagiarized, the better the results get. I.e. the more an author steals in his work, the more likely it is that it is detected by the algorithm.

Finally, the best option to improve the approach in future work is to reduce the number of false-positives, which is depicted in Figure 10. Diagram (a) shows the number of false-positives and false-negatives, respectively, where over 35% of all test corpus documents have wrongly been marked as plagiarized. On the horizontal axis the number of detected plagiarized sections for false-positives, and the number of not detected sections for false-negatives are shown. For about half of the wrongly predicted false-positives, the algorithm detected less than 5 plagiarism cases, and for about 10% of the documents the algorithm detected only one plagiarized section. This means that improving the algorithm in a way such that the false-positives that contain only one suspicious passage could be reduced or eliminated, this would lead to a significant better overall result.

In diagram (b) the percentage of predictions is shown. According to the high number of false-positives, the algorithm predicted too much for about half of the documents, i.e. it

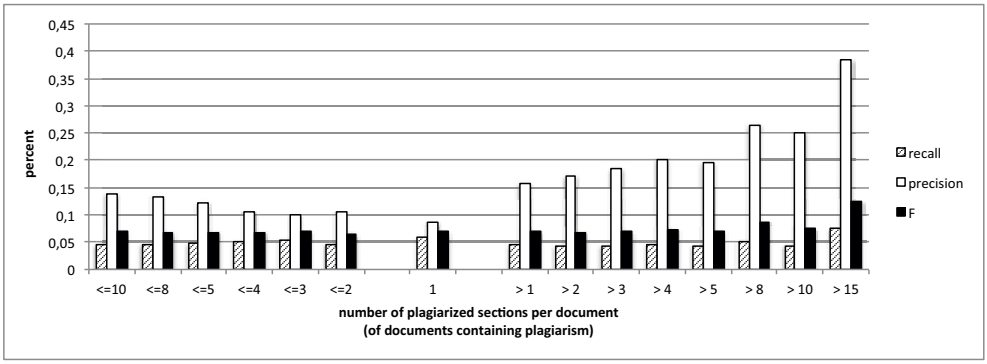
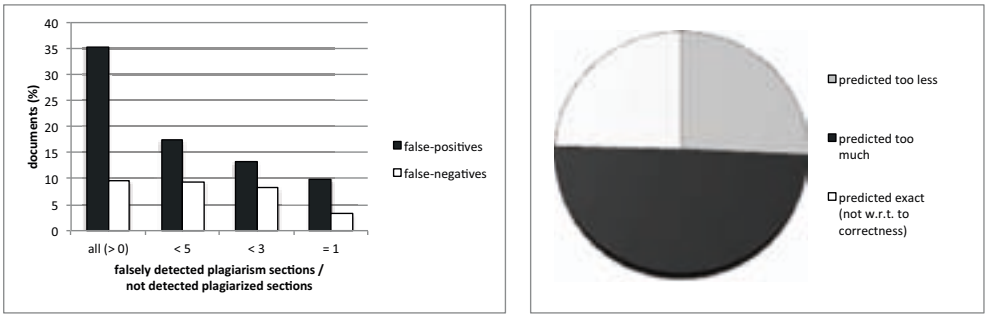


Figure 9: Evaluation Results Correlated to Number of Plagiarized Sections per Document.

detected plagiarized sections where there originally were less or even none. For the other half, too less or the exact number of sections have been detected. It has to be stressed that the amount of exact predictions in terms of plagiarized sections does not necessarily correspond to the number of *correct* detections. For example, the algorithm might have found four plagiarized sections in a document that contained exactly four plagiarized sections, but it might be the case that only two of them are correct. Nevertheless, manual inspections of the results have shown that the majority of exact predictions really correspond to the correct sections.



(a) (b)

Figure 10: False-Positives, False-Negatives and General Prediction Statistics.

5 Related Work

An often used technique in intrinsic plagiarism detection algorithms are n-grams [Sta09, KLD11], where a text document is broken into sets of two-, three- or four-letter chunks

and subsequently analyzed by their number of occurences within sliding windows. With the additional use of a style change function [Sta09] could reach a recall and precision value of about 33% over the PAN11 test corpus. Another approach also uses the sliding window technique but is based on word frequencies and the assumption that authors use a significant set of words [OLRV11].

An approach that tries to recognize paraphrased sections based on the phrase structure of sentences and the structure of verb phrases is described in [UKN05]. In this work, sentence-initial and -final phrases are inspected together with predefined semantic classes of verbs [Lev93] by part-of-speech tagging. It uses POS-tags only, i.e. without referring to computationally intense full grammar tree parses.

[SM09] discusses the comparison of binary strings calculated from word groups like nouns or verbs using complexity analysis. Approaches in the field of author detection and genre categorization also use NLP tools to analyze documents based on syntactic annotations [SKF00], but do not use grammar trees for comparisons. Word- and text-based statistics like the average sentence length or the average parse tree depth are used in [Kar00].

6 Conclusion and Future Work

In this paper the intrinisic plagiarism detection approach *Plag-Inn* is described: it aims to find plagiarism in text documents by inspecting the suspicious document only. The main idea is to analyze grammatical structures that authors use to build sentences, and to find inconsistencies in the syntax. After inconsistencies have been found by using Gaussian normal distribution fitting, an algorithm that selects and combines suspicious sentences is presented.

Furthermore, various parameters of the algorithm have been optimized by using static configurations and genetic algorithms. Using the best parameter setting, the algorithm achieves an F-measure of about 23%. By additionally using different settings for short and long documents, an overall F-measure of about 35% could be achieved, which is a rather high value for intrinsic plagiarism detection systems. In addition, an F-score of over 50% could be gained for short documents. Thereby the splitting number for the two document subsets has intuitively been chosen, and it should be considered to find the optimal value algorithmically in future work.

Extensive evaluations showed that the approach works very well on short documents containing less than 300 sentences, and that the more authors plagiarize, the more likely it is for the algorithm to detect the according sections. For documents consisting of less than 50 sentences, an F-measure of nearly 70% could be reached. Nevertheless, a drawback of the approach is that it predicts too much in many cases - i.e. it detects plagiarism where there is none - leading to a high number of false-positives. Future work should concentrate on reducing this number to improve the algorithm significantly.

On the other side, the number of false-negatives is low, implying that the approach is well-suited for ensuring that a document is not plagiarized. Evaluations showed that if the algorithms states that a document is plagiarism-free, it is right in over 90% of the cases.

Other future work includes the application of the algorithm to other languages like German, French or Spanish, which could produce the same or even better results, because other languages often use more complex grammar syntaxes than English. The more choice an author has to build his sentences, the more individual *style* the documents gets, and the easier it should be for the Plag-Inn algorithm to find irregularities.

The approach could also be adapted to be able to verify and/or attribute authors of single-author text documents, or to verify authorships in multi-author text documents. Moreover, as the approach relies on the style of authors, it could be used for genre detection, spam filtering or recommender systems as well.

References

[ABG10] Nikolaus Augsten, Michael Böhlen, and Johann Gamper. The pq-Gram Distance between Ordered Labeled Trees. *ACM TRANSACTIONS ON DATABASE SYSTEMS (TODS)*, 2010.

[Bal09] Enrique Vallés Balaguer. Putting Ourselves in SME's Shoes: Automatic Detection of Plagiarism by the WCopyFind tool. In *Proceedings of the SEPLN'09 Workshop on Uncovering Plagiarism, Authorship and Social Software Misuse*, pages 34–35, 2009.

[BHR00] L. Bergroth, H. Hakonen, and T. Raita. A Survey of Longest Common Subsequence Algorithms. In *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, SPIRE '00, pages 39–48, Washington, DC, USA, 2000. IEEE Computer Society.

[Bil05] Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computuer Science*, 337:217–239, June 2005.

[BSL+04] Jun-Peng Bao, Jun-Yi Shen, Xiao-Dong Liu, Hai-Yan Liu, and Xiao-Di Zhang. Semantic Sequence Kin: A Method of Document Copy Detection. In Honghua Dai, Ramakrishnan Srikant, and Chengqi Zhang, editors, *Advances in Knowledge Discovery and Data Mining*, volume 3056, pages 529–538. Springer Berlin, Heidelberg, 2004.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[Got10] Thomas Gottron. External Plagiarism Detection Based on Standard IR Technology and Fast Recognition of Common Subsequences - Lab Report for PAN at CLEF 2010. In Martin Braschler, Donna Harman, and Emanuele Pianta, editors, *CLEF (Notebook Papers/LABs/Workshops)*, 2010.

[Kar00] Jussi Karlgren. *Stylistic Experiments For Information Retrieval*. PhD thesis, Swedish Institute for Computer Science, 2000.

[KLD11] Mike Kestemont, Kim Luyckx, and Walter Daelemans. Intrinsic Plagiarism Detection Using Character Trigram Distance Scores. In V. Petras, P. Forner, and P. Clough, editors, *CLEF 2011 Labs and Workshop, Notebook Papers*, Amsterdam, The Netherlands, 2011.

[KM03] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 423–430, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[Lev93] Beth Levin. *English Verb Classes and Alternations : A Preliminary Investigation*. University Of Chicago Press, September 1993.

[MMS93] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–330, June 1993.

[OLRV10] Gabriel Oberreuter, Gaston L'Huillier, Sebastián A. Ríos, and Juan D. Velásquez. FastDocode: Finding Approximated Segments of N-Grams for Document Copy Detection. In Martin Braschler, Donna Harman, and Emanuele Pianta, editors, *Notebook Papers of CLEF 10 Labs and Workshops*, 2010.

[OLRV11] Gabriel Oberreuter, Gaston L'Huillier, Sebastián A. Ríos, and Juan D. Velásquez. Approaches for Intrinsic and External Plagiarism Detection - Notebook for PAN at CLEF 2011. In Vivien Petras, Pamela Forner, and Paul D. Clough, editors, *Notebook Papers of CLEF 11 Labs and Workshops*, 2011.

[PEBC+11] Martin Potthast, Andreas Eiselt, Alberto Barrón-Cedeño, Benno Stein, and Paolo Rosso. Overview of the 3rd International Competition on Plagiarism Detection. In Vivien Petras, Pamela Forner, and Paul Clough, editors, *Notebook Papers of CLEF 11 Labs and Workshops*, 2011.

[PSBCR10] Martin Potthast, Benno Stein, Alberto Barrón-Cedeño, and Paolo Rosso. An Evaluation Framework for Plagiarism Detection. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, Beijing, China, August 2010. Association for Computational Linguistics.

[SG00] Mark Stevenson and Robert Gaizauskas. Experiments on sentence boundary detection. In *Proceedings of the sixth conference on Applied natural language processing*, ANLC '00, pages 84–89, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[SKF00] Efstathios Stamatatos, George Kokkinakis, and Nikos Fakotakis. Automatic text categorization in terms of genre and author. *Comput. Linguist.*, 26:471–495, December 2000.

[SM09] Leanne Seaward and Stan Matwin. Intrinsic Plagiarism Detection using Complexity Analysis. In *CLEF (Notebook Papers/Labs/Workshop)*, 2009.

[Sta09] Efstathios Stamatatos. Intrinsic Plagiarism Detection Using Character n-gram Profiles. In *CLEF (Notebook Papers/Labs/Workshop)*, 2009.

[Sta12] Stanford Parser Website. A statistical parser. http://nlp.stanford.edu/software/lex-parser.shtml, visited January 2012.

[TS12] Michael Tschuggnall and Günther Specht. Plag-Inn: Intrinsic Plagiarism Detection Using Grammar Trees. In Gosse Bouma, Ashwin Ittoo, Elisabeth Métais, and Hans Wortmann, editors, *NLDB*, volume 7337 of *Lecture Notes in Computer Science*, pages 284–289. Springer, 2012.

[UKN05] Özlem Uzuner, Boris Katz, and Thade Nahnsen. Using Syntactic Information to Identify Plagiarism. In *Proc. 2nd Workshop on Building Educational Applications using NLP. Ann Arbor*, 2005.

Composition Methods for Link Discovery

Michael Hartung, Anika Groß, Erhard Rahm

Department of Computer Science, University of Leipzig
{hartung,gross,rahm}@informatik.uni-leipzig.de

Abstract: The Linked Open Data community publishes an increasing number of data sources on the so-called Data Web and interlinks them to support data integration applications. We investigate how the composition of existing links and mappings can help discovering new links and mappings between LOD sources. Often there will be many alternatives for composition so that the problem arises which paths can provide the best linking results with the least computation effort. We therefore investigate different methods to select and combine the most suitable mapping paths. We also propose an approach for selecting and composing individual links instead of entire mappings. We comparatively evaluate the methods on several real-world linking problems from the LOD cloud. The results show the high value of reusing and composing existing links as well as the high effectiveness of our methods.

1 Introduction

The Linked Open Data (LOD) community publishes an increasing number of data sources from different domains [BHBL09]. These sources are frequently linked with each other to support distributed queries and other forms of data integration. The support of open standards and uniform data and link representation in RDF simplifies the broad use of LOD sources in diverse applications. In addition to general data sources such as DBpedia [BLK+09] there are hundreds of domain-specific sources. For instance, Bio2RDF [BNT+08] provides many life science datasets and ontologies while GeoNames[1] and the New York Times[2] publish data about geographical entities.

There are already numerous RDF links between LOD sources available (≈500 million in Sep. 2012[3]). Still, there is a strong need for increasing the number of links as most sources are linked to only one or a few other sources and new sources need to be linked. The size of the sources makes a manual link discovery infeasible, hence (semi-) automatic match algorithms are needed to determine so-called *mappings* (sets of links) between sources. Many approaches have thus been proposed to directly match the objects of different sources (see Related Work). We aim at complementing these approaches by reusing and composing existing links and mappings to indirectly create new links. Such an approach is especially promising for domains with many existing mappings, e.g., in the life sciences.

[1] GeoNames: http://www.geonames.org/
[2] New York Times - Linked Open Data: http://data.nytimes.com/
[3] http://www4.wiwiss.fu-berlin.de/lodcloud/state/

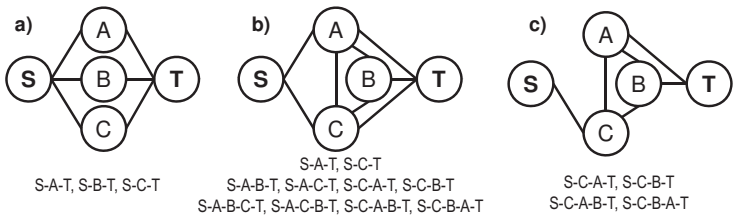


Figure 1: Example scenarios with alternative routes for mapping composition

We already investigated mapping composition for matching biomedical ontologies [GHKR11]. That work focused on scenarios as shown in Fig. 1a where we only compose two mappings (via one intermediate source) per path. By combining several such composed mappings via different intermediates we were able to achieve high quality results with little computation overhead. In [HGKR12] we also started to investigate methods to select the most promising routes for cases when we can compose across several intermediate sources. A main goal of the present paper is to investigate mapping composition for more general mapping topologies and for different domains. Furthermore, we study not only the composition and combination of entire mappings but also the composition of individual links.

As shown in Fig. 1 there are typically many alternative paths to create a mapping between two sources, S and T. For instance, in Fig. 1b the intermediates are connected with each other resulting in ten possible composition routes compared to only three in Fig. 1a (for the same sources). There can be also situations like in Fig. 1c where no route between S and T exists with only one intermediate. Thus, one must consider longer mapping chains consisting of >2 mappings. We therefore need an automatic and general approach to select the most suitable routes that likely result in the best composed mappings.

In this paper, we make the following contributions:

- We study the composition of mappings for link discovery in general, i.e., for arbitrary mapping topologies and paths of arbitrary length.
- We propose different methods to select and combine composed mappings along different paths. (Sec. 3) We further propose a link-based composition approach for selecting and composing individual links instead of entire mappings. (Sec. 4)
- We comparatively evaluate the methods for two domains, namely to interconnect anatomy ontologies and geographical data sources. The results show that we are able to select the most promising routes along sources and entities for efficient mapping composition resulting in high quality mappings. (Sec. 5)

In Sec. 2 we introduce our source and mapping model, discuss the concept of mapping composition and outline the problem that we investigate. We discuss related work in Sec. 6 and summarize in Sec. 7. The Appendix provides the pseudo-code for the algorithms proposed in the paper.

2 Preliminaries

We first describe our source and mapping model. We then discuss mapping composition for two and multiple mappings. Finally, we outline the problem that we address.

2.1 Data Sources, Links and Mappings

A linked data source DS consists of a set of entities. Each entity has an unique URI that is used to reference the object. For instance, the city Leipzig in DBpedia is unambiguously referenced by `http://dbpedia.org/resource/Leipzig`. Entities and their relationships are described by RDF triples of the form (*subject*, *predicate*, *object*) where the third component is either a literal or a reference to an entity of the same or a different source. For example we can use the following triple with a literal to specify the population of Leipzig: (`http://dbpedia.org/resource/Leipzig`,`populationTotal`, 528049). On the other hand, we use an object reference to specify that Leipzig is the largest city of Saxony: (`http://dbpedia.org/resource/Leipzig`,`largestCityOf`, `http://dbpedia.org/resource/Saxony`).

For linking different sources, we mainly use links of type `owl:sameAs` denoting that the linked objects are equal, i.e., represent the same real-world entity. For example, the triple (`http://dbpedia.org/resource/Leipzig`, `owl:sameAs`, `http://data.nytimes.com/N86446625683764674801`) specifies that Leipzig in DBpedia matches to an entity in the New York Times data source. Note that there can be other link types but in this work we will focus on determining sameAs-links since they make up the majority of links between different data sources in the LOD.

A *mapping* between two data sources S and T, $M_{S,T} = \{(o_1, o_2, sim)|o_1 \in S, o_2 \in T, sim \in [0, 1]\}$, consists of a set of sameAs-links between these sources, e.g., as determined by some link discovery (match) method. Each link (correspondence) interconnects two related objects o_1 and o_2. Their relatedness is represented by a similarity value sim between 0 and 1 determined by the used match approach. The greater the sim value the more similar are the corresponding objects. We assume a similarity of 1 for manually curated links.

2.2 Mapping Composition

2.2.1 Binary Mapping Composition

In general mapping composition is applied to derive new mappings between two data sources by reusing already existing mappings. Thus, new mappings are generated indirectly via one or more intermediate sources instead of a direct match between the two input sources. The basic situation is the following. We have two data sources (S,T) and two mappings ($M_{S,IS}$,$M_{IS,T}$) w.r.t. an intermediate source IS. Using `domain` and

range of the mappings we can find out which entities of S, T or IS are covered by the given mappings, e.g., the entities covered by $M_{IS,T}$ in T are in the range of the mapping: range($M_{IS,T}$). Mapping composition is then applied in the following way. A compose operator takes as input two mappings (from S and T to IS) and produces new links between objects of S and T if links share the same object in IS:

$$M_{S,T} = \texttt{compose}(M_{S,IS}, M_{IS,T}) = M_{S,IS} \circ M_{IS,T} = \{(o_1, o_2, aggSim(sim_1, sim_2)) | o_1 \in S, o_2 \in T, b \in IS : \exists (o_1, b, sim_1) \in M_{S,IS} \wedge \exists (b, o_2, sim_2) \in M_{IS,T}\}$$

The similarity values of input links are aggregated (aggSim) into new similarity values, e.g., by computing their maximum, average or by multiplication.

2.2.2 n-ary Mapping Composition

To define the composition of more than two mappings, we first introduce the notion of mapping paths. In particular, a mapping path $P = (M_{S_1,T_1}, M_{S_2,T_2}, \ldots, M_{S_n,T_n})$ of size n w.r.t. a given set of mappings $\mathcal{M}$ is an ordered chain of mappings with the following properties:

1. *Composability*: $\forall M_{S_i,T_i} \in P : M_{S_i,T_i} \in \mathcal{M} \wedge T_i = S_{i+1}$
2. *Start/End Sources*: the input sources S and T form the start and end of the path, i.e., $S = S_1$ and $T = T_n$
3. *Max. Occurrence*: A mapping $M_{S_i,T_i} \in \mathcal{M}$ occurs at most one time in a path P
4. *Acyclicity*: P has no circles, i.e., there is no sub path $(M_{S_j,T_j}, \ldots, M_{S_k,T_k})$ in P such that $S_j = T_k$

Property 1 ensures that we can traverse (compose) along the path, i.e., the range of a mapping must equal the domain of the succeeding mapping. Furthermore, we can only use mappings available in $\mathcal{M}$. Property 2 guarantees that the start (end) of the path are our sources to be matched, i.e., S or T, respectively. According to property 3 we only allow one occurrence of a mapping within a path. Finally, property 4 restricts the number of possible paths to those with no circles. Together with property 3 we thus exclude paths of infinite length as well as paths visiting intermediate sources multiple times.

To generate a mapping $M_{S,T}$ using a mapping path $P = (M_{S_1,T_1}, M_{S_2,T_2} \ldots, M_{S_n,T_n})$ with $S_1 = S$ and $T_n = T$ we can n-1 times apply the binary compose operator ($\circ$) in the following way:

$$M_{S,T} = \texttt{compose}(P) = (\ldots (M_{S_1,T_1} \circ M_{S_2,T_2}) \circ \ldots) \circ M_{S_n,T_n}$$

Starting with the first mapping M_{S_1,T_1} we compose succeeding mappings along the mapping path with the binary operator. The result of one binary compose step is used as input for the next step until we processed the last mapping M_{S_n,T_n} of the path.

2.3 Problem Statement

For two data sources S and T and a given set of mappings $\mathcal{M}$, the problem we investigate is to use composition-based methods to determine a new mapping $M_{S,T}$ consisting of links between entities of S and T. The mappings in $\mathcal{M}$ should contain at least one mapping path between S and T but otherwise there are no restrictions about the number of mappings or the degree of connectedness. The resulting mapping should be of good quality, i.e., all discovered links should be correct (precision) and the number of discovered links should be as high as possible (recall). A composition method should be efficient and scalable to large sources and a large number of mappings.

3 Mapping-based Composition

In the following we propose different methods based on mapping composition to solve the problem we address. We first present an *All* strategy that composes and combines all mapping paths for a given set of mappings $\mathcal{M}$. We then present *Selection* methods that select the most promising mapping paths by considering their effectiveness or complement.

To exemplarily show how the proposed methods and algorithms work, we will use the simple yet comprehensive running example shown in Fig. 2. The sources and mappings are shown on the left side, while a more detailed view on the entities and links is provided on the right side. For simplicity, we assume that all links have an unique similarity of 1.0.

3.1 All Strategy

The idea behind the *All Strategy* is to evaluate all possible mapping paths between the two input sources S and T. For this purpose, we first need to find all possible paths. We can then compose the mappings per path and combine the composed mappings. The first part is related to computing the transitive closure of $\mathcal{M}$. However, we are only interested in all S-T paths and do not consider paths between all available sources.

The determination of all mapping paths $\mathcal{MP}$ between two sources S and T for a given set of mappings $\mathcal{M}$ requires a traversal of mappings starting in S (see Algorithm 1 in the Appendix). We assume that we can traverse a mapping in both directions, e.g., in our example we can traverse from A to B as well as from B to A using $M_{A,B}$. In our running example of Fig. 2(left), we would first select $M_{S,A}$ and $M_{S,B}$ as possible starting paths. In the first round, we consider $(M_{S,A}, M_{A,T})$ as a final path. Furthermore, temporary paths $(M_{S,A}, M_{A,B})$, $(M_{S,B}, M_{B,A})$ as well as $(M_{S,B}, M_{B,C})$ are created. The second round would produce $(M_{S,B}, M_{B,A}, M_{A,T})$ and $(M_{S,B}, M_{B,C}, M_{C,T})$ as final paths, one temporary path namely $(M_{S,A}, M_{A,B}, M_{B,C})$ remains. In the last round, we can use $M_{C,T}$ to build $(M_{S,A}, M_{A,B}, M_{B,C}, M_{C,T})$. Thus, we find four mapping paths between S and T.

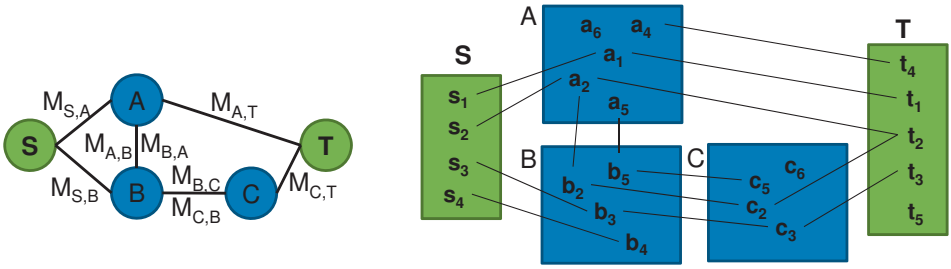


Figure 2: Composition scenario: sources and mappings (left), entity links (right)

Having found all possible paths between the input sources, we can now perform composition as described in Sec. 2.2.2. In particular, we generate $|\mathcal{MP}|$ composed (partial) mappings which we need to merge (unify) to create a final mapping between S and T (see Algorithm 2 in the Appendix). In this paper, we apply a union operator, i.e., the links from all partial mappings are unified. For our example, composing along $(M_{S,A}, M_{A,T})$ results in a mapping consisting of two links: (s_1,t_1) and (s_2,t_2). The mapping path along B and C produces one link: (s_3,t_3). No link is created when considering the $(M_{S,B}, M_{B,A}, M_{A,T})$ path. The longest path via A, B and C creates a link between s_2 and t_2: (s_2,t_2). We now merge all determined links to get the final mapping: $M_{S,T} = \{(s_1, t_1), (s_2, t_2), (s_3, t_3)\}$.

3.2 Selection Strategies

The introduced All Strategy evaluates all possible mapping paths. However, the individual mapping paths are often redundant by leading to the same links. The *Selection Strategy* tries to avoid the repeated calculation of the same links by selecting the most valuable mapping paths and only considers these paths for composition and combination. In the following we first introduce the notion of effectiveness for a mapping path. We will then use this measure as well as others to rate mapping paths w.r.t. their usefulness for composition.

The basic situation for composing two mappings via one intermediate is illustrated in Fig. 3 [HGKR12]. We observe that the compose can at best create new links between entities of S/T that are mapped to the intermediate source IS. The more entities are covered by a mapping to IS the more likely it is that they can be interlinked to entities in the other data source. Thus, intermediates where mappings only cover a small portion of S/T are less effective compared to those covering larger portions. Furthermore, there should be a high overlap of mapped objects in IS, i.e., many IS objects should be in both $\texttt{range}(M_{S,IS})$ and $\texttt{domain}(M_{IS,T})$. This is because new links can only be created if there are intermediate objects for the composition. By contrast, a small overlap will only result in a few correspondences, i.e., small and likely incomplete mappings.

Summarizing these observations, we can estimate the effectiveness of two mappings $M_{S,IS}$

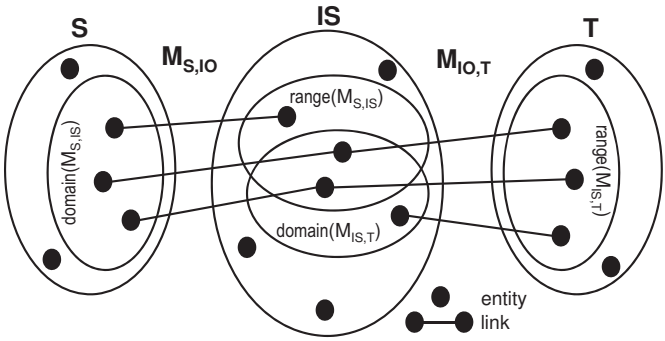


Figure 3: General situation for mapping composition with two mappings and one intermediate

/ $M_{IS,T}$ to be composed as follows:

$$eff(M_{S,IS}, M_{IS,T}) = \frac{2 \cdot |range(M_{S,IS}) \cap domain(M_{IS,T})|}{|S| + |T|}$$

The measure is mainly based on the size of overlapping objects in the intermediate, i.e., the larger the overlap the better the effectiveness. Second, we relate this overlap to the sizes of the sources to be matched S and T. Thus, only mappings with many links can produce a high overlap and a good coverage of objects in S and T. For instance, applying the measure to the scenario displayed in Fig. 3, we would get an effectiveness of $\frac{2 \cdot 2}{5+5}$=0.4.

We can generalize the effectiveness measure for mapping paths of arbitrary length. When performing composition along multiple mappings of a path, it is intuitive that the effectiveness of the path decreases with more mappings. Since each single compose step (see Sec. 2.2.2) has its own effectiveness, the overall effectiveness of a path $P = (M_{S_1,T_1}, \ldots, M_{S_n,T_n})$ can be estimated by multiplying the single effectiveness values for all mapping pairs along the path:

$$eff((M_{S_1,T_1}, \ldots, M_{S_n,T_n})) = \prod_{i=1}^{n-1} eff(M_{S_i,T_i}, M_{S_{i+1},T_{i+1}})$$

Considering our running example from Fig. 2 we would derive the following effectiveness values for our paths. For $(M_{S,A}, M_{A,T})$ the effectiveness is $\frac{2 \cdot 2}{4+5} \approx 0.44$. The two paths of length three result in an effectiveness of $\frac{2 \cdot 0}{4+5} \cdot \frac{2 \cdot 1}{4+5} = 0$ for $(M_{S,B}, M_{B,A}, M_{A,T})$ and $\frac{2 \cdot 1}{4+4} \cdot \frac{2 \cdot 2}{4+5} \approx 0.11$ for $(M_{S,B}, M_{B,C}, M_{C,T})$, respectively. The longest path $(M_{S,A}, M_{A,B}, M_{B,C}, M_{C,T})$ has an effectiveness of $\frac{2 \cdot 1}{4+4} \cdot \frac{2 \cdot 2}{5+4} \cdot \frac{2 \cdot 2}{4+5} \approx 0.05$.

SelectByEffectiveness We can now use the effectiveness measure to select the most valuable (e.g., the best k) paths and compose and combine only these selected paths (*select-ByEffectiveness*). For instance, in our example we could only use the two best paths ($(M_{S,A}, M_{A,T})$ and $(M_{S,B}, M_{B,C}, M_{C,T})$) for composition. This would lead to exactly the same mapping $M_{S,T}$ as performing the All Strategy described in Sec. 3.1.

SelectByComplement A second option for path selection is to consider complementing paths. The strategy would first select the most effective mapping path according to our effectiveness measure. After that, we iteratively select those paths with the largest complement compared to the already covered entities in S/T by the previous selected mapping paths. The intuition behind this procedure is to increase the number of covered entities in S/T in the mapping (and thus the recall). For instance, when linking two general data sources about geography, one might consider paths which include complementing knowledge about airports, countries, waters, cities etc.. For our running example and k=2 we would select $(M_{S,A}, M_{A,T})$ (most effective path) and the $(M_{S,B}, M_{B,C}, M_{C,T})$ path, since it offers the best complement (s_3 and s_4 in S, and t_3 in T).

The overall procedure of the *Selection Strategy* (see Algorithm 3 for details) first determines all possible mapping paths. Afterwards we apply the effectiveness measure on each of the possible paths to compute a ranked list of mapping paths. We then can select and compose the most promising (top k) paths either by their effectiveness or complement.

4 Link-based Composition

The introduced strategies so far composed and combined entire mappings. The *Link-based Strategy* aims at a more fine-grained approach by selecting and composing individual links to generate composed links between the two sources to interconnect. For this purpose, we model link discovery as a graph problem and reuse known graph algorithms such as Shortest Path to identify the most promising link paths for composition. In the following we first describe how we create the graph representation from the given sources and mappings. We then show how we select and compose the links to determine the mapping $M_{S,T}$.

We assume a directed, weighted graph $G = (V, E)$ consisting of vertexes V and edges E. Each directed edge $e = (v_1, v_2, weight) \in E$ interlinks two vertexes of V (from v_1 to v_2) including a similarity-based $weight$ which we will later use for path selection within the graph. The transformation from the given mappings in $\mathcal{M}$, the data sources S/T to be linked into such a graph $G = (V, E)$ can be described by some basic rules:

1. Each entity referenced by a link in a mapping of $\mathcal{M}$ becomes a vertex $v \in V$.
2. For each link (o_1, o_2, sim) in a mapping $M_{S',T'} \in \mathcal{M}$ we create edges as follows:
   (a) $if(S' = S)$: $(o_1, o_2, 1 - sim + \epsilon) \in E$
   (b) $if(T' = T)$: $(o_1, o_2, 1 - sim + \epsilon) \in E$
   (c) $otherwise$: $(o_1, o_2, 1 - sim + \epsilon) \in E$ and $(o_2, o_1, 1 - sim + \epsilon) \in E$
3. There exists an unambiguous target vertex $target \in V$.
4. All vertexes v representing entities of T are connected with $target$, i.e., we create edges $(v, target, \epsilon)$ for all $v \in T$.

The idea of the transformation is to model a routing problem, i.e., we like to find the shortest paths from each source vertex (representing an entity in S) to the unambiguous

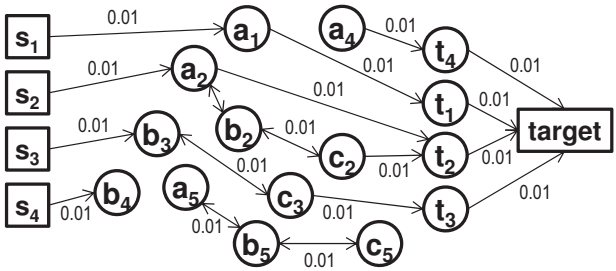


Figure 4: Resulting graph for running example displayed in Fig. 2(right)

$target$ vertex. We thus consider each entity as a vertex and transform links into directed, weighted edges. Vertexes representing source entities have no incoming edges, whereas vertexes of target entities have no outgoing edges (except the ones to the unambiguous $target$ vertex). Edges between entities of intermediate sources can be traversed in both directions (two edges for one link). The greater the similarity of a link, the smaller the weight of an edge, i.e., routing algorithms will likely traverse along edges with small weights. We consider basic costs ϵ for each edge to prefer short paths over longer ones. For our running example of Fig. 2(right) we would create the graph shown in Fig. 4 when using an ϵ of 0.01. We consider $s_1, \ldots, s_4$ as starting vertexes with only outgoing edges. The unambiguous $target$ vertex is displayed on the right hand side. All other vertexes involved in at least one link are shown in circles. The vertexes $t_1, \ldots, t_4$ of T have only outgoing edges to the unambiguous $target$ vertex. Links between entities of A, B, or C are binary, e.g., one can traverse from $b_2 \in B$ to $c_2 \in C$ and vice versa. Since we assume an unique similarity value of 1.0 for each edge, we have weights of 0.01 for each link, e.g., the link $(a_1, t_1, 1.0)$ is transformed into an edge $(a_1, t_1, 1.0 - 1.0 + 0.01) = (a_1, t_1, 0.01)$.

Using the generated graph we can now exploit the structure to find the most cost-effective routes between entities of S and T. In particular, we will make use of the Shortest-Path (Dijkstra) algorithm [Dij59] to solve the problem (see Algorithm 4 in the Appendix). We iterate over all entities of source S and try to find the shortest path to the $target$ vertex according to the given graph G. For paths found, we create a new link between the current source entity and the last entity before $target$ in the path belonging to T. The similarity is computed according to the formula described in Sec. 2.2.1. The newly created link is added to the mapping $M_{S,T}$. Considering the graph of our running example, we would detect the following paths and thus links between S and T. For entity s_1 there is only one route via a_1 and t_1 to reach $target$. Thus, we would create a link (s_1, t_1) for the mapping. For entity s_2 the shortest path is using the (s_2,a_2,t_2,$target$) route with costs of 0.03. The route via b_2 and c_2 has more costs (0.05) and is not considered. From the third entity s_3 one can traverse along b_3, c_3 and t_3 with minimum costs of 0.03 to the $target$ vertex. For s_4 no path to the $target$ exists (route stops in b_4). Hence, no link for s_4 can be created. In summary, we determine three links, namely (s_1, t_1), (s_2,t_2) and (s_3,t_3) for $M_{S,T}$.

The previously explained procedure returns only the shortest paths in one direction, namely from S to T. This could result in incomplete mappings, e.g., when one entity in a data

source links to multiple entities in the opposite source. We therefore evaluate both directions from S to T, and from T to S to find all links between both sources. Traversing in the opposite direction (from T to S) is analogously implemented than the forward traversal already described. When constructing the graph we now insert an unambiguous $source$ vertex where all entities of S are connected with. Furthermore, vertexes representing entities of T have only outgoing edges and we search for the shortest paths from those vertexes to the unambiguous $source$ vertex. The overall procedure of link-based composition is shown in Algorithm 5 in the Appendix.

5 Evaluation

We evaluate our composition methods by analyzing four real-world link discovery problems from two domains. In particular, we produce mappings for the *Geography* instance matching tasks[4] and the *Anatomy*[5] match task of the Ontology Alignment Evaluation Initiative (OAEI). By doing so, we can evaluate the quality of our computed mappings w.r.t. the publicly available OAEI gold standard mappings using precision, recall and F-measure. We first introduce the experimental setup, the used data sources and mappings. We then compare the effectiveness and efficiency of our composition strategies and analyze the impact of the number k of selected mappings and the number of intermediate sources.

5.1 Setup and Overview

For *Geography*, we focus on interlinking NYTimes Data (NYT) with the three LOD sources DBpedia (DBp), FreeBase (FB) and GeoNames (GeoN), i.e., we compare NYT-DBp, NYT-FB and NYT-GeoN. In each case, two of the sources are not matched and can thus be used for composition. We further use mappings to three other intermediate sources from the LOD cloud, namely WorldFactBook (WFB), LinkedGeoData (LGeo) and YAGO. For *Anatomy*, we generate mappings between Adult Mouse Anatomy (MA) and the anatomy part of NCI Thesaurus (NCIT) by composing mappings to four further intermediate sources, namely RadLex, Foundational Model of Anatomy (FMA), Unified Medical Language System (UMLS) and Uberon.

While our composition methods should reuse existing high quality mappings, we did not have them for the considered scenarios. We thus precomputed approximate mappings between any two sources of a domain. These input mappings are generated by a standard metadata-based match technique using our prototype GOMMA [KGHR11]. We compute links between entities based on the similarity of their names and synonyms, i.e., we use links with similarity values between 0 and 1 in the evaluation. We include links of high TriGram similarity and select only the best correspondence(s) per entity. All experiments were performed on an Intel(R) Core (TM) i5-2500 CPU, 4x3.30GHz, 8GB memory ma-

[4] http://www.instancematching.org/oaei/
[5] http://oaei.ontologymatching.org/2012/anatomy/

(a)

Source Size	MA 2738	NCIT 3298	UMLS 87913	FMA 81059	RadLex 30773	Uberon 4958	
Mapping Size		*1270*	2975	1601	1082	2300	**MA**
			4214	2337	1347	1703	**NCIT**
				63051	17266	5497	**UMLS**
					21781	3504	**FMA**
						2374	**RadLex**
							UberOn

(b)

Source Size	NYT 1920	DBp 1920	GeoN 1780	FB 1920	YAGO 1086	LGeo 436	WFB 254	
Mapping Size		*1406*	*1781*	*1971*	1130	459	221	**NYT**
			1230	1997	1154	243	232	**DBp**
				1866	1088	472	234	**GeoN**
					1222	480	236	**FB**
						216	202	**YAGO**
							25	**LGeo**
								WFB

Figure 5: Source and mapping sizes for *Anatomy* (a) and *Geography* scenario (b).

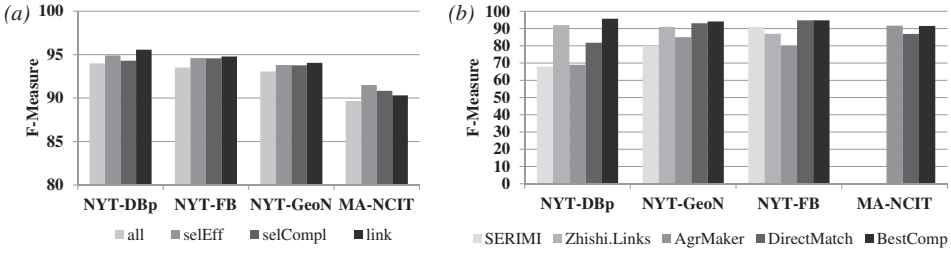
chine with 64-bit Windows 7 Professional OS and a 64-bit JVM.

Fig. 5 gives an overview about the size of the used data sources and mappings between them. For *Anatomy* (Fig. 5a), there are two very large intermediate ontologies (UMLS, FMA) with more than 80,000 entities and a mapping between them with more than 63,000 links. Uberon is the smallest of the used intermediate sources. However, it provides links to 2,300 MA entities while the large FMA covers only ≈1,600 links to MA. UMLS provides most links to MA (≈3,000) and NCIT (≈4,200). Note, that we do not use the mapping between MA and NCIT (size printed in italic numbers) for composition.

The sources and mappings for the *Geography* domain are comparatively small (Fig. 5b). NYT, DBp and FB cover more than 1,900 geographical entities while LGeo and WFB comprise less then 500. While DBpedia and some of the other sources contain many more entities the goal of the OAEI Instance Matching task is to find links w.r.t. NYT in the geography area so that the sources were restricted to the relevant subsets. This also leads to small mapping sizes of <500 links between LGeo/WFB and the other sources. For each of the considered geographical data sources most links point to FB (Freebase). Again, we do not use the shown direct mappings for composition in case this is the mapping to be evaluated (italic numbers). For instance, when computing the NYT-DBp mapping we include direct mappings between all sources except the one between NYT and DBp.

5.2 Comparison of Composition Methods

We consider the All Strategy (*all*), the two selection strategies SelectByEffectiveness (*selEff*) and SelectByComplement (*selCompl*) as well as the Link-based Strategy (*link*) for evaluation. The results achieved for each method and match task are displayed in Fig. 6a. For each match task we used the maximum number of available mappings. This results in 325 possible paths for each of the three Geography tasks and 64 paths for Anatomy. For all tasks we are able to achieve F-measure values over 90%. However, there are some slight differences between the methods. The *all* strategy performs worst for all tasks, apparently because the large number of mapping paths lead to a relatively low precision (incorrect links). By contrast, *link* achieves the best quality in all geography tasks. The two selection methods and especially *selEff* also perform well. *selCompl* is slightly less



(c)

	NYT-DBp				NYT-FB				NYT-GeoN				MA-NCIT			
Strategy	all	selEff	selCompl	link	all	selEff	selCompl	link	all	selEff	selCompl	link	all	selEff	selCompl	link
Runtime in s	81	<1	<1	6	73	<1	<1	5	78	<1	<1	5	296	6	5	20

Figure 6: Comparison of composition strategies – *(a)* Composition results for all four tasks *(b)*, Comparison with direct match approaches, *(c)* Execution times (in s)

effective since it may select paths with a good complement but lower effectiveness when the complementing entities cannot be linked.

Regarding runtime efficiency the differences between the methods are even greater (see Fig. 6c). As expected, *all* requires the most time (with up to 5 minutes for Anatomy) since it composes all possible mapping paths. The selection methods are the fastest with $<$1s for each Geography task and about 5s for Anatomy. *link* requires some more time than the selection strategies due to the time needed for constructing the graphs and running the Shortest Path algorithm. In summary, the results show that using the selection or link strategy one can achieve high quality results with very short execution times.

We further compare the effectiveness of our composition approaches (BestComp) with those of the systems that participated in the OAEI 2011 campaign (SERMI [AHSdV11], Zhishi.Links [NRZW11], AgreementMaker [CSC$^+$11]) and with our own match strategy (DirectMatch) described in the setup. The results in Fig. 6b show that composition of existing mappings can improve the match quality compared to traditional match approaches. In particular, for all Geography tasks we achieve the best quality in terms of F-measure. Interestingly, the results of DirectMatch are topped by our composition methods which use mappings produced with DirectMatch. This shows that mapping composition can harvest additional knowledge in intermediate sources to discover more and better links. For Anatomy, AgreementMaker achieves the best quality (SERMI, Zhishi.Links did not participate in this track). They also exploit background knowledge from other anatomy ontologies and combine the results with those from a direct match of the sources.

5.3 Sensitivity Analysis

When applying the selection strategies, one needs to set the value of k to specify how many of the possibly numerous paths should be considered. The diagram presented in Fig. 7a

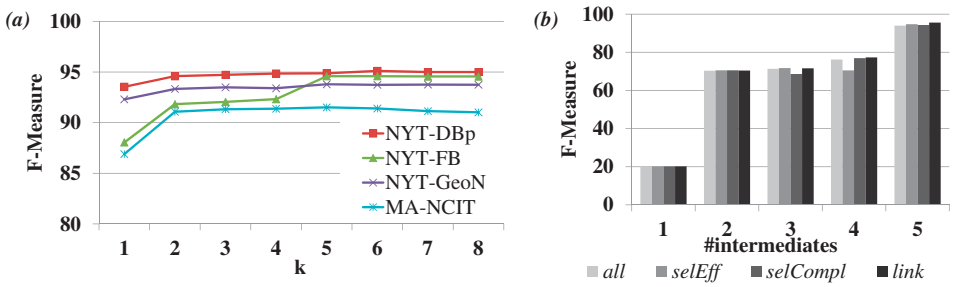


Figure 7: Sensitivity Analysis – *(a)* Influence of k for *selEff* strategy, *(b)* Results for increasing number of intermediates for NYT-DBp match task

shows how the number of selected paths for *selEff* influences the final match quality. We increased the number from 1 to 8 and noticed a similar behavior in all scenarios. A single mapping path generally leads to insufficient match quality but the combination of two or three paths achieves already high F-Measure values. The match quality can not be improved further for more than 6 mapping paths or may even decrease at some point (e.g., for Anatomy). These results show, that one can already achieve a good match quality when selecting only a few but effective paths.

In a further sensitivity experiment we test how the methods perform for a varying number of intermediates. In particular, we increase the number of possible intermediates (and thus mappings) and measured the quality. In each step we considered all available mappings among the used intermediates. The results for the NYT-DBp task are shown in Fig. 7b. We observe that an increasing number of intermediates leads to a better match quality, since more mapping paths can be exploited. When using one or two intermediates the methods do not differ due to the small number of possible paths, namely only 1 (4) paths for one (two) intermediates. *link* achieves the best quality for 5 intermediates with 325 paths. This shows that the link-based strategy is especially valuable for composition scenarios where a large number of possible paths need to be explored. When only a few paths exist, one can apply *all* or a selection strategy instead.

6 Related Work

Many approaches have already been published for link discovery and the related problems of entity resolution and ontology matching. General frameworks for link discovery include SILK [VBGK09], LIMES [NNA11] and Zhishi.links [NRZW11]. These approaches support different similarity measures to directly compute links between LOD data sources. Some of them incorporate methods to scale with large data sources, e.g., LIMES exploits the mathematical characteristics of metric spaces to speed up the match process, or SILK performs a blocking step to reduce the number of comparisons. Many more approaches have been proposed for entity resolution (see [KR10, EIV07] for surveys) as well as ontol-

ogy matching (see [ES07, RB01] for surveys). Usually the approaches directly compare the input sources by employing different lexical or structural methods in workflows.

The principle of composition has mainly been studied for schemas [DL04, Rah11] and in model management [FKPT05, BM07]. Only a few approaches consider this technique for ontology matching or link discovery. For instance, [ZB05] utilizes the FMA ontology to derive mappings between MA and NCIT. Furthermore, the SAMBO system [LT06] or AgreementMaker [CSC+11] utilize background knowledge like the UMLS or Uberon to find additional links in their match process. [TGO+10] presents an empirical study of mapping composition with mappings from BioPortal. In own previous works, we investigated one-hop mapping composition for ontologies in the life sciences [GHKR11, HGKR12] and found out that the usage of multiple intermediates can help to increase the overall match quality.

In contrast to previous works, we study mapping composition for link discovery in general and differ in the following points. First, we match indirectly by reusing existing mappings and by applying composition along different mapping paths of different length. Second, the proposed methods can cope with various mapping composition scenarios, i.e., we can perform composition for a fully connected network of sources as well as for sparsely interconnected sources. Third, we evaluate the effectiveness and usefulness of paths to select and process only the most promising one for a fast and effective link discovery.

7 Summary and Future Work

We proposed general composition methods to solve the link discovery problem of the Data Web. The introduced mapping- and link-based methods can be applied in different link discovery scenarios with sparsely or heavily interconnected data sources. The evaluation on real-world link discovery problems showed that focusing on the most effective mapping paths / links is a good strategy to produce mappings of high quality in very short execution times. For scenarios with only few mapping paths one can apply a selection strategy or the all strategy to create new mappings. For more complex networks with a large number of possible paths the link-based strategy is most promising.

In future work we aim at investigating more complex kinds of mapping composition by also taking into account relationships within intermediate sources. We further plan to study other graph algorithms such as Ford & Fulkerson for selecting links for composition.

References

[AHSdV11] S. Araújo, J. Hidders, D. Schwabe, and A.P. de Vries. SERIMI - resource description similarity, RDF instance matching and interlinking. In *OM*, 2011.

[BHBL09] C. Bizer, T. Heath, and T. Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems*, 5(3), 2009.

[BLK+09] C. Bizer, J. Lehmann, G. Kobilarov, et al. DBpedia - A Crystallization Point for the Web of Data. *Journal of Web Semantics*, 7(3), 2009.

[BM07] P.A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *Proc. of SIGMOD*, 2007.

[BNT+08] F. Belleau, M.A. Nolin, N. Tourigny, et al. Bio2RDF: Towards a mashup to build bioinformatics knowledge systems. *Journal of Biomedical Informatics*, 41(5), 2008.

[CSC+11] I.F. Cruz, C. Stroe, F. Caimi, et al. Using AgreementMaker to Align Ontologies for OAEI 2011. In *OM*, 2011.

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), 1959.

[DL04] E.C. Dragut and R. Lawrence. Composing Mappings Between Schemas Using a Reference Ontology. In *Proc. of CoopIS/DOA/ODBASE*, 2004.

[EIV07] A.K. Elmagarmid, P.G. Ipeirotis, and V.S. Verykios. Duplicate record detection: A survey. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1), 2007.

[ES07] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag New York, 2007.

[FKPT05] R. Fagin, P.G. Kolaitis, L. Popa, and W.C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *Transactions on Database Systems*, 30(4), 2005.

[GHKR11] A. Gross, M. Hartung, T. Kirsten, and E. Rahm. Mapping Composition for Matching Large Life Science Ontologies. In *Intl. Conf. on Biomedical Ontology (ICBO)*, 2011.

[HGKR12] M. Hartung, A. Gross, T. Kirsten, and E. Rahm. Effective Mapping Composition for Biomedical Ontologies. In *Proc. of SIMI Workshop @ ESWC*, 2012.

[KGHR11] T. Kirsten, A. Gross, M. Hartung, and E. Rahm. GOMMA: A Component-based Infrastructure for managing and analyzing Life Science Ontologies and their Evolution. *Journal of Biomedical Semantics*, 2:6, 2011.

[KR10] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2), 2010.

[LT06] P. Lambrix and H. Tan. Sambo–A system for aligning and merging biomedical ontologies. *Web Semantics: Science, Services and Agents on the Web*, 4(3), 2006.

[NNA11] A.C. Ngonga Ngomo and S. Auer. LIMES: a time-efficient approach for large-scale link discovery on the web of data. In *Proc. Intl. Conf. on Artificial Intelligence*, 2011.

[NRZW11] X. Niu, S. Rong, Y. Zhang, and H. Wang. Zhishi.links results for OAEI 2011. In *OM*, 2011.

[Rah11] E. Rahm. Towards large-scale schema and ontology matching. *Schema matching and mapping*, 2011.

[RB01] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4), 2001.

[TGO+10] A. Tordai, A. Ghazvinian, J. Ossenbruggen, et al. Lost in translation? Empirical analysis of mapping compositions for large ontologies. In *OM*, 2010.

[VBGK09] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk–a link discovery framework for the web of data. In *Proc. of the 2nd Linked Data on the Web Workshop*, 2009.

[ZB05] S. Zhang and O. Bodenreider. Alignment of multiple ontologies of anatomy: Deriving indirect mappings from direct mappings to a reference. In *AMIA*, 2005.

A Algorithms

In the following, we show the pseudo-code of the algorithms used by the different strategies proposed in the paper.

Algorithm 1 (`findAllMappingPaths`) is used to determine all possible mapping paths between two sources S and T based on given mappings in a mapping set $\mathcal{M}$.

Algorithm 1: findAllMappingPaths

Input: source S, target T, set of all mappings $\mathcal{M}$
Output: all mapping paths $\mathcal{MP}$ between S and T

$\mathcal{MP} \leftarrow \emptyset$;
$\mathcal{P} \leftarrow$ `getAllMappingsWithDomain(`$\mathcal{M}, S$`)`;
while $\mathcal{P} \neq \emptyset$ **do**
  $\mathcal{P}' \leftarrow \emptyset$;
  foreach $P \in \mathcal{P}$ **do**
    $lastDataSource \leftarrow$ `getLastDataSource(`P`)`;
    $\mathcal{CM} \leftarrow$ `getAllMappingsWithDomain(`$\mathcal{M}, lastDataSource$`)`;
    foreach $M_{S',T'} \in \mathcal{CM}$ **do**
      if $\neg$`contains(`P, T'`)` **then**
        P.`append(`$M_{S',T'}$`)`;
        if $T = T'$ **then**
          $\mathcal{MP} \leftarrow \mathcal{MP} \cup \{P\}$;
        else
          $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{P\}$;
  $\mathcal{P} \leftarrow \mathcal{P}'$
return $\mathcal{MP}$;

With the help of Algorithm 2 (`composeAndMergeMappingPaths`) we perform composition along the paths in $\mathcal{MP}$ to create the mapping $M_{S,T}$ between S and T.

Algorithm 2: composeAndMergeMappingPaths

Input: source S, target T, mapping paths $\mathcal{MP}$
Output: mapping $M_{S,T}$ between S and T

$allMappings \leftarrow \emptyset$;
foreach $P \in \mathcal{MP}$ **do**
  $M_{S,Tmp} \leftarrow P$.`getNextMapping()`;
  while P.`hasNextMapping()` **do**
    $M_{S',T'} \leftarrow P$.`getNextMapping()`;
    $M_{S,Tmp} \leftarrow$ `compose(`$M_{S,Tmp}, M_{S',T'}$`)`;
  $allMappings \leftarrow allMappings \cup \{M_{S,Tmp}\}$;
$M_{S,T} \leftarrow$ `union(`$allMappings$`)`;
return $M_{S,T}$;

Algorithm 3 (`composeSelectionStrategy`) shows the procedure for selection-based mapping composition either by considering path effectiveness or complement.

Algorithm 3: composeSelectionStrategy

Input: source S, target T, mappings $\mathcal{M}$
Output: mapping $M_{S,T}$ between S and T

1 $\mathcal{MP}_{all} \leftarrow$ `findAllMappingPaths(`S,T,$\mathcal{M}$`)`;
2 $\mathcal{MP}_{ranked} \leftarrow$ `computeEffectiveness(`$\mathcal{MP}_{all}$`)`;
3 $\mathcal{MP}_{topK} \leftarrow$ `selectByEffectiveness(`$\mathcal{MP}_{ranked}$`)` or `SelectByComplement(`$\mathcal{MP}_{ranked}$`)`;
4 **return** `composeAndMergeMappingPaths(`S,T,$\mathcal{MP}_{topK}$`)`;

Using `shortestPathCompose` (Algorithm 4) we create a mapping $M_{S,T}$ by determining the shortest paths between entities of S and T in graph G.

Algorithm 4: shortestPathCompose

Input: source S, target T, graph $G = (V, E)$
Output: mapping $M_{S,T}$ between S and T

1 $M_{S,T} \leftarrow \emptyset$;
2 **foreach** $s \in S$ **do**
3 $\quad shortestPath \leftarrow$ `getShortestPath(`$s, target, G$`)`;
4 $\quad$ **if** $\neg shortestPath.$`isEmpty()` **then**
5 $\quad\quad link \leftarrow$ `compose(`$shortestPath$`)`;
6 $\quad\quad M_{S,T} \leftarrow M_{S,T} \cup \{link\}$;
7 **return** $M_{S,T}$;

Algorithm 5 (`linkBasedCompose`) shows the overall procedure for the link-based composition approach. In particular, we create a forward and a backward graph on which we perform the shortest path algorithm (`shortestPathCompose`). We finally unify the results to create the mapping $M_{S,T}$ between in the input sources.

Algorithm 5: linkBasedCompose

Input: source S, target T, mappings $\mathcal{M}$
Output: mapping $M_{S,T}$ between S and T

1 $G_{forward} \leftarrow$ `buildComposeGraph(`$S, T, \mathcal{M}$`)`;
2 $M_{S,T} \leftarrow$ `shortestPathCompose(`$S, T, G_{forward}$`)`;
3 $G_{backward} \leftarrow$ `buildComposeGraph(`$T, S, \mathcal{M}$`)`;
4 $M_{T,S} \leftarrow$ `shortestPathCompose(`$T, S, G_{backward}$`)`;
5 $M_{S,T} \leftarrow M_{S,T} \cup$ `inverse(`$M_{T,S}$`)`;
6 **return** $M_{S,T}$;

Datenmanagementpatterns in Simulationsworkflows

Peter Reimann und Holger Schwarz

Institut für Parallele und Verteilte Systeme, Universität Stuttgart
Vorname.Nachname@ipvs.uni-stuttgart.de

Abstract: Simulationsworkflows müssen oftmals große Datenmengen verarbeiten, die in einer Vielzahl proprietärer Formate vorliegen. Damit diese Daten von den im Workflow eingebundenen Programmen und Diensten verarbeitet werden können, müssen sie in passende Formate transformiert werden. Dies erhöht die Komplexität der Workflowmodellierung, welche i.d.R. durch die Wissenschaftler selbst erfolgt. Dadurch können sich diese weniger auf den Kern der eigentlichen Simulation konzentrieren. Zur Behebung dieses Defizits schlagen wir einen Ansatz vor, mit dem die Aktivitäten zur Datenbereitstellung in Simulationsabläufen abstrakt modelliert werden können. Wissenschaftler sollen keine Implementierungsdetails, sondern lediglich die Kernaspekte der Datenbereitstellung in Form von Patterns beschreiben. Die Spezifikation der Patterns soll dabei möglichst in der Sprache der mathematischen Simulationsmodelle erfolgen, mit denen Wissenschaftler vertraut sind. Eine Erweiterung des Workflowsystems bildet die Patterns automatisch auf ausführbare Workflowfragmente ab, welche die Datenbereitstellung umsetzen. Dies alles reduziert die Komplexität der Modellierung von Simulationsworkflows und erhöht die Produktivität der Wissenschaftler.

1 Einleitung

In den vergangenen Jahren haben sich im Unternehmensumfeld Workflows zur Beschreibung und Ausführung von Geschäftsprozessen durchgesetzt. Immer häufiger wird diese Technologie auch in der Wissenschaft eingesetzt, z.B. um Simulationsabläufe zu beschreiben [Gö11]. Charakteristisch für solche Simulationsabläufe sind komplexe mathematische Berechnungen sowie verschiedene Datenverwaltungs- und Datenbereitstellungsaktivitäten. Oftmals müssen große Datenmengen, die in einer Vielzahl proprietärer Formate vorliegen, aus verschiedenen Quellen verarbeitet werden. Damit diese Daten durch einen Simulationsworkflow und den von ihm eingebundenen Programmen und Diensten verarbeitet werden können, müssen sie in passende Formate transformiert werden. Dies erhöht die Komplexität der Workflowmodellierung, welche i.d.R. durch die an den Simulationsergebnissen interessierten Wissenschaftler selbst erfolgt. Wissenschaftler besitzen aber selten vertiefte Kenntnisse im Bereich der Workflowmodellierung oder der Datenverwaltung. Daher impliziert diese hohe Komplexität der Workflowmodellierung einerseits, dass sich Wissenschaftler weniger auf ihre Kernaufgaben konzentrieren können, d.h. auf die Entwicklung von mathematischen Simulationsmodellen, die Durchführung der Simulationen und die Interpretation der Ergebnisse. Andererseits birgt eine komplexe Datenverwaltung auch die Gefahr einer großen Fehlerrate in sich [Re11].

Um diese Defizite bei der Modellierung von Simulationsworkflows zu beheben, schlagen wir einen Ansatz vor, der es erlaubt, die Aktivitäten zur Datenbereitstellung in Simulationsabläufen abstrakt zu modellieren. Wissenschaftler sollen keine Implementierungsdetails, sondern lediglich die Kernaspekte der Datenbereitstellung in Form von Datenmanagementpatterns direkt beschreiben und wenige relevante Parameter festlegen müssen. Die Parametrisierung der Patterns soll dabei möglichst in der Sprache der jeweiligen Simulationsmodelle und Simulationstechnik erfolgen, mit denen Wissenschaftler besser umgehen können als mit den Sprachen zur Workflow- oder Datenmodellierung. Wenn für eine Simulation beispielsweise Daten von einem Rechner auf einen anderen transferiert werden müssen, so nutzen Wissenschaftler hierfür ein entsprechendes Datentransferpattern. Als Parametrisierung dieses Patterns werden beispielsweise der Pfad einer zu transferierenden Datei sowie das Programm, für welches die Datei bereitgestellt werden soll, festgelegt. Diese Informationen über Patterns und ihre Parametrisierungen sowie zusätzliche Metadaten werden von der Modellierungsumgebung und der Ausführungsumgebung des Simulationsworkflows genutzt, um die Patterns regelbasiert und möglichst automatisch auf ausführbare Workflowfragmente abzubilden, welche die Datenbereitstellung umsetzen. Auf Basis der Umsetzung dieses Abstraktionskonzepts in einem Simulationsworkflowsystem und auf Basis dessen Anwendung auf eine Simulation von Strukturänderungen in Knochen bewerten und diskutieren wir schließlich den vorgestellten Ansatz.

Dieser Beitrag ist wie folgt gegliedert: Zunächst gibt Abschnitt 2 einen Einblick in die Welt der Simulationsworkflows, insbesondere in die wesentlichen Anforderungen an die Datenbereitstellung in solchen Workflows. Die Datenmanagementpatterns sowie die auf ihnen aufbauende Abstraktionsunterstützung stehen im Mittelpunkt von Abschnitt 3. In Abschnitt 4 erläutern wir beispielhaft die regelbasierte Abbildung der Patterns auf ausführbare Workflowfragmente und stellen dabei die Ergebnisse der Bewertung und Diskussion des vorgestellten Ansatzes vor. Verwandte Arbeiten werden in Abschnitt 5 diskutiert, bevor der Beitrag in Abschnitt 6 mit einem Fazit und einem Ausblick abschließt.

2 Datenbereitstellung in Simulationsworkflows

Eine Abstraktionsunterstützung für die Datenbereitstellung in Simulationsworkflows muss eine Reihe spezifischer Anforderungen berücksichtigen. Wir leiten die Diskussion dieser Anforderungen mit einem Beispiel ein, das die wesentlichen Aspekte veranschaulicht.

2.1 Workflows für eine gekoppelte Simulation von Strukturänderungen in Knochen

Die Untersuchung mancher komplexer Probleme erfordert die Kopplung von Simulationsmodellen verschiedener wissenschaftlicher Anwendungsgebiete. Als Beispiel betrachten wir Untersuchungen zu Strukturänderungen in Knochen, die z.B. bei Heilungsprozessen nach Knochenbrüchen relevant sind [Kr11]. Diese Simulation koppelt Simulationsmodelle der Anwendungsgebiete *Biomechanik* und *Systembiologie* und berechnet die Struktur

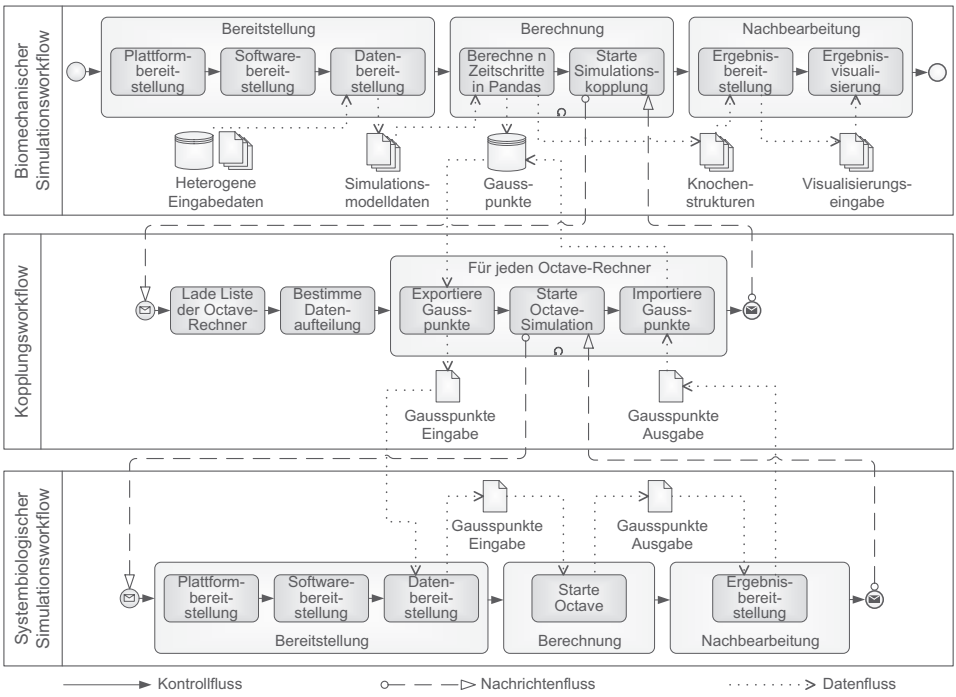


Abbildung 1: Workflows für eine gekoppelte Simulation von Strukturänderungen in Knochen

eines Knochens zeitabhängig unter einer veränderlichen Last. Das biomechanische Simulationsmodell beschreibt das Verhalten von Knochen auf einer makroskopischen Gewebeebene, wobei der Massenaustausch zwischen porösen Festkörpern und darin enthaltenen Flüssigkeiten im Vordergrund steht. Das feingranulare systembiologische Simulationsmodell bestimmt die Bildung oder den Abbau des Knochengewebes auf Basis der Interaktion von Zellen. Zu dem in Abbildung 1 gezeigten gekoppelten Simulationsprozess gehören ein *biomechanischer* und ein *systembiologischer Simulationsworkflow* sowie ein *Kopplungsworkflow*. Die biomechanische Simulation nutzt das auf der Finite-Elemente-Methode (FEM) basierende Pandas-Rahmenwerk[1] und berechnet für mehrere Zeitschritte jeweils die mechanische Belastungsverteilung im Knochengewebe. Die Belastungsverteilung des letzten berechneten Zeitschritts bildet anschließend die Eingabe für die systembiologische Simulation, die mithilfe der Rechenumgebung GNU Octave[2] umgesetzt wird. Deren Ergebnisse werden genutzt, um die Knochenkonfiguration des biomechanischen Modells anzupassen und Belastungsverteilungen für weitere Zeitschritte zu berechnen.

Beide Simulationsworkflows sind in die Phasen *Bereitstellung*, *Berechnung* und *Nachbearbeitung* aufgeteilt. In der Bereitstellungsphase werden zunächst notwendige Plattformen erzeugt, insbesondere Verzeichnisstrukturen auf den jeweiligen Rechnern, in welche die

[1] http://www.mechbau.uni-stuttgart.de/pandas/index.php
[2] http://www.gnu.org/software/octave/

nachfolgenden Aktivitäten Softwarepakete für Pandas bzw. Octave installieren sowie Eingabedaten kopieren können. Im biomechanischen Simulationsworkflow beschreiben diese Eingabedaten das entsprechende Simulationsmodell. Der Workflow muss Daten aus mehreren heterogenen Datenquellen (relationale Datenbanken, strukturierte und unstrukturierte Textdokumente) extrahieren und diese in Dateiformate transformieren, mit denen Pandas arbeiten kann. In der sequentiellen Schleife der Berechnungsphase berechnet Pandas die mechanischen Belastungsverteilungen der ersten n Zeitschritte und speichert sie in einer Datenbank. Die Daten sind in die einzelnen Zeitschritte und in mehrere Tausend oder Millionen Elemente eines FEM-Gitters strukturiert. Jedes Element enthält mehrere Gausspunkte als Stützstellen für die Interpolation der Belastungsverteilung im Knochen. Für jeden Zeitschritt und jeden Gausspunkt speichert Pandas Werte zu zehn Variablen des biomechanischen Simulationsmodells. Die systembiologische Simulation benötigt nur die Werte des letzten berechneten Zeitschritts und nur für zwei der zehn Variablen, was entsprechende Filterungen der Daten nötig macht. Da die systembiologische Simulation feingranularer und somit rechenintensiver ist, wird sie zudem parallelisiert und es findet eine Aufteilung der Daten auf mehrere Rechner und Instanzen von Octave statt.

Der *Kopplungsworkflow* steuert diese Filterung und Aufteilung der Daten. Er lädt dazu eine Liste aller verfügbaren Octave-Rechner aus einem Repository und bestimmt die Aufteilung der Daten auf diese Rechner gemäß der Vorgaben der Wissenschaftler. Die nachfolgende parallele Schleife iteriert über die Liste der Octave-Rechner. In jedem Schleifendurchlauf exportiert der Workflow die passenden Daten aus der Datenbank der Gausspunkte und speichert sie in eine CSV-Datei (Comma-separated Values) auf dem Pandas-Rechner. Anschließend startet er eine neue Instanz des systembiologischen Simulationsworkflows. Dieser stellt die notwendigen Plattformen und Softwarepakete für Octave bereit und kopiert in der Datenbereitstellung die CSV-Datei auf den jeweiligen Octave-Rechner. Anschließend startet er die Software Octave, welche mit der CSV-Datei als Eingabe die geänderten Werte der Gausspunkte in einer weiteren CSV-Datei speichert. Diese wird in der Nachbearbeitungsphase zurück auf den Pandas-Rechner kopiert. Der Kopplungsworkflow importiert die darin enthaltenen Daten in die Datenbank der Gausspunkte, womit die Knochenkonfiguration des biomechanischen Modells angepasst wird.

Der biomechanische Simulationsworkflow wiederholt diesen Prozess, bis alle Zeitschritte der Simulation betrachtet wurden. Zusätzlich zu den mechanischen Belastungsverteilungen speichert der Workflow auch die Knochenstrukturen für alle Zeitschritte in einem Pandas-spezifischen Dateiformat. In der Ergebnisbereitstellung werden diese Daten in Datenformate transformiert, mit denen das von den Wissenschaftlern gewünschte Visualisierungstool arbeiten kann, und bei Bedarf auf den Rechner dieses Tools kopiert.

2.2 Anforderungen an die Datenbereitstellung in Simulationsworkflows

Die Workflows für die Simulation von Strukturänderungen in Knochen enthalten eine Vielzahl an Datenmanagement- und Datenbereitstellungsschritten, welche Daten in vielen heterogenen Datenformaten verarbeiten. Solch eine komplexe Datenlandschaft ist typisch für Simulationen, die über verschiedene Anwendungsbereiche gekoppelt sind, da jeder An-

wendungsbereich eigene Anforderungen wie auch Lösungen für das Datenmanagement besitzt. Wissenschaftler modellieren ihre Simulationsworkflows häufig selbst und müssen dabei auch einen Großteil des Datenmanagements spezifizieren oder implementieren. Sie besitzen zwar eine hohe Expertise in ihrem Anwendungsbereich der Simulationsmodellierung, weisen aber i.d.R. eingeschränkte Fähigkeiten im Bereich der Workflowmodellierung und des Datenmanagements auf. Dies kann eine hohe Fehlerrate implizieren. Zudem verschwenden Wissenschaftler Zeit, die sie eigentlich für ihre Kernaufgaben aufbringen möchten, nämlich die Simulationen selbst. Eine essenzielle Anforderung an die Datenbereitstellung in Simulationsworkflows ist folglich eine geeignete *Abstraktionsunterstützung für die Definition von Datenbereitstellungsschritten*. Diese sollte Wissenschaftler zum Einen davon befreien, Implementierungsdetails der Datenbereitstellung zu spezifizieren. Zum Anderen soll sie Wissenschaftler dazu befähigen, mehr in der Sprache ihrer Simulationsmodelle zu arbeiten, und weniger in den Sprachen der Workflow- oder Datenmodellierung. Es muss also die Brücke zwischen der Welt der Simulationen sowie der Welt der Workflows und Daten geschlagen werden. Die hierfür erforderliche Abstraktionsunterstützung steht im Fokus dieses Beitrags. Bei deren Umsetzung müssen jedoch noch weitere Anforderungen beachtet werden. Wir stellen nachfolgend die wichtigsten drei vor:

- Die erste Anforderung ergibt sich direkt aus dem Wunsch, Simulationen sowie heterogene Daten- und Anwendungslandschaften aus verschiedenen Anwendungsbereichen zu koppeln. Hierfür muss eine Abstraktionsunterstützung hinreichend *generisch und erweiterbar* sein und alle Anwendungsbereiche unterstützen [Re11].
- Die Gesamtgröße der in Simulationsworkflows involvierten Daten kann zwischen wenigen 100 KB und einigen Terabytes liegen sowie sich während des Ablaufs einer Simulation ständig ändern. Dies führt zwangsläufig zu Anforderungen bzgl. der *Effizienz* der Datenverarbeitung und bzgl. der Unterstützung entsprechender Optimierungsmöglichkeiten für diese Datenverarbeitung [Vr07].
- Wissenschaftler führen häufig ad-hoc Änderungen an Workflows während deren Laufzeit durch [SK10]. Dazu muss eine ausreichende Überwachung der Workflowausführungen möglich sein. Ein weiterer wichtiger Aspekt ist die Sicherstellung der Wiederholbarkeit einer Simulation und der Nachvollziehbarkeit ihrer Ergebnisse, was den Begriff Provenance geprägt hat [Fr08]. Diese beiden Aspekte können unter dem Begriff *transparentes Datenmanagement* zusammengefasst werden.

3 Abstraktionsunterstützung durch Datenmanagementpatterns

In diesem Abschnitt stellen wir unseren Ansatz vor, Datenmanagementpatterns für eine Abstraktionsunterstützung der Datenbereitstellung in Simulationsworkflows zu nutzen. Um Datenmanagementpatterns in Simulationsworkflows zu identifizieren, haben wir sowohl eine Reihe von Szenarien aus der Literatur [TDG07, SR09] als auch reale Simulationsprozesse analysiert. Neben der in Abschnitt 2.1 vorgestellten Simulation gehört hierzu insbesondere das in [RK11] betrachtete Beispiel. Im Folgenden erläutern wir zunächst die

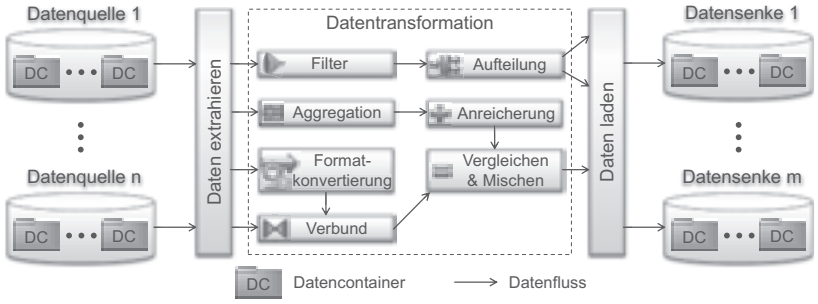


Abbildung 2: Allgemeines Datentransfer- und -transformationspattern

grundlegenden Datenmanagementpatterns. Danach gehen wir auf das zugrundeliegende SIMPL-Rahmenwerk ein (*SimTech - Information Management, Processes, and Languages*) [Re11]. Anschließend wird beschrieben, wie die vorgestellten Patterns in eine umfassendere Patternhierarchie eingegliedert sind und wie diese Hierarchie die Brücke zwischen der Welt der Simulationen und der Welt der Workflows schlagen kann.

3.1 Grundlegende Datenmanagementpatterns in Simulationsworkflows

Die allgemeine Form des *Datentransfer- und -transformationspatterns* (Abbildung 2) beschreibt den Transfer von Daten aus einer oder mehreren Datenquellen in eine oder mehrere Datensenken. Dabei können auf beiden Seiten mehrere Datencontainer angesprochen werden, die jeweils eine identifizierbare Datenmenge umfassen, z.B. eine Datenbanktabelle oder eine Datei. Zu einem solchen Pattern gehören auch ETL-Operationen, mit denen Daten aus den Datenquellen extrahiert, geeignet transformiert und in die Datensenken geladen werden [Re11, TDG07]. In den in Abschnitt 2.1 beschriebenen Workflows kommen z.B. häufig Formatkonvertierungen und Filter als ETL-Operationen zum Einsatz. Weiterhin lassen sich dort drei Varianten des Datentransfer- und -transformationspatterns unterscheiden, je nachdem ob (1) Daten von einem Datencontainer auf einen anderen übertragen werden, ob sie (2) von einem Container auf mehrere Container aufgeteilt werden oder ob sie (3) aus mehreren Container in einen Container zusammengeführt werden. Die erste Variante findet sich in den Bereitstellungs- und Nachbearbeitungsphasen der Simulationsworkflows, während der Kopplungsworkflow Daten aufteilt und wieder zusammenführt.

Das Grundprinzip, dem die *Dateniterationspatterns* folgen, ist die Iteration über einer Datenmenge S und die Ausführung eingebetteter Operationen für einzelne Teilmengen von S. Das *Parallele Dateniterationspattern* (Abbildung 3) umfasst eine Aufteilungs-, eine Operations- und eine Mischphase. Das Ziel ist die parallele Bearbeitung einer Operation auf mehreren Ressourcen. In der *Aufteilungsphase* wird ein Datenaufteilungspattern genutzt, um S in n Teilmengen $T_i \subseteq S$ aufzuteilen und diese auf die Ressourcen zu verteilen. In der *Operationsphase* dienen diese T_i als Eingabe für die anzuwendenden Operationen, die jeweils das zugehörige $T_i{}'$ als Ergebnis liefern. Das anschließende Datenmischpattern

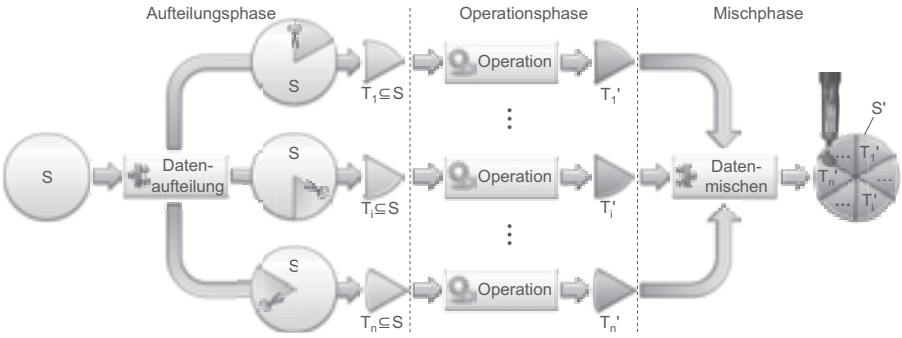


Abbildung 3: Paralleles Dateniterationspattern. S, S', T_i und T_i' entsprechen Datenmengen.

sorgt in der *Mischphase* für die Integration der Teilmengen $T_1{}'$ bis $T_n{}'$, womit sich die Ergebnisdatenmenge S' ergibt. Bei der in Abschnitt 2.1 beschriebenen Simulation ist dieses Pattern im Kopplungsworkflow zu finden. Dabei entsprechen die Daten in der Datenbank zu Gausspunkten den Datenmengen S und S'. Der Aufruf des systembiologischen Simulationsworkflows stellt die Operation dar, während die CSV-Dateien die Rolle der Teilmengen T_i bzw. $T_i{}'$ einnehmen. Die zweite Variante, das *Sequentielle Dateniterationspattern*, umfasst weder eine Parallelverarbeitung noch eine Aufteilung der Datenmenge S, sondern die Iterationsschritte werden nacheinander ausgeführt. Solch ein Pattern kann im Beispiel aus Abschnitt 2.1 sinnvoll sein, um Berechnungen sequentiell durchzuführen, falls für den gewünschten Parallelisierungsgrad zu wenig Octave-Rechner zur Verfügung stehen.

3.2 Das SIMPL-Rahmenwerk für eine Abstraktionsunterstützung

Das SIMPL-Rahmenwerk bietet eine Reihe von Abstraktionsunterstützungen für die Definition der Datenbereitstellung in Simulationsworkflows an [Re11]. Abbildung 4 zeigt, wie es die Architektur eines Simulationsworkflowsystems erweitert. Zur besseren Lesbarkeit lassen wir Komponenten der Gesamtarchitektur aus, die für die Datenbereitstellung weniger relevant sind. Dies betrifft z.B. eine Komponente für das dynamische Binden von Services oder Ressourcen [Gö11]. Die im Rahmen dieses Beitrags relevanten Hauptkomponenten sind die *Workflowmodellierumgebung*, die *Workflowausführungsumgebung*, die *regelbasierte Patterntransformationsumgebung* und der *Service Bus*. Im Folgenden erläutern wir zuerst die Datenzugriffsabstraktion, während die darauf aufbauende Abstraktionsunterstützung mittels Datenmanagementpatterns und deren regelbasierten Transformation auf ausführbare Workflows im nächsten Teilabschnitt diskutiert wird.

Die Datenzugriffsabstraktion basiert auf dem *SIMPL Core*, einer Erweiterung des Service Bus. Diese stellt Wissenschaftlern generische Operationen für den einheitlichen Zugriff auf externe Datenressourcen zur Verfügung. Hierzu gehören (1) *IssueCommand* für das Absenden von Befehlen zur Datenmanipulation oder -definition, (2) *RetrieveData* zum Laden von Daten, (3) *WriteDataBack* für das Zurückschreiben dieser Daten sowie

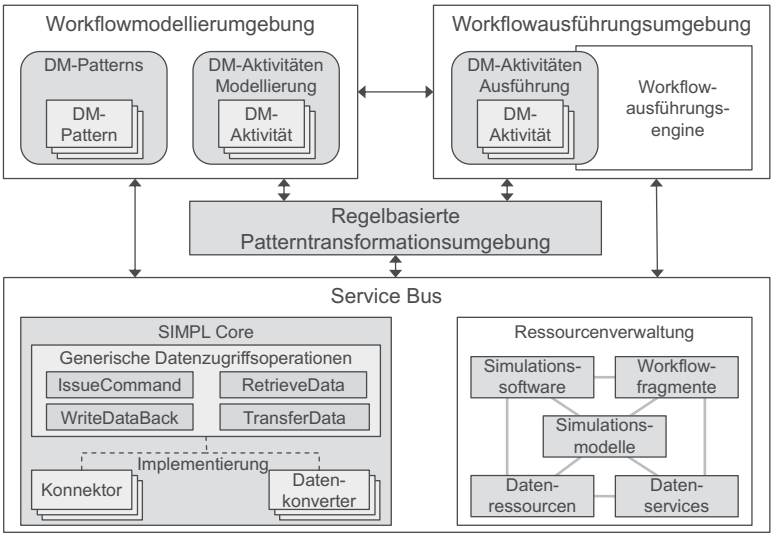


Abbildung 4: Zentrale Komponenten eines durch das SIMPL-Rahmenwerk erweiterten Simulationsworkflowsystems, vgl. [Re11], [Gö11]. Bestandteile des SIMPL-Rahmenwerks sind grau eingefärbt.

(4) *TransferData* für Datentransfers. *Konnektoren* implementieren diese Operationen für bestimmte Datenressourcen und berücksichtigen deren spezifische Zugriffsmechanismen. Für die RetrieveData- und WriteDataBack-Operationen transformieren *Datenkonverter* die Daten vom Format eines Konnektors in das der Client-Anwendung und umgekehrt. Zusätzlich erweitert SIMPL die Ressourcenverwaltung um Metadaten zur expliziten Beschreibung von *Datenressourcen*. Diese Metadaten bilden insbesondere die generischen Zugriffsoperationen auf die konkreten Zugriffsmechanismen einzelner Datenressourcen ab, indem sie u.a. jede Datenressource mit dem passenden Konnektor und Datenkonvertern verknüpfen. Damit die Funktionalität des SIMPL Core auch direkt in Workflowmodellen genutzt werden kann, bieten sowohl die Modellier- als auch die Ausführungsumgebung eine Unterstützung für Datenmanagementaktivitäten (*DM-Aktivitäten*). Die Aktivitäten entsprechen dabei sinngemäß den vier Operationen des SIMPL Core. Sie sind jeweils einer Datenressource zugeordnet, beinhalten einen Befehl in deren Befehlssprache – z.B. in SQL oder XQuery – und senden diesen Befehl bei der Ausführung der Aktivität über den SIMPL Core an die Ressource. Als Alternative können Workflowmodellierer nach wie vor Services für das Datenmanagement verwenden – sog. *Datenservices*.

3.3 Datenmanagementpatterns als Brücke zwischen Simulationen und Workflows

Trotz der erläuterten Datenzugriffsabstraktion mittels DM-Aktivitäten müssen Wissenschaftler in ihren Workflowmodellen viele Details der Datenbereitstellung spezifizieren. Verwenden Wissenschaftler Datenservices, müssen sie passende Services suchen bzw. An-

Abbildung 5: Hierarchie von Datenmanagementpatterns

forderungsbeschreibungen in einer geeigneten Sprache definieren. Bei DM-Aktivitäten müssen sie Datenmanagementoperationen wie z.B. Datentransformationen oder Datenaufteilungen sogar über die Befehlssprachen der involvierten Datenressourcen beschreiben. Da Sprachen für Anforderungsbeschreibungen und vor allem Befehlssprachen von Datenressourcen i.d.R. wenig mit den Sprachen der Simulationsmodelle zu tun haben, fällt Wissenschaftlern dies häufig schwer. Daher erweitert SIMPL die Workflowmodellierumgebung um eine weitere Komponente. Diese ermöglicht die Nutzung der in Abschnitt 3.1 beschriebenen und weiterer Datenmanagementpatterns (*DM-Patterns*) als Bausteine für Datenbereitstellungsschritte in Simulationsworkflows. Durch die Einteilung der Datenmanagementoperationen in einzelne voneinander abgrenzbare Patterns können wir für jedes Pattern die Freiheitsgrade in der Spezifikation dieser Operationen einschränken. Dies reduziert die Komplexität der Spezifikation und ermöglicht eine weiterführende Abstraktion auf Basis der Patterns. Wissenschaftler können die für sie relevanten Patterns auswählen und in ihre Workflowmodelle einfügen. Sie werden dann für jedes Pattern bei der Spezifikation der konkreten Operation unterstützt. Insbesondere müssen sie nur wenige Parameterwerte angeben, anstatt vollständige Implementierungsdetails zu spezifizieren.

Abbildung 5 ordnet Datenmanagementpatterns in einer Hierarchie an. Je höher die Hierarchieebene, desto mehr Informationen bzgl. den zugrundeliegenden Datenmanagementoperationen und Befehlssprachen werden verdichtet. Dementsprechend müssen Wissenschaftler bei der Spezifikation von Operationen über immer weniger Detailwissen verfügen. Mit diesem steigenden Abstraktionsgrad erhöht sich auch der Bezug zwischen den für Patterns anzugebenden Parameterwerten und den Sprachen der jeweiligen Simulationsmodelle, wobei der Bezug zu den Sprachen der Workflow- oder Datenmodellierung entsprechend geringer wird. Umgekehrt müssen die verdichteten Informationen auf dem Weg nach unten durch die Hierarchie wieder angereichert werden, um auf die Ebene *ausführbarer Workflowfragmente* bzw. *Datenservices* zu kommen. Letztgenannte setzen die Patterns in den Ebenen darüber um und beinhalten dabei viele Implementierungsdetails. Die nächsthöhere Ebene umfasst die in Abschnitt 3.1 beschriebenen *grundlegenden Datenmanagementpatterns*. Die Ebene der *zusammengesetzten Datenmanagementpatterns* nutzt die grundlegenden Patterns als Basis und schafft einen höheren Abstraktionsgrad. Hier können z.B. mehrere Datentransfer- und -transformationspatterns, die das gleiche Ziel für den Datentransfer

definieren, in einem *allgemeinen Datenbereitstellungspattern* zusammengefasst werden. Den stärksten Bezug zu Simulationen und damit den für Wissenschaftler höchsten Abstraktionsgrad stellt die Ebene der *simulationsorientierten Patterns* her. Als Beispiel sei ein Pattern für die Kopplung verschiedener Simulationsmodelle genannt. Wissenschaftler können die Kopplung mit diesem Pattern vollständig über Begriffe spezifizieren, die ihnen aus ihren Simulationsmodellen geläufig sind. Im betrachteten Beispiel aus Abschnitt 2.1 geben sie im Wesentlichen die Abhängigkeiten zwischen den verschiedenen Variablen der beiden Simulationsmodelle sowie den relevanten Zeitschritt an. Weiterhin spezifizieren sie abstrakt, dass die Daten gleichmäßig nach Gausspunkten aufgeteilt werden sollen.

Die *regelbasierte Patterntransformationsumgebung* des SIMPL-Rahmenwerks enthält eine erweiterbare Menge von Abbildungsregeln, welche von Wissenschaftlern parametrisierte Patterns Schritt für Schritt nach unten durch die einzelnen Ebenen der Hierarchie abbilden. Hierbei werden Regeln so lange und ggf. rekursiv angewandt, bis alle Patterns in einem Workflow schließlich durch ausführbare Workflowfragmente oder Datenservices ersetzt wurden. Die Regeln nutzen dabei Metadaten bzgl. *Simulationssoftware*, *Workflowfragmenten*, *Simulationsmodellen*, *Datenressourcen* und *Datenservices* sowie Abhängigkeiten zwischen diesen Metadaten, um die für die Abbildung von Patterns notwendige Informationsanreicherung umzusetzen. Auf diese Weise bildet eine Regel z.B. das simulationsorientierte Pattern für die Spezifikation einer Simulationsmodellkopplung auf ein Paralleles Dateniterationspattern ab. Die Parametrisierung dieses Patterns sowie dessen Abbildung auf einen ausführbaren Workflow diskutieren wir in Abschnitt 4.1.

4 Bewertung und Diskussion der Abstraktionsunterstützung

Um die vorgestellte Abstraktionsunterstützung bewerten zu können, entwickelten wir einen Prototypen des SIMPL-Rahmenwerks. Dieser Prototyp nutzt die Workflowsprache Business Process Execution Language (BPEL) [JE07], das Workflowmodelliertool Eclipse BPEL Designer[3] Version 0.8.0 und die Workflowausführungsengine Apache Orchestration Director Engine[4] (ODE) Version 1.3.5. Im nächsten Teilabschnitt illustrieren wir den Einsatz des Parallelen Dateniterationspatterns im Kopplungsworkflow aus Abschnitt 2.1 und wie das resultierende ausführbare Workflowmodell aussieht. Anschließend bewerten wir, inwieweit dieses Pattern eine für Wissenschaftler geeignete Abstraktionsunterstützung zur Definition des Kopplungsworkflows darstellt. Schließlich diskutieren wir unseren Ansatz bzgl. der in Abschnitt 2.2 beschriebenen Anforderungen.

4.1 Umsetzung für die Simulation von Strukturänderungen in Knochen

Das Dateniterationspattern ersetzt im Kopplungsworkflow alle Aktivitäten zwischen dem Eingang der Nachricht vom biomechanischen Simulationsworkflow und dem Rücksenden

[3] http://www.eclipse.org/bpel/
[4] http://ode.apache.org/

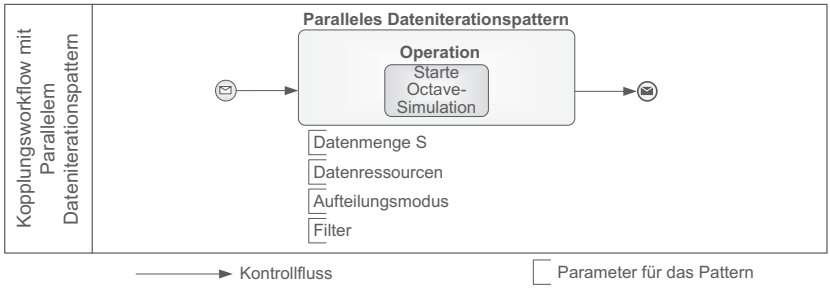


Abbildung 6: Einsatz des Parallelen Dateniterationspatterns im Kopplungsworkflow aus Abbildung 1

der Antwortnachricht (siehe Abbildung 6). Der Aufruf des systembiologischen Simulationsworkflows ist die in das Pattern eingebettete Operation. Mithilfe des Patterns reduziert sich die Anzahl der für Wissenschaftler sichtbaren Aktivitäten und der zu spezifizierenden Parameter, was bei der Definition des Workflowmodells eine erhebliche Erleichterung darstellt. Der Wert des ersten Parameters identifiziert die *Datenmenge* S, über die iteriert werden soll. Der Wissenschaftler gibt hier abstrakt die mechanische Belastungsverteilung im Knochen an, welche bei den Metadaten zum biomechanischen Simulationsmodell als Ausgabedaten registriert ist. Der zweite Parameter bestimmt die *Datenressourcen*, auf welche die Datenmenge S verteilt werden soll. Hier gibt der Wissenschaftler mithilfe von Metadaten zu Services eine Referenz auf einen geeigneten Repositoryservice an, der eine Liste der verfügbaren Octave-Rechner liefert. Mithilfe der Metadaten zum biomechanischen Simulationsmodell kann der Wissenschaftler alle weiteren Parameter des Patterns über Begriffe festlegen, die ihm aus diesem Simulationsmodell geläufig sind. Beim *Aufteilungsmodus* gibt er an, dass die Datenmenge S gleichverteilt nach Gausspunkten aufgeteilt werden soll. Der Parameter *Filter* ermöglicht die Einbindung weiterer, vor der Datenaufteilung durchzuführender Filteroperationen für S. Hier definiert der Wissenschaftler abstrakt die beiden in Abschnitt 2.1 beschriebenen Filter: einen nach dem letzten berechneten Zeitschritt und einen nach den relevanten Variablen des biomechanischen Simulationsmodells.

Über die Anwendung von Abbildungsregeln entsteht das in Abbildung 7 dargestellte ausführbare Workflowmodell. Nachdem der Workflow über den Repositoryservice die Liste der verfügbaren Octave-Rechner geladen hat, holt er sich über eine SIMPL RetrieveData-Aktivität die ID des letzten berechneten Zeitschritts aus der Datenbanktabelle zu Gausspunkten. Dazu setzt er eine SQL SELECT-Anweisung ab, die eine Aggregatfunktion für die maximale Zeitschritt-ID beinhaltet. Da in der Tabelle zu Gausspunkten Daten für mehrere Simulationen gespeichert sein können, ist zusätzlich ein Filterprädikat bzgl. der aktuellen Simulations-ID erforderlich. Die nächste RetrieveData-Aktivität speichert die Anzahl der relevanten Gausspunkte in eine Workflow-Variable. Die SELECT-Anweisung beinhaltet eine entsprechende Aggregatfunktion sowie Filterprädikate nach der Simulations-ID und nach dem Zeitschritt. Anschließend bestimmt ein XPath-Ausdruck in einer BPEL Assign-Aktivität die Anzahl der Gausspunkte pro Octave-Rechner. In der Aufteilungsphase der parallelen Dateniteration realisiert eine IssueCommand-Aktivität für jeden Octave-Rechner den Export der Daten aus der Datenbanktabelle in eine CSV-Datei.

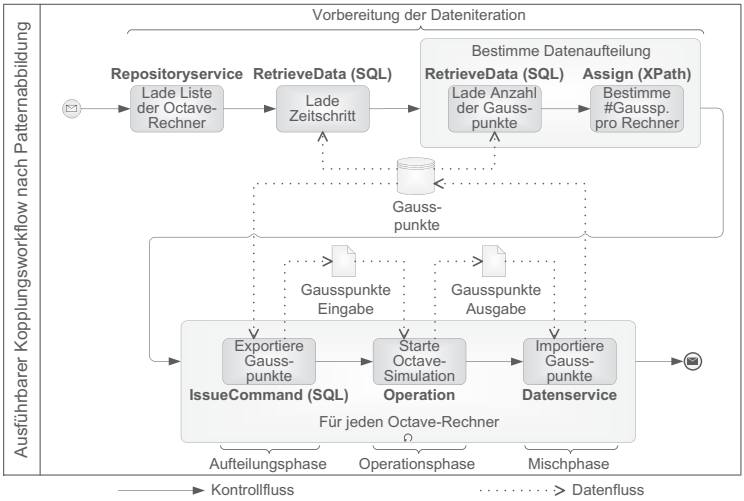


Abbildung 7: Ausführbarer Kopplungsworkflow nach der regelbasierten Abbildung von Patterns

Die eingebettete SQL-Anweisung beinhaltet die Projektionen auf die relevanten Variablen des biomechanischen Simulationsmodells, zwei Filterprädikate nach dem Zeitschritt und der Simulations-ID sowie LIMIT- und OFFSET-Ausdrücke für die Extraktion der richtigen Gausspunkte. Anschließend startet der Kopplungsworkflow den systembiologischen Simulationsworkflow. Sobald dieser beendet ist, nutzt der Kopplungsworkflow einen proprietären Datenservice, um die resultierende CSV-Datei in die Datenbank zu portieren.

4.2 Bewertung der Abstraktionsunterstützung

Modellieren Wissenschaftler direkt ausführbare Workflowmodelle wie den in Abbildung 7 gezeigten Kopplungsworkflow müssen sie auch alle in Abschnitt 4.1 beschriebenen Details der einzelnen Workflowaktivitäten, Serviceaufrufe, SQL-Anweisungen und XPath-Ausdrücke festlegen. Dies würde einen großen, für die Wissenschaftler nicht akzeptablen Aufwand darstellen. Die Modellierung mithilfe des Parallelen Dateniterationspatterns stellt für Wissenschaftler eine erhebliche Vereinfachung dar, da sie solche Implementierungsdetails nicht explizit definieren müssen. Insbesondere müssen sie deutlich weniger Workflowaktivitäten modellieren und können den Großteil des Datenmanagements über Begriffe aus ihren Simulationsmodellen und damit in einem hohen Abstraktionsgrad definieren. Die Hierarchie von Datenmanagementpatterns, die Umformungsregeln und die Metadaten der SIMPL Ressourcenverwaltung schlagen zudem die Brücke zwischen der Welt der Simulationen – dem Pattern – sowie der Welt der Workflows und Daten – dem ausführbaren Workflowmodell. Insgesamt reduziert unser Ansatz die Komplexität in der Workflowmodellierung deutlich. Dadurch sparen Wissenschaftler Zeit und können sich besser auf ihre Kernprobleme konzentrieren, nämlich die eigentlichen Simulationen.

Der auf einer Hierarchie und auf Abbildungsregeln basierende Ansatz ermöglicht zudem die Trennung der Aufgaben entsprechend der Kenntnisse verschiedener Personengruppen. So nutzen Wissenschaftler ihre Kenntnisse im Bereich der Simulationsmodellierung, um Patterns der höchsten Hierarchieebene zu parametrisieren. IT-Experten entwickeln die ausführbaren Workflowfragmente und Services der untersten Ebene und die für diese Ebene genutzten Abbildungsregeln sowie für deren Anwendung benötigte Metadaten. Für die Hierarchieebenen dazwischen können Workflowfragmente, Abbildungsregeln und Metadaten von Experten der Schnittstellen zwischen Simulation und IT entwickelt werden. Dabei dienen die voneinander abgrenzbaren Patterns auch als Mittel, um die jeweils zu erfüllenden Anforderungen zwischen diesen Personengruppen zu kommunizieren.

4.3 Diskussion bezüglich der Anforderungen

Die Anforderung, dass die Abstraktionsunterstützung *generisch in verschiedenen Anwendungsbereichen und Problemfeldern* eingesetzt werden kann, wird in unserem Ansatz im Wesentlichen durch die Wahl der generischen Patterns in der in Abbildung 5 dargestellten Hierarchie unterstützt. Diese Patterns und deren Modellierkonstrukte bzw. Parameter können unabhängig vom Problem oder dem Anwendungsgebiet verwendet werden. Die einzelnen von den Wissenschaftlern definierten Parameterwerte sowie die Abbildungsregeln und die ausführbaren Workflowfragmente bzw. Services berücksichtigen die problem- oder anwendungsgebietspezifischen Aspekte. Zudem ermöglicht die Trennung zwischen Patterns und deren Umsetzung in diesem regelbasierten Ansatz die Erweiterung um weitere problemspezifische Abbildungsregeln und Workflowfragmente bzw. Services.

Der regelbasierte Ansatz zur Abbildung von Patterns auf ausführbare Workflowmodelle ermöglicht die nahtlose Integration entsprechender regelbasierter Optimierungsentscheidungen, wie sie z.B. bei Techniken zur Restrukturierung und Optimierung von Workflowmodellen verwendet werden [Vr07]. Damit kann die *Effizienz der Datenverarbeitung* in Simulationsworkflows erhöht werden. Als Beispiel betrachten wir ein Datentransfer- und -transformationspattern, das auf einen Workflowschritt für eine Datenformatkonvertierung und einen Schritt für den eigentlichen Datentransfer aufgeteilt wird. Reduziert die Datenformatkonvertierung die Datengröße, ist es i.d.R. sinnvoll, sie vor dem Datentransfer auszuführen und umgekehrt. Außerdem können die Parametrisierungen der Patterns um Beschreibungen nichtfunktionaler Anforderungen, z.B. bzgl. der Qualität von Daten [Re12], ergänzt und diese in den Regeln als Optimierungsentscheidungen berücksichtigt werden.

Während unser Ansatz die Anzahl und Komplexität der für Wissenschaftler sichtbaren Workflowaktivitäten reduziert, kann dies zu einem Problem bzgl. des *transparenten Datenmanagements* führen. Die Workflowausführungsumgebung kennt ausschließlich die durch die regelbasierte Abbildung von Patterns entstehenden komplexeren Workflowmodelle. Damit ist die Korrelation für die in der Modellierumgebung sichtbaren Patterns und die in der Ausführungsumgebung gesammelten Audit- bzw. Provenance-Informationen nicht mehr per se gegeben. Damit Wissenschaftler dennoch Workflowausführungen überwachen bzw. Simulationsergebnisse nachvollziehen können, müssen Ausführungsumgebungen erweitert werden und diese Informationen für die Patterns aggregieren.

5 Verwandte Arbeiten

Wir haben in diesem Beitrag eine auf Patterns basierende Abstraktionsunterstützung für die Datenbereitstellung in Simulationsworkflows vorgestellt. Dementsprechend gehen wir in diesem Abschnitt auf verwandte Arbeiten in den Bereichen Workflowsysteme für wissenschaftliche Prozesse und Workflow-Patterns ein. Systeme wie das *Scientific Data Management Center* sowie das dazugehörige Workflowsystem Kepler ermöglichen ebenfalls die Definition und Ausführung wissenschaftlicher Prozesse [Sh07, Lu06]. Die beiden Systeme betrachten aber Prozesse zur Analyse von Daten, die von Simulationen oder Experimenten erzeugt wurden. Im Gegensatz zu unserem Ansatz beschäftigen sie sich nicht mit Simulationsworkflows als Vorstufe solcher Datenanalysen und vor allem nicht mit einer patternbasierten Abstraktionsunterstützung für die Datenbereitstellung in Simulationsworkflows. Das System Microsoft Trident ist hingegen universell für alle Arten von wissenschaftlichen Prozessen und damit auch für Simulationsworkflows einsetzbar [Ba08]. Allerdings fehlt auch hier der Bezug zu einer patternbasierten Abstraktionsunterstützung.

Russel et. al. beschreiben allgemeine Datenpatterns in Workflows [Ru05]. Allerdings betrachten sie in erster Linie Patterns, die typisch für Geschäftsprozesse sind, und nicht für die Datenbereitstellung in Simulationsworkflows. Es handelt sich um sehr feingranulare Patterns, die vor allem bei der Evaluation verschiedener Workflowsprachen und Workflowsysteme als Bewertungsgrundlage dienen, inwieweit diese die Patterns unterstützen. Z.B. werden die Fragen gestellt, ob Workflowaktivitäten bzw. Workflowinstanzen Daten untereinander per Wert oder per Referenz übertragen können. Bezogen auf unseren Ansatz klassifizieren diese feingranularen Patterns eher Implementierungsdetails in Workflowfragmenten auf der untersten Ebene der in Abbildung 5 dargestellten Hierarchie von Datenmangementpatterns. Sie sind also nicht für eine Abstraktionsunterstützung angedacht.

6 Fazit und Ausblick

In diesem Beitrag haben wir einen generischen Ansatz vorgestellt, mit dem Wissenschaftler die Datenbereitstellung in Simulationsworkflows abstrakt modellieren können. Kern dieses Ansatzes bildet eine Hierarchie von Datenmanagementpatterns. Das Workflowsystem bildet Parametrisierungen dieser Patterns über Abbildungsregeln automatisch auf ausführbare Workflowfragmente ab. Über die prototypische Realisierung dieses patternbasierten Ansatzes haben wir gezeigt, dass Wissenschaftler deutlich weniger Workflowschritte wie auch Implementierungsdetails der Datenbereitstellung definieren müssen. Darüber hinaus können sie die Parameterwerte eher in den Sprachen der jeweiligen Simulationsmodelle angeben, mit denen sie besser umgehen können als mit den Sprachen zur Workflow- oder Datenmodellierung. Dies reduziert die Komplexität der Modellierung von Simulationsworkflows, und Wissenschaftler können sich wieder verstärkt auf die eigentliche Simulationsproblematik konzentrieren. Als nächsten Schritt werden wir unseren Ansatz in weiteren Beispielen für Simulationsworkflows einsetzen, um dessen universelle Einsetzbarkeit genauer zu evaluieren. Weiterhin werden wir Integrationsmöglichkeiten von Optimierungsentscheidungen für eine effizientere Datenverarbeitung untersuchen.

Danksagung: Die Autoren danken der Deutschen Forschungsgemeinschaft für die Förderung des Projekts im Rahmen des Exzellenzclusters Simulation Technology. Weiterhin danken wir Michael Reiter und Christoph Stach für ihre hilfreichen Korrekturvorschläge sowie Henrik Pietranek für die Umsetzung des Prototyps im Rahmen seiner Diplomarbeit.

Literatur

[Ba08] R. Barga et al. The Trident Scientific Workflow Workbench. In *Tagungsband der 4. International Conference on e-Science*, Indianapolis, IN, 2008.

[Fr08] J. Freire et al. Provenance for Computational Tasks: A Survey. *Computing in Science and Engineering*, 10(3), 2008.

[Gö11] K. Görlach et al. Conventional Workflow Technology for Scientific Simulation. In *Guide to e-Science*, Kapitel 11. Springer, London, UK, 2011.

[JE07] D. Jordan und J. Evdemon. *Web Services Business Process Execution Language Version 2.0*. Organization for the Advancement of Structured Information Standards, 2007.

[Kr11] R. Krause et al. Bone Remodelling: A Combined Biomechanical and Systems-Biological Challenge. *Applied Mathematics and Mechanics*, 11(1), 2011.

[Lu06] B. Ludäscher et al. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience*, 18(10), 2006.

[Re11] P. Reimann et al. SIMPL - A Framework for Accessing External Data in Simulation Workflows. In *Gesellschaft für Informatik (Hrsg.): Datenbanksysteme für Business, Technologie und Web*, Kaiserslautern, Deutschland, 2011.

[Re12] M. Reiter et al. Quality-of-Data-Driven Simulation Workflows. In *Tagungsband der 8. International Conference on e-Science*, Chicago, IL, 2012.

[RK11] J. B. Rommel und J. Kästner. The Fragmentation-Recombination Mechanism of the Enzyme Glutamate Mutase Studied by QM/MM Simulations. *Journal of the American Chemical Society*, 26(133), 2011.

[Ru05] N. Russel et al. Workflow Data Patterns: Identification, Representation and Tool Support. In *Tagungsband der 24. International Conference on Conceptual Modeling (ER 2005)*, Klagenfurt, Österreich, 2005.

[Sh07] A. Shoshani et al. SDM Center Technologies for Accelerating Scientific Discoveries. *Journal of Physics: Conference Series (SciDAC 2007)*, 78(1), 2007.

[SK10] M. Sonntag und D. Karastoyanova. Next Generation Interactive Scientific Experimenting Based on the Workflow Technology. In *Tagungsband der 21. International Conference on Modelling and Simulation (MS 2010)*, Prag, Tschechische Republik, 2010.

[SR09] A. Shoshani und D. Rotem. *Scientific Data Management: Challenges, Technology, and Deployment*. Computational Science Series. Chapman & Hall, 2009.

[TDG07] I. Taylor, E. Deelman und D. Gannon. *Workflows for e-Science - Scientific Workflows for Grids*. Springer, London, UK, 2007.

[Vr07] M. Vrhovnik et al. An Approach to Optimize Data Processing in Business Processes. In *Tagungsband der 33. International Conference on Very Large Data Bases (VLDB 2007)*, Wien, Östereich, 2007.

Lernen häufiger Muster aus intervallbasierten Datenströmen - Semantik und Optimierungen

Dennis Geesen[1], H.-Jürgen Appelrath[1], Marco Grawunder[1], Daniela Nicklas[2]

[1]Informationssysteme,[2]Datenbank- und Internettechnologien
Department für Informatik, Universität Oldenburg, 26121 Oldenburg
{dennis.geesen, appelrath, marco.grawunder, daniela.nicklas}@uni-oldenburg.de

Abstract: Das Erkennen und Lernen von Mustern über Ereignisdatenströmen ist eine wesentliche Voraussetzung für effektive kontextbewusste Anwendungen, wie sie bspw. in intelligenten Wohnungen (Smart Homes) vorkommen. Zur Erkennung dieser Muster werden i.d.R. Verfahren aus dem Bereich des Frequent Pattern Mining (FPM) eingesetzt. Das Erlernen relevanter Muster findet aktuell entweder auf aufgezeichneten Ereignisströmen statt oder wird online mit Hilfe spezieller, an die Besonderheiten der Stromverarbeitung angepasste FPM-Algorithmen durchgeführt. Auf diese Weise muss entweder auf die Onlineverarbeitung verzichtet oder existierende und bewährte effiziente FPM-Algorithmen können nicht eingesetzt werden. In diesem Beitrag stellen wir einen Ansatz vor, der es ermöglicht, beliebige Datenbank-basierte FPM-Algorithmen ohne Anpassung auch auf Datenströmen durchzuführen. Da unsere Semantik auf der bekannten relationalen Algebra basiert, können weitere Optimierungen bspw. durch Anfrageumschreibungen erfolgen. Wir evaluieren den Ansatz im Datenstrom-Framework Odysseus und zeigen, dass bspw. beim Einsatz des FPM-Algorithmus „FP-Growth“ das Lernen in konstanter Zeit erfolgen kann und somit ein kontinuierliches Lernen auf dem Datenstrom möglich ist.

1 Einleitung

Durch den zunehmenden Einsatz von Sensoren und Aktoren in intelligenten Wohnungen oder Alltagsgegenständen werden neuartige Anwendungen ermöglicht, die individuell und situativ handeln und bestimmte Aktionen nicht nur basierend auf Sensorschwellwerten auslösen. Eine bestimmte Lampe wird nicht eingeschaltet, weil es dunkel ist, sondern es wird zusätzlich berücksichtigt, ob der Bewohner diese Lampe in der aktuellen Situation (dem Kontext) überhaupt verwenden würde. Solche kontextbewussten Anwendungen (vgl. [DAS01]) basieren i.d.R. auf einer Fusion von Sensordaten und müssen sich an die verschiedenen Umgebungen, Personen und Einstellungen anpassen, sie erlernen können. Aus zusammen auftretenden Aktionen, die entweder durch den Bewohner ausgelöst werden, wie z.B. das Betätigen eines Schalters, oder durch vorliegende äußere Umstände, z.B. die aktuelle Temperatur und Helligkeit können anschließend Regeln, wie z.B. *wenn der Fernseher an ist und es draußen dunkel, dann schalte die Ecklampe an und die Deckenlampe aus* erzeugt werden. Wurde eine solche Regel erlernt, kann sie danach von der Anwendung verwendet werden, um bspw. das Licht im Raum fernsehgerecht zu gestalten.

Das Problem solcher lernender Anwendungen ist zu unterscheiden, wann Aktionen nur zufällig zusammen auftreten und wann sie so häufig sind, dass sie mit hoher Wahrscheinlichkeit eine Vorliebe des Bewohners sind. In der explorativen Analyse statischer Daten wird dazu Frequent Pattern Mining (FPM) [HCXY07] eingesetzt, um häufige Muster zu finden. Dessen Algorithmen sind jedoch nicht auf die Verarbeitung potentiell unendlicher Sensordaten ausgelegt, sodass die Umsetzung meist funktionsspezifisch in monolithischen Anwendungen erfolgt. Entsprechend werden sie separat implementiert und sind daher selten wiederverwendbar und aufwändig anpassbar. Des Weiteren werden gleiche Algorithmen häufig auch parallel ausgeführt, selbst wenn sie auf denselben Sensordaten arbeiten, was sich negativ auf Rechenzeit und Speicherverbrauch auswirkt.

Für eine wiederverwendbare, flexible und universell einsetzbare Sensordatenfusion wurden Datenstrommanagementsysteme (DSMS) als Technologie entwickelt, die Rechenzeit und Speicherverbrauch durch Optimierungstechniken, wie Restrukturierung oder Mehrfachverwendung von Verarbeitungsschritten verringern können. Daraus ergibt sich die Annahme, dass selbige Eigenschaften auch für FPM verwendet werden können. In dieser Arbeit stellen wir daher einen Ansatz vor, der FPM in ein DSMS integriert. Neben den Optimierungstechniken oder Wiederverwendbarkeit, können zusätzlich auch Funktionen wie Scheduling oder die Anfrageschnittstelle des DSMS ausgenutzt werden. Diese stellt eine höhere Ebene für die Anwendungsentwicklung bereit, welches wie bei Datenbankmanagementsystemen (DBMS) die Entwicklungszeit verringert (vgl. [EN06, GS02]). Des Weiteren speichert ein DSMS keine Rohdaten ab, wie es alternativ ein DBMS machen würde. Handelt es sich dabei um detaillierte personenbezogene Daten, wirkt sich dies positiv auf Datenschutz und Akzeptanz der Benutzer aus.

Die Verarbeitung von potentiell unendlichen Datenströmen bietet jedoch bestimmte Herausforderungen, die bei der Integration berücksichtigt werden müssen. Des Weiteren muss auch der zeitliche Zusammenhang der zu lernenden Aktivitäten beachtet werden. Obwohl es viele Lösungen gibt, die auf Datenströmen ausgelegt sind (vgl. [GHP+03, LL09, KRS11] u.a.), verlangen diese eine Anpassung des FPM, z.B. durch approximative Zusammenfassung von Daten. Ferner wurden einige Algorithmen zwar zur Evaluation in ein DSMS integriert, jedoch wurden bisher keine Optimierungsmöglichkeiten betrachtet, die durch die Integration in ein DSMS ermöglicht werden. Unser Beitrag ist daher wie folgt:

- Wir zeigen, wie FPM mit Hilfe des Intervall-Ansatzes [KS09] in ein DSMS integriert werden kann. Obwohl dies auf den ersten Blick eine Einschränkung bedeutet – nicht mehr alle Ereignisse – stellt es sich doch als sinnvoller dar, da typischerweise in diesem Szenario nur zeitlich korrelierende Ereignisse relevant sind. Der Ansatz lässt sich auch auf andere intervallbasierte Systeme übertragen.
- Wir definieren eine formale Semantik für FPM auf Basis einer relationalen Algebra für Datenströme. Die klare Semantik bietet eine deterministische und nachvollziehbare Berechnung von häufigen Mustern.
- Auf Basis der formalen Semantik evaluieren wir Optimierungsmöglichkeiten, indem ein FPM-Algorithmus als Algebraoperator in Zusammenhang mit existierenden Operatoren wie Selektion und Projektion betrachtet wird.

- Durch die Kapselung des konkreten Algorithmus zum FPM sind die vorgestellten Konzepte nicht auf einen bestimmten Algorithmus beschränkt. Es können bereits evaluierte und bewährte Verfahren auch aus Datenströmen eingesetzt werden.

Die restliche Arbeit gliedert sich wie folgt. Abschnitt 2 führt zunächst das verwendete Datenstrommodell anhand eines Beispiels ein und erläutert notwendige Definitionen und Annahmen. Darauf aufbauend zeigt Abschnitt 3, wie FPM in ein DSMS integriert werden kann und bietet dazu eine formale Semantik. Die Semantik wird in Abschnitt 4 verwendet, um algebraische Optimierungsmöglichkeiten aufzuzeigen. Integration als auch dessen Optimierung werden in Abschnitt 5 evaluiert. Während Abschnitt 6 verwandte Arbeiten betrachtet, fasst Abschnitt 7 die Arbeit abschließend zusammen und gibt einen Ausblick.

2 Motivation und Datenstrommodell

In einem Smart Home werden meist regelbasierte Systeme eingesetzt, die anhand erkannter Sensormesswerte bestimmte Aktorik ausführen. Ein Problem hierbei stellt die Definition der Regeln dar, die typischerweise von Benutzern vorgegeben werden müssen. Häufig können oder wollen sie sich jedoch nicht mit der Regelerstellung befassen oder wissen welche Regeln sinnvoll sind. Daher verwenden Hersteller meist eine Reihe von einfachen Standardregeln. Mit Hilfe von Lernverfahren können jedoch auch individuelle Regeln abgeleitet werden, indem regelmäßige Aktionen eines Benutzers erkannt werden. Abbildung 1 stellt mehrere solcher Aktionen dar. Um eine Regel abzuleiten, werden zunächst

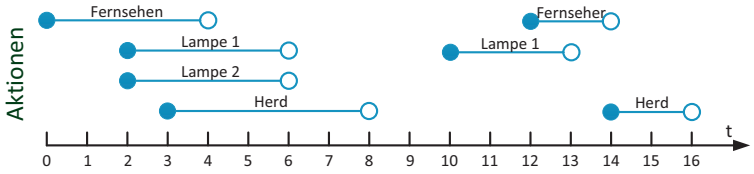


Abbildung 1: Beispiel der Gültigkeit von zusammen auftretenden Aktionen

häufig zusammen auftretende Aktionen betrachtet. Dazu wird der jeweilige Verwendungszeitraum als Gültigkeitsintervall gesehen. Beim *Fernseher* bspw. ist dies vom Einschaltzeitpunkt $t = 0$ bis Ausschaltzeitpunkt $t = 4$. Man sieht dann z.B., dass *Lampe 1*, *Lampe 2* und der *Herd* gleichzeitig mit dem *Fernseher* verwendet wurden. Damit liegt auch ein Muster *Fernseher, Lampe 1, Lampe 2, Herd* vor. Dieses tritt im Gegensatz zu dessen Teilmuster *Fernseher, Lampe 1* nur einmalig auf. Da das Teilmuster zweimal auftritt, ist es hier häufig und kann als Basis für eine Regel dienen, die bspw. *Fernseher* $\Rightarrow$ *Lampe 1* lautet, um automatisch *Lampe 1* einzuschalten, wenn die Aktion *Fernseher* auftritt.

Anhand des Beispiels aus Abbildung 1 kann bereits der intervallbasierte Charakter erkannt werden, auf dem das hier vorgestellte Prinzip beruht. Entsprechend bietet sich ein Datenstrommodell an, das auch auf Gültigkeitsintervalle basiert. Insbesondere eignet sich der Intervall-Ansatz [KS09], bei dem der Datenstrom durch Fenster in endliche Teile partitio-

niert wird. Zum einen definiert der Ansatz bereits Gültigkeitsintervalle pro Datentupel, so dass dies genau einer Aktion und dessen Verwendungszeitraum entspricht. Zum anderen bietet der Ansatz eine formale Semantik, die u.a. deterministische Ergebnisse und Optimierungen ermöglicht. Dazu wird der Datenstrom in mehrere Schnappschüsse unterteilt, wobei jeder Schnappschuss mit dem Zustand eines DBMS vergleichbar ist. Dadurch bietet der Ansatz äquivalente Operatoren der relationalen Algebra [DGK82] wie Selektion, Projektion oder Aggregation und auch dessen algebraische Optimierungsmöglichkeiten. Die Semantik dazu basiert auf der Definition eines logischen Datenstroms (vgl. [KS09]). Sei dazu $\mathbb{S}^l$ die Menge aller logischen Datenströme und $\mathbb{S}^l_{\mathcal{A}} \subseteq \mathbb{S}^l$ die Menge aller logischen Datenströme vom Schema $\mathcal{A}$. Dann ist ein *logischer Datenstrom* $S^l \in \mathbb{S}^l_{\mathcal{A}}$ wie folgt definiert: $S^l := \{(e,t,n) | e \in \Omega_{\mathcal{A}} \wedge t \in T \wedge n \in \mathbb{N} \wedge n > 0\}$ Dabei ist $\Omega_{\mathcal{A}}$ die Menge aller Nutzdatentupel vom Schema $\mathcal{A}$ und T die Menge aller Zeitstempel, sodass $\Omega_{\mathcal{A}} \times T \times \mathbb{N}$ ein Tupel im Datenstrom ist und ein Nutzdatentupel a mit einem Schema A zum Zeitpunkt t genau n mal auftritt. Abbildung 2 veranschaulicht diese Ansichtsweise für das zuvor genannte Beispiel. Betrachtet man zu jedem Zeitpunkt alle gültigen Aktionen, erhält man jeweils einen

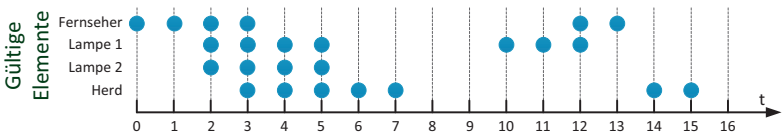


Abbildung 2: Beispiel der Gültigkeit von Tupeln eines logischen Datenstroms

Schnappschuss. So ergeben sich aus dem Beispiel in Abbildung 2 die in Tabelle 1 dargestellten Schnappschüsse. Obwohl diese logische Darstellung die erforderliche Semantik

T	Gültige Aktionen
0	{*Fernseher*}
1	{*Fernseher*}
2	{*Fernseher*, *Lampe* 1, *Lampe* 2}
3	{*Fernseher*, *Lampe* 1, *Lampe* 2, *Herd*}
4	{*Lampe* 1, *Lampe* 2, *Herd*}
5	{*Lampe* 1, *Lampe* 2, *Herd*}
6	{*Herd*}
7	{*Herd*}

T	Gültige Aktionen
8	{}
9	{}
10	{*Lampe* 1}
11	{*Lampe* 1}
12	{*Fernseher*, *Lampe* 1}
13	{*Fernseher*}
14	{*Herd*}
15	{*Herd*}

Tabelle 1: Gültige Schnappschüsse pro Zeitpunkt für den logischen Datenstrom

und dadurch auch eine Optimierung ermöglicht, entspricht diese Repräsentation nicht dem in Abbildung 1 gezeigten Intervallen und müsste demnach umgewandelt werden. Jedoch wäre eine Umwandlung in einen logischen Datenstrom aufwändig und zum anderen ist die Repräsentation einer Aktion ineffizient, wenn diese über mehrere zusammenhängende Zeitpunkte gültig ist, da es bspw. für *Lampe 2* vier statt nur eine Instanz geben müsste. Daher wird die logische Darstellung im Wesentlichen für die Semantik und darauf aufbauend für die Optimierung verwendet. Die eigentliche Verarbeitung erfolgt, wie auch im Intervall-Ansatz, durch eine physische Repräsentation. Diese fasst mehrere zusammenhängende Gültigkeitszeitpunkte zu einem Gültigkeitsintervall zusammen. Dieses Intervall

beschreibt jeweils, dass ein Tupel von einem Startzeitstempel $t_s \in T$ bis zu einem Endzeitstempel $t_e \in T$ gültig ist. Sei $\mathbb{S}^p$ die Menge aller physischen Datenströme, dann ist $\mathbb{S}^p_{\mathcal{A}} \subseteq \mathbb{S}^p$ die Menge aller physischen Datenströme mit dem Schema $\mathcal{A}$ und ein *physischer Datenstrom* $S^p \in \mathbb{S}^p_{\mathcal{A}}$ (vgl. [KS09]) definiert durch:$S^p := \{(e, [t_s, t_e)|a \in \Omega_{\mathcal{A}} \wedge t_s, t_e \in T\}$. Im Gegensatz zum logischen Datenstrom wird hier die Reihenfolge aller Tupel e betrachtet, indem sie monoton aufsteigend anhand der Startzeitstempel t_s sortiert sind und zwei Aktionen sind gleichzeitig gültig, wenn sich ihre Zeitintervalle überschneiden.

3 Intervallbasiertes Frequent Pattern Mining

Für die Integration von FPM wir ein neuer Operator eingeführt, der analog zum Intervall-Ansatz sowohl auf logischen als auch auf physischen Datenströmen definiert wird. Dazu wird zunächst die Semantik des Operators als eine Abbildung zwischen logischen Datenströmen beschrieben. Existierende Verfahren zum FPM werden meist zur Warenkorbanalyse eingesetzt, indem bspw. zusammen gekaufte Produkte als eine sogenannte Transaktion betrachtet werden, zwischen denen dann Gemeinsamkeiten gesucht werden. Die Idee hinter FPM auf intervallbasierten Aktionen ist es, gleichzeitig gültige Aktionen genau als eine solche Transaktion aufzufassen. Dadurch können vorhandene Algorithmen zur Warenkorbanalyse wie bspw. Apriori [AS+94] oder FP-Growth [HPY00] wiederverwendet werden und Algorithmen müssen nicht für Datenströme angepasst werden. Diese Algorithmen erwarten jedoch eine endliche Anzahl an Transaktionen, obwohl der Datenstrom unendlich viele Transaktionen liefern kann. Analog zu einem gleitenden Fenster (vgl. [KS09]), wie er im Intervall-Ansatz zum Bestimmten der Gültigkeit einer Aktion verwendet wird, wird hier ein Fenster bestimmt, in dem zusammenhängend gelernt wird. Anhand eines vorgegebenen Zeitraums oder einer Anzahl, werden veraltete Transaktionen aus dem Fenster entfernt, sobald neue Transaktionen hinzukommen. Dieses Fenster erlaubt es außerdem, dass sich das Lernen anpassen kann, indem es veraltete Muster verlernt. Dadurch kann mit Concept Drifts (vgl. [GZK05]) umgegangen werden, die bspw. auftreten, wenn der Bewohner sein Verhalten ändert oder Geräte und damit Aktionen nicht mehr existieren. In Tabelle 2 links ist beispielhaft ein Ausschnitt von $w = 5$ Transaktionen zum Zeitpunkt $t = 6$ dargestellt. Hat man einen Zeitfortschritt zu $t = 7$, dann verschiebt sich

Transaktionen (t=6)
{Fernseher, Lampe 1, Lampe 2}
{Fernseher, Lampe 1, Lampe 2, Herd}
{Lampe 1, Lampe 2, Herd}
{Lampe 1, Lampe 2, Herd}
{Herd}

Transaktionen (t=7)
{Fernseher, Lampe 1, Lampe 2, Herd}
{Lampe 1, Lampe 2, Herd}
{Lampe 1, Lampe 2, Herd}
{Herd}
{Herd}

Tabelle 2: Beispiel für zwei Ausschnitte von Transaktionsmengen

der Ausschnitt um eine Transaktion, wie Tabelle 2 rechts zeigt. Da es sich um eine endliche Menge von Transaktionen handelt, können existierende Algorithmen zum FPM auf statischen Daten übernommen werden. Der hier vorgestellte Ansatz ist dadurch nicht auf

einen bestimmten Algorithmus zugeschnitten. Jedoch muss die Semantik der Algorithmen äquivalent sein, indem sie dasselbe Ergebnis liefern würden. Dabei ist eine Kombination von Aktionen ein häufiges Muster, wenn es einen Support s erfüllt, indem die Kombination mindestens s mal im Ausschnitt vorkommt. Des Weiteren muss auch jede Teilmenge des häufigen Musters selbst ein häufiges Muster sein [HPY00]. Ist X die Menge aller möglichen Aktionen, so ergibt die Potenzmenge $\wp(X)$ die Menge aller möglichen Muster (bis auf die leere Menge), die ein FPM als mögliche Kandidaten betrachtet. Von dieser Kandidatenmenge sind dann nur solche häufig, die mindestens s mal in der betrachteten Transaktionsmenge vorkommen. Hierbei ist anzumerken, dass verschiedene Algorithmen dieselben Ergebnisse liefern, aber gerade das Erzeugen der Kandidatenmenge, das Zählen des Support und das Prüfen gegenüber dem Datenbestand den unterschied der Algorithmen ausmacht. Für den Ausschnitt $t = 6$ aus Tabelle 2 ergibt sich die in Tabelle 3 gezeigte Kandidatenmenge. Betrachtet man noch den Support s, so dass bspw. ein minimaler Sup-

mögliches Muster	Support s
{Fernseher}	2
{Herd}	4
{Lampe 1}	4
{Lampe 2}	4
{Fernseher, Herd}	1
{Fernseher, Lampe 1}	2
{Fernseher, Lampe 2}	2
{Herd, Lampe 1}	3

mögliches Muster	Support s
{Herd, Lampe 2}	3
{Lampe 1, Lampe 2}	4
{Fernseher, Herd, Lampe 1}	1
{Fernseher, Herd, Lampe 2}	1
{Fernseher, Lampe 1, Lampe 2}	2
{Herd, Lampe 1, Lampe 2}	3
{Fernseher, Herd, Lampe 1, Lampe 2}	1

Tabelle 3: Kandidaten für $t = 6$

port von $s = 2$ notwendig ist, ergeben sich die in Tabelle 4 gezeigten häufigen Muster. Analog dazu werden auch zum Zeitpunkt $t = 7$ entsprechende Kandidaten berechnet. Um

häufiges Muster ($t = 6$)	s
{Fernseher}	2
{Herd}	4
{Lampe 1}	4
{Lampe 2}	4
{Fernseher, Lampe 1}	2
{Fernseher, Lampe 2}	2
{Herd, Lampe 1}	3
{Herd, Lampe 2}	3
{Lampe 1, Lampe 2}	4
{Fernseher, Lampe 1, Lampe 2}	2
{Herd, Lampe 1, Lampe 2}	3

häufiges Muster ($t = 7$)	s
{Herd}	5
{Lampe 1}	3
{Lampe 2}	3
{Herd, Lampe 1}	3
{Herd, Lampe 2}	3
{Lampe 1, Lampe 2}	3
{Herd, Lampe 1, Lampe 2}	3

Tabelle 4: Häufige Muster für minimalen Support von $s = 2$

die Semantik eines solchen Verfahrens zum FPM zu beschreiben, werden die beiden zuvor genannten Schritte formalisiert. Dazu wird auf Grundlage logischer Datenströme eine Abbildung definiert, die den betrachteten Ausschnitt an Transaktionen bestimmt. Darauf

aufbauend wird anschließend eine weitere Abbildung definiert, welche die Erzeugung von häufigen Mustern nach obiger Semantik formalisiert.

Sei dazu $w \in \mathbb{N}$ mit $w > 0$ ein Parameter und $S^l \in \mathbb{S}^l$ ein logischer Datenstrom mit entsprechenden Tupeln $(e, \hat{t}, n) \in S^l$. Dann definieren wir eine Abbildung $\tau_t^w : \mathbb{S}^l \to \mathbb{S}^l$, die zu einem Zeitpunkt t bis zu w Transaktionen aus S^l entnimmt:

$$\tau_t^w(S^l) := \{(e, \hat{t}, n) \in S^l | max(0, t - w + 1) \leq \hat{t} \leq t\} \tag{1}$$

Wie erwähnt, ist die so erzeugte Transaktionsmenge im Gegensatz zu S^l endlich, so dass im Anschluss daran ein entsprechendes FPM ausgeführt werden kann. Obwohl hierzu ein konkreter Algorithmus, wie Apriori oder FP-Growth verwendet wird, abstrahieren wir das konkrete Verfahren. Im Wesentlichen werden alle vorhandenen Ausprägungen, die ein Tupel (e, t, n) für e annehmen kann, betrachtet. Aus dieser Menge wird dann eine Potenzmenge erzeugt, die der Kandidatenmenge entspricht. Die daraus entnommenen häufigen Muster ist letztendlich eine Teilmenge der Potenz- bzw. Kandidatenmenge. Dies bedeutet, dass FPM als eine Abbildung $\phi_t : \mathbb{S}^l \to \mathbb{S}^l$ gesehen werden kann, die zu einem Zeitpunkt $t \in T$ eine Menge von häufigen Mustern berechnet und wie folgt definiert werden kann:

$$\phi_t(S^l) := \{(m, t, 1) | m \subseteq \wp(X) \wedge X := \{e | (e, t, n) \in S^l\} \tag{2}$$

Hierbei ist zu erkennen, dass m hier im Vergleich zu τ_t^w selbst eine Menge von Tupeln ist. Der Unterschied liegt darin, dass das Nutzdatentupel e ein festes Schema hat, wie z.B. der Name der Aktion. Das Problem bei häufigen Mustern ist jedoch, dass es kein festes Schema gibt, da die Muster unterschiedliche Längen haben. Um dennoch eine Verarbeitung mit festem Schema zu erlauben, werden die häufigen Muster zu einem Tupel verschachtelt. Gibt es demnach mehrere Nutzdatentupel e_i, die jeweils ein eigenes Schema $\mathcal{A}_i$ haben, sodass $e_i \in \Omega_{\mathcal{A}_i}$ mit $1 \leq i \leq k$, dann können diese zu einer Menge $\{e_1, ..., e_k\}$ mit Schema $\mathcal{A}$ zusammengefasst werden. Dabei ist $k \in \mathbb{N}$ die Länge eines häufigen Musters und kann somit variieren. Für den Spezialfall $k = 1$ kann vereinfacht $\Omega_{\mathcal{A}} = \{\Omega_{\mathcal{A}_1}\}$ angenommen werden. Ein Tupel hat daher statt der Form $\Omega_{\mathcal{A}} \times T \times \mathbb{N}$ die Form $\{\Omega_{\mathcal{A}}\} \times T \times \mathbb{N}$, sodass man ein Datenmodell erhält, welches der Non-First-Normal-Form (NF^2) [SP82] ähnelt. Obwohl auch hier die Tupel wie bei der NF^2 verschachtelt werden (vgl. *nest*-Operation), gelten hier jedoch alle algebraischen Operationen auf das gesamte Tupel und nicht nur jeweils auf die verschachtelten Teile. Eine Selektion bspw. würde bei der NF^2 nur verschachtelte Teile entfernen und das hier vorgestellte Modell hingegen das gesamte Tupel verwerfen, wenn nur ein Teil nicht dem Prädikat genügt. Die Unterscheidung ist zum einen dadurch motiviert, dass ein häufiges Muster nur als Gesamtes seine Aussage behält und zum anderen ist jede Teilmenge eines häufigen Musters selbst ein häufiges Muster, sodass ein Entfernen eines Teils zu einem vorhandenen Muster führen würde und damit redundant vorhanden wäre. Anzumerken ist, dass ϕ_t auch für potenziell unendliche Datenströme definiert ist, da t lediglich den Zeitstempel setzt, aber keinen Ausschnitt wählt. Ferner kann hier auch ein FPM-Algorithmus eingesetzt werden, der bspw. inkrementell berechnet werden kann und für Datenströme ausgelegt ist. Für existierende Algorithmen, die auf endliche Datensätze ausgelegt sind, sollte ϕ_t jedoch stets in Verbindung mit τ_t^w verwendet werden. Für ein intervallbasiertes FPM ergibt sich daher der eigentliche logische FPM-Operator ρ_t^w für einen logischen Datenstrom S^l durch die Komposition $\phi_t \circ \tau_t^w$ wie

folgt $\rho_t^w(S^l) := \phi_t(\tau_t^w(S^l))$. Hierbei sei auf den Zusammenhang von s und w hingewiesen, indem ein Wert maximal w-mal auftreten kann und maximal s mal auftreten muss. Folglich muss $s \leq w$ gelten, damit ein FPM hier noch Ergebnisse liefern kann.

Da die logische Repräsentation wie erwähnt ineffizient ist, wird die Implementierung anhand physischer Datenströme vorgenommen. Zusätzlich kann dadurch derselbe logische Operator während der Transformation durch verschiedene physische Implementierungen mit konkreten Algorithmen wie Apriori oder FP-Growth ausgetauscht werden. Um hier jedoch von einem konkreten Algorithmus zu abstrahieren, stellen wir einen allgemeinen physischen Operator vor. Die physische Repräsentation von ρ_t^w kann dabei durch den in Algorithmus 1 gezeigten Pseudocode umgesetzt werden.

Algorithmus 1 FPM-Operator

Require: Physischer Datenstrom S_{in}
Ensure: Physischer Datenstrom S_{out}
1: $min_{t_s} \in T \cup \{\bot\}$;$min_{t_s} \leftarrow \bot$
2: Sei *puffer* eine Prioritätsqueue für Elemente $(e, [t_s, t_e))$ mit Ordnungsrelation $\leq_{t_s}$
3: Sei *transactions* eine FIFO-Liste für Mengen der Form $\{(e, [t_s, t_e))\}$
4: Sei w die Anzahl der Transaktionen
5: Sei *fpm* ein Algorithmus zum FPM
6: **for** $s := (e, [t_s, t_e)) \hookleftarrow S_{in_j}$ **do**
7: **if** $t_s > min_{t_s}$ **then**
8: $ta \leftarrow \emptyset$
9: **while not** *puffer.isEmpty()* **do**
10: $\hat{s} := (\hat{e}, [\hat{t_s}, \hat{t_e})) \leftarrow$ *puffer.peek()*
11: **if** $\hat{t_s} \leq t_s$ **then**
12: $\hat{s} \leftarrow$ *puffer.poll()*
13: *ta.insert*$(\hat{s})$
14: **else**
15: **break**
16: **end if**
17: **end while**
18: **if** *transactions.size()* == w **then**
19: *transactions.poll()*
20: **end if**
21: *transactions.offer(ta)*
22: *frequentsets = fpm(transactions)*
23: **for all** *frequentset* $\hookleftarrow$ *frequentsets* **do**
24: $\widetilde{s} :=$ (*frequentset*, $[min_{t_s}, t_s)$)
25: $\widetilde{s} \hookrightarrow S_{out}$
26: **end for**
27: **end if**
28: $min_{t_s} = t_s$
29: *puffer.offer(s)*
30: **end for**

Der Algorithmus speichert alle eingehenden Tupel in einen *puffer*. Wenn bei einem Zeitfortschritt $t_s > min_{t_s}$ gilt, sind alle nötigen Tupel vorhanden, aus denen dann eine Trans-

aktion *ta* gebildet wird, die zu den vorhandenen Transaktionen *transactions* hinzugefügt wird. Gibt es w Transaktionen, dann wird die älteste Transaktion entfernt und setzt damit Abbildung τ_t^w um. Abbildung ϕ_t hingegen wird einerseits durch den austauschbaren Algorithmus *fpm* für das eigentliche FPM und andererseits durch setzen des Zeitintervalls $[min_{t_s}, t_s)$ umgesetzt. Hierbei sollte *fpm* eine Menge aus mehreren *frequentset* liefern, die wiederum Kombinationen aus den Eingangstupeln *e* sind. Jedes *frequentset* wird dem ausgehenden Datenstrom S_{out} übergeben. Der Algorithmus ist datengetrieben und wird durch *s* angestoßen. Jedoch wird eine Transaktion erst erzeugt, wenn zum Zeitpunkt t_s alle Tupel für den Ausschnitt $[min_{t_s}, t_s)$ bekannt sind. Entsprechend muss auf ein Tupel mit $t_s + 1$ gewartet werden, sodass blockiert wird. Kommen bspw. keine Tupel mehr, können Heartbeats bzw. Punctuations [TMSF03] jedoch eine vorzeitige Berechnung auslösen, indem sie den Zeitfortschritt angeben. Dazu ersetzt der Zeitstempel des Heartbeats t_h die entsprechenden t_s in Zeile 7-27.

4 Algebraische Optimierungen

Durch die semantische Beschreibung eines FPM auf Basis der relationalen Algebra entsteht die Möglichkeit für Optimierungen. Analog zu einem DBMS verfügt ein DSMS häufig auch über einen Anfrageoptimierer, der eine Anfrage in Form einen gerichteten Graphen aus Algebraoperatoren (dem Anfrageplan) entgegennimmt und auf Grundlage von Äquivalenz-Regeln, die zwischen verschiedenen Operatorkombinationen gelten, optimiert. So können bspw. zwei Operatoren getauscht oder unnötige Operatoren entfernt werden, ohne dass sich das Ergebnis dadurch ändert. Die gebräuchlichsten Optimierungen basieren dabei auf dem Tauschen mit Selektionen und Projektionen, da diese die Anzahl der Tupel bzw. Attribute verringert und im Vergleich zu anderen Operatoren wie Verbünde weniger rechenintensiv sind. Daher wird versucht, Selektionen möglichst nah zu den Quellen zu verschieben und mit Projektionen alle unnötigen Attribute zu entfernen. Im Folgenden betrachten wir daher Selektionen und Projektionen in Bezug auf FPM. Um zu zeigen, dass eine Selektion bzw. Projektion in Kombination mit einem FPM-Operator verlustfrei optimiert werden kann, muss die Äquivalenz gezeigt werden. Für die intervallbasierten Standardoperatoren zeigt [KS09] entsprechende Äquivalenzen, bei dem zwei Kombinationen aus Operatoren äquivalent sind, wenn dessen resultierenden Datenströme zu jedem Zeitpunkt dieselben Tupel enthalten. Im Folgenden untersuchen wir damit die Kombination von FPM und Selektion, sowie FPM und Projektion.

4.1 Optimierung mit Selektionen

Möchte ein Bewohner bspw. nicht, dass der *Fernseher* automatisch geschaltet wird, würde man entsprechende Muster mit *Fernseher* nach dem FPM entfernen. Es bietet sich an, dass eine solche Selektion nicht nach dem kostenintensiven FPM-Operator durchgeführt wird, sondern wenn möglich bereits vorher, damit durch die Selektion weniger Elemente verarbeitet werden müssen. Hierbei muss jedoch zum einen beachtet werden, dass der

verwendete FPM-Algorithmus Häufigkeiten nur absolut und nicht relativ zählen darf, da eine Selektion die Grundmenge verändert und zum anderen liegen vor und nach dem FPM verschiedene Schema bzw. Modelle vor, wie in Abschnitt 3 beschrieben wird. Vor einem FPM gilt das flache Modell, bei dem die Selektion für ein Prädikat p nach [KS09] durch $\sigma_p(S) := \{(e,t,n) \in S | p(e)\}$ definiert ist. Nach einem FPM liegen jedoch verschachtelte Tupel vor, auf denen das Prädikat für alle Teiltupel gelten muss. Eine verschachtelte Selektion ist daher wie folgt definiert: $\hat{\sigma}_p(S) := \{(\hat{e},\hat{t},\hat{n}) \in S | \forall \hat{a} \in \hat{e} : p(\hat{a})\}$. Der Unterschied zum NF^2 Modell, bei dem nur die Teiltupel und nicht das gesamte Tupel entfernt wird, ist wie folgt motiviert. Sei bspw. *{Fernseher, dunkel, Lampe}* ein häufiges Muster, aus dem eine Regel erzeugt wird, welche die *Lampe* anschaltet sobald es *dunkel* ist und der *Fernseher* angemacht wird. Möchte der Bewohner keine Regeln mit *Fernseher*, so kann das gesamte zugehörige häufige Muster *{Fernseher, dunkel, Lampe}* entfernt werden, da auf Grund der Eigenschaften eines häufigen Musters, das Teilmuster *{dunkel, Lampe}* ohnehin vorhanden ist und sonst Redundanz vorliegen würde. Liegt ein Datenstrom S vor, auf dem ein FPM und anschließend eine Selektion ausgeführt wird, gilt $\hat{\sigma}_p(\rho_t^w(S))$. Bei einer Optimierung würde die Selektion vor dem FPM ausgeführt werden, sodass $\rho_t^w(\sigma_p(S))$. Daraus folgt folgende Optimierungsregel[1]:$\hat{\sigma}_p(\rho_t^w(S)) = \rho_t^w(\sigma_p(S))$.

4.2 Optimierung mit Projektion

Bei der Optimierung mit einer Projektion π wird die Verwendung von Attributen betrachtet. Sowohl der Teil τ_t^w als auch der Teil ϕ_t der Abbildung ρ_t^w verwenden zwar nicht explizit Attribute, jedoch basiert das Zählen der Häufigkeiten im FPM-Algorithmus auf der Äquivalenz von Tupeln, die sich nach den verwendeten Attributen richtet. Zwei verschiedene Tupel *(Fernseher, an)* und *(Fernseher, aus)* würden so zu äquivalenten Tupeln *(Fernseher)*. Obwohl es durchaus möglich ist, nur einen Teil der Attribute (im Beispiel nur das erste Attribut) im FPM zu verwenden, bietet sich dieses nicht an. Zum einen würde das FPM dadurch selbst eine Projektion durchführen und die Funktion wäre dann redundant und nicht mehr gekapselt. Zum anderen wäre die Frage, wie nicht verwendete Attribute für das häufige Muster zusammengeführt werden. Im Beispiel würde man zwar zweimal *Fernseher* zählen, jedoch wäre die Frage ob *an* oder *aus* verwendet werden soll. Ist $\hat{\pi}$ analog zur Selektion eine Projektion auf verschachtelten Tupeln und π das entsprechende Gegenstück für flache Tupel, dann folgt die Regel $\hat{\pi}_p(\rho_t^w(S)) \neq \rho_t^w(\pi(S)$. Somit ist eine Verschiebung einer Projektion bei Verwendung eines FPM-Operators nicht möglich[2].

5 Evaluation

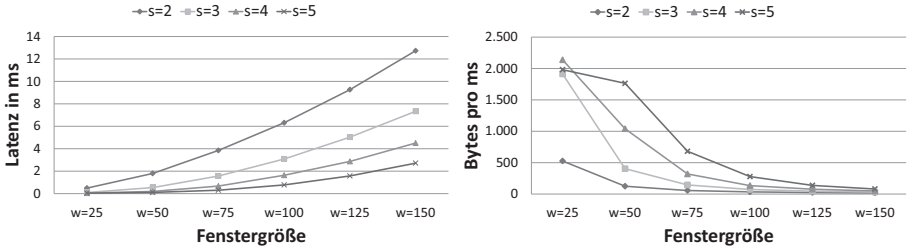
Für die Evaluation wurde der physische FPM-Operator in das DSMS Odysseus [AGG+12] integriert. Als Testdatensatz dient *T10I4D100K* [AS+94], der sehr häufig zum Vergleich

[1] Ein formaler Beweis ist vorhanden, kann aber aus Platzgründen nicht dargestellt werden
[2] Ein formaler Beweis wurde aus Platzgründen entfernt

von FPM-Algorithmen verwendet wird. Die insgesamt $D = 100.000$ Transaktionen haben durchschnittlich $T = 10$ Werte und haben häufige Muster mit einer durchschnittlichen Länge von $I = 4$. Die insgesamt 1.010.229 Werte werden nacheinander durch Odysseus eingelesen. Dabei haben die Werte einer Transaktion dasselbe Gültigkeitsintervall erhalten, sodass bspw. die Werte der ersten Transaktion $[0, 50)$ und die der zweiten Transaktion $[100, 150)$ als Gültigkeit erhalten haben. Für die Evaluation wurde der FP-Growth [HPY00] implementiert, kann prinzipiell jedoch gegen andere Verfahren ausgetauscht werden. Der verwendete Rechner verfügt über eine Intel Core i5 CPU mit zwei Kernen mit je 2.50 GHz und 16 GB Arbeitsspeicher unter einem 64-Bit Windows 7. Es wurden Durchsatz und Latenz gemessen, da diese die maßgeblichen Metriken in der Datenstromverarbeitung sind. Bei der Latenz, die Dauer zwischen Ein- und Austritt eines Wertes angibt, konnte der Wert nicht direkt gemessen werden, weil ein erzeugtes häufiges Muster nur indirekt von eingehenden Werten abhängt. Analog zur Latenzberechnung bei einer Aggregation wurde die Latenz von dem Wert gemessen, der als letztes zur Berechnung des häufigen Musters beigetragen hat. Die Evaluation selbst behandelt in Abschnitt 5.1 zunächst die Integration des FPM und anschließend in Abschnitt 5.2 die Optimierung.

5.1 Integration von FPM

Bei der Integration wurden die Machbarkeit und die Skalierbarkeit für einen potenziell unendlichen Datenstrom evaluiert. Dazu zeigen die Latenzen in Abbildung 3a, dass einerseits ein kleinerer Support s und andererseits ein größeres Fenster w zu höheren Latenzen führt. In beiden Fällen müssen jeweils mehr Daten vorgehalten werden, um potenziel-



(a) Latenz (b) Durchsatz

Abbildung 3: Evaluation bei verschiedenen Fenstergrößen und Support

le häufige Muster zu erkennen. Da bei n möglichen Ausprägungen bis zu 2^n potenzielle Muster (vgl. Potenzmenge in Definition 2) entstehen können, wächst auch hier der Aufwand nicht linear mit zunehmender Größe des Baums. Ein selbiges Verhalten spiegelt sich auch im Durchsatz wider, der in Abbildung 3b gezeigt wird. Auch hier sinkt der Durchsatz bei kleinerem Support s bzw. größerem Fenster w. Hierbei ist zu erwähnen, dass der maximale Durchsatz systembedingt bei etwa 2.200 Bytes pro ms lag. Betrachtet man die Latenz über die Zeit, wie sie in Abbildung 4 dargestellt ist, erkennt man für verschiedene Support s, dass die Latenz nicht ansteigt, sondern um einen konstanten Wert berechnungs-

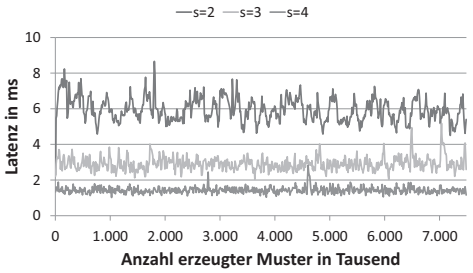
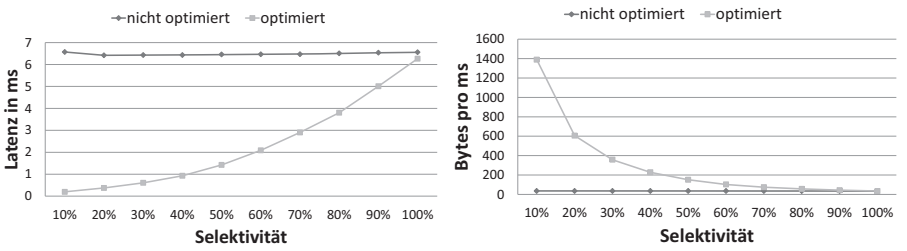


Abbildung 4: Durchschnittliche Latenz über die Zeit für $w = 100$

bedingt schwankt. Hieraus lässt sich erkennen, dass die Integration dieses Verfahrens auch für potenziell unendliche Datenströme skaliert.

5.2 Optimierung von FPM

Um den Mehrwert der Optimierung zu zeigen, wird dieser mit dem unoptimierten Fall verglichen. Maßgeblich bei der Selektion ist die Selektivität des Prädikats. Da die Werte in *T10I4D100K* gleich verteilt zwischen 0 und 999 liegen (vgl. [AS+94]), wurde bspw. bei eine Selektivität von 10% das Prädikat „< 100“ verwendet, um die Selektivität zu simulieren. Abbildung 5a zeigt die Latenzen, die dem erwarteten Verhalten entsprechen. Die



(a) Latenz (b) Durchsatz

Abbildung 5: Vergleich bei Optimierung mit Selektion

Latenzen sind im nicht optimierten Fall konstant, da sowohl FPM als auch die nachfolgende Selektion unabhängig von der Selektivität für jedes Element durchgeführt werden muss. Der optimierte Fall hingegen ist stark von der Selektivität abhängig, da am Anfang (je nach Selektivität) bereits viele Daten verworfen werden können und dadurch die zwischengespeicherten Daten bzw. Kandidaten für das FPM kleiner gehalten werden. Dies wird verdeutlicht, indem bei einer Selektivität von 100% fast keine Optimierung mehr zu erkennen ist. Der dennoch vorhandene Unterschied bei 100% liegt daran, dass die Selektion $\hat{\sigma}$ im nicht optimierten Fall bei Werten von 0 bis 999 je nach Support bis zu 2^{1000} Muster prüfen muss. Im optimierten Fall sind es hingegen alle 1.010.229 Werte des Da-

tensatzes. Analog zur Latenz spiegelt auch der Durchsatz das Ergebnis wider, indem eine kleinere Selektivität im optimierten Fall zu einem höheren Durchsatz führt und sich dieser bei 100% dem unoptimierten Fall angleicht.

6 Verwandte Arbeiten

Die Berechnung häufiger Muster bzw. Frequent Pattern Mining (FPM) wird auch als Warenkorb- oder Assoziationsanalyse bezeichnet und ist ein Ansatz des Data Mining (vgl. [HKP11, WFH11] u.a.) und werden dabei i.d.R. zur explorativen Analyse von statischen Daten benutzt. Apriori [AS+94] ist einer der ersten Algorithmen zum FPM, auf dessen Basis anschließend Weitere entwickelt wurden. Da die Kandidatengenerierung von Apriori gerade bei kleinem Support exponentiell wächst, wurde FP-Growth [HPY00] entwickelt, der einen Präfix-Baum als Alternative einsetzt, auf denen dann wiederum neuere Verfahren basieren. Da klassische Data-Mining-Algorithmen für Datenbanken ausgelegt sind, gibt es weitere Anpassungen, die Besonderheiten für Datenströme berücksichtigen. Als Pendant für Datenströme hat sich das Data Stream Mining (vgl. [Gam07, Gam10, GZK05] u.a.) entwickelt, in denen auch FPM auf Datenströmen [JG06] betrachtet wird. Einige Ansätze wie [LL09, TMP08] beschäftigen sich dabei mit dem Zählen von Häufigkeiten (vgl.. [CH08]), da dies ein grundlegendes Problem potentiell unendlicher Datenströme ist. Als Alternative gibt es Fensteransätze, die den Datenstrom in endliche Teile partitionieren. FP-Stream [GHP+03] bspw. versucht eine Historie von Fenstern aufzubauen, um so auch eine (nachträgliche) explorative Analyse auf Datenströmen zu ermöglichen. Weitere Algorithmen, wie [CWYM04, LL09, KRS11] u.a., fokussieren jeweils spezielle Probleme bei Datenströmen, beschränken sich auf den eigentlichen Algorithmus. So haben sie zwar verschiedene Ansätze oder betrachten andere Arten von häufigen Mustern, jedoch wird nicht auf die Integration in ein bestehendes DSMS eingegangen. Unser Ansatz hingegen zeigt eine solche Integration, bei dem zusätzlich keine speziellen Anpassungen an Datenströme nötig sind und Algorithmen wie FP-Growth wiederverwendet werden können.

Des Weiteren bietet keine der vorhandenen Algorithmen eine formale Semantik, die auf einer vorhandenen Algebra beruht. DSMS wie Aurora [A+03], Borealis [A+05], STREAM [ABB+04], Odysseus [AGG+12] oder PIPES [KS09] sind i.A. auf einer Algebra aufgebaut. Dieser Ansatz beruht dabei auf der Intervall-Algebra nach [KS09], die u.a. in PIPES und Odysseus eingesetzt wird, da diese sich sehr gut für die Repräsentation von Gültigkeitsintervallen der Aktionen eignet. Einige Systeme, insbesondere Stream Mill Miner [T+11], integrieren zwar Data Mining und Datenstrommanagement, berücksichtigen dabei jedoch nur Klassifikation und Clustering. Des Weiteren beruhen die Umsetzungen auf einer benutzerdefinierten Aggregation und verfügen damit nicht über eine formale Semantik. Aus diesem Grund werden auch keine Optimierungen betrachtet. Dieser Ansatz betrachtet sowohl FPM im Kontext von Datenstrommanagement, beschreibt diesen formal und betrachtet gleichzeitig auch Optimierungen von Data Mining und vorhandenen Datenstrom-Operatoren. Neben der Optimierung der Verarbeitung, spiegelt sich ein integrierter Ansatz auch in der Anwendungsentwicklung wider, wie es [GS02] analog für die Integration von Data Mining in ein DBMS zeigt.

7 Zusammenfassung

In diesem Beitrag haben wir einen neuen Operator zum Frequent Pattern Mining (FPM) auf Datenströmen eingeführt. Dazu haben wir auf Basis der existierenden relationalen Algebra eine klare Semantik formuliert und eine mögliche Implementierung gezeigt. Durch die Verwendung des Intervall-Ansatzes ist es möglich, beliebige FPM-Algorithmen für statische Datenquellen in einem DSMS einzusetzen, ohne dass die Algorithmen für die Verarbeitung von Datenströmen angepasst werden müssen. Des Weiteren haben wir ein neues Konzept eingeführt, indem der FPM-Operator in Zusammenhang mit Projektion und Selektion betrachtet wird, um daraus Optimierungen zu ermöglichen. Entsprechend haben wir gezeigt, dass ein Vertauschen mit einer Projektion nicht möglich ist und dass das Vertauschen mit einer Selektion Latenzen senken und den Durchsatz steigern kann. Sowohl die Performanz der eigentlichen Integration als auch die Optimierung bei einer vorhandenen Selektion haben wir anschließend für verschiedene Parameterausprägungen evaluiert. Die Ergebnisse der Evaluation spiegelten dabei die zu erwarteten Effekte wider.

Obwohl Selektion und Projektion durch das Entfernen von Attributen und Tupeln das größte Optimierungspotenzial bietet, untersuchen wir ergänzend weitere Operatoren wie Kreuzprodukt oder Differenz, um damit die minimale Menge der Operatoren der relationalen Algebra abzudecken. Das Wiederverwenden eines FPM-Operators ist ein weiterer Punkt, der ebenfalls Optimierungspotenzial in DSMS bietet. Hierbei muss unter anderen die semantische Äquivalenz zweier FPM-Operatoren betrachtet werden, die eventuell auch durch ein Teilen des einen FPM-Operators oder durch zusätzliche Operatoren, wie z.B. eine Selektion, erreicht werden kann. Des Weiteren evaluieren wir die Umsetzbarkeit des Konzepts anhand einer prototypischen Fallstudie, indem Odysseus und der FPM-Operator als Teil einer Demonstration mit verschiedenen Sensoren verwendet werden.

Literatur

[A+03] D. J. Abadi et al. Aurora: a new model and architecture for data stream management. *VLDB Journal*, 12(2), 2003.

[A+05] D. Abadi et al. The Design of the Borealis Stream Processing Engine. In *CIDR*, 2005.

[ABB+04] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, R. Motwani, R. Motwani und U. Srivastava. STREAM: The Stanford Data Stream Management System, 2004.

[AGG+12] H.-Jürgen Appelrath, Dennis Geesen, Marco Grawunder, Timo Michelsen und Daniela Nicklas. Odysseus: a highly customizable framework for creating efficient event stream management systems. In *DEBS '12*. ACM, 2012.

[AS+94] R. Agrawal, R. Srikant et al. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, Jgg. 1215, 1994.

[CH08] Graham Cormode und Marios Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2), 2008.

[CWYM04] Y. Chi, H. Wang, P.S. Yu und R.R. Muntz. Moment: Maintaining closed frequent itemsets over a stream sliding window. In *ICDM'04*. IEEE, 2004.

[DAS01] A.K. Dey, G.D. Abowd und D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *HCI*, 16(2-4), 2001.

[DGK82] Umeshwar Dayal, Nathan Goodman und R.H. Katz. An extended relational algebra with control over duplicate elimination. In *ACM PODS*. ACM, 1982.

[EN06] Ramez Elmasri und Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 5th edition. Auflage, 2006.

[Gam07] Joao Gama. *Learning from Data Streams: Processing Techniques in Sensor Networks*. Springer Berlin Heidelberg, 2007.

[Gam10] Joao Gama. *Knowledge discovery from data streams*. Taylor and Francis, 2010.

[GHP+03] C. Giannella, J. Han, J. Pei, X. Yan und P. Yu. Mining Frequent Patterns in Data Streams at Multiple Time Granularities. *Next generation data mining*, 2003.

[GS02] Ingolf Geist und Kai-uwe Sattler. Towards data mining operators in database systems: Algebra and implementation. In *DBFusion*. Citeseer, 2002.

[GZK05] Mohamed Medhat Gaber, Arkady Zaslavsky und Shonali Krishnaswamy. Mining Data Streams : A Review. *ACM SIGMOD Record*, 34(2), 2005.

[HCXY07] J. Han, H. Cheng, D. Xin und X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1), 2007.

[HKP11] Jiawei Han, Micheline Kamber und Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 3. Auflage, 2011.

[HPY00] J. Han, J. Pei und Y. Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, Jgg. 29. ACM, 2000.

[JG06] Nan Jiang und Le Gruenwald. Research issues in data stream association rule mining. *SIGMOD Rec.*, 35(1), Marz 2006.

[KRS11] D. Klan, Th. Rohe und K.-U. Sattler. Quantitatives Frequent Pattern Mining in drahtlosen Sensornetzen. In *BTW Workshops*, March 2011.

[KS09] Jürgen Krämer und Bernhard Seeger. Semantics and implementation of continuous sliding window queries over data streams. *ACM TODS*, 34(1), April 2009.

[LL09] H.F. Li und S.Y. Lee. Mining frequent itemsets over data streams using efficient window sliding techniques. *Expert Systems with Applications*, 36(2), 2009.

[SP82] H.J. Schek und P. Pistor. Data structures for an integrated data base management and information retrieval system. In *VLDB*, 1982.

[T+11] Hetal Thakkar et al. SMM : a Data Stream Management System for Knowledge Discovery. In *ICDE*, 2011.

[TMP08] F. Tantono, N. Manerikar und T. Palpanas. Efficiently discovering recent frequent items in data streams. In *SSDM*. Springer, 2008.

[TMSF03] P.a. Tucker, D. Maier, T. Sheard und L. Fegaras. Exploiting punctuation semantics in continuous data streams. *IEEE TKDE*, 15(3), Mai 2003.

[WFH11] Ian H. Witten, Eibe Frank und Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 3. Auflage, 2011.

Towards an Energy-Proportional Storage System using a Cluster of Wimpy Nodes

Daniel Schall, Theo Härder

Databases and Information Systems Group
University of Kaiserslautern, Germany
{schall,haerder}@cs.uni-kl.de

Abstract: Previous DB research clearly concluded that the most energy-efficient configuration of a single-server DBMS is typically the highest performing one. This observation is certainly true if we focus in isolation on specific applications where the DBMS can steadily run in the peak-performance range. Because we noticed that typical DBMS activity levels—or its average system utilization—are much lower and that the energy use of single-server systems is far from being *energy proportional*, we came up with the hypothesis that better energy efficiency may be achieved by a cluster of nodes whose size is dynamically adjusted to the current workload demand. We will show that energy proportionality of a storage system can be approximated using a cluster of nodes, built of commodity hardware. To simulate data-intensive workloads, synthetic benchmarks submit read/write requests against a distributed DBMS (WattDB) and, in turn, its HDD- and SSD-based storage system, where time and energy use are captured by specific monitoring and measurement devices. The cluster dynamically adjusts its configuration such that energy consumption and performance are tuned to fit the current workload. For each benchmark setting, an optimal number of nodes is processing the queries in the most energy-efficient way, which does not necessarily correspond to the best performing configuration. The chosen workload is rather simple and primarily serves the purpose to deliver a proof of existence that energy proportionality can be approximated for certain kinds of query processing and, especially, for storage systems.

1 Introduction

The need for more energy efficiency in all areas of IT gained interest in research recently and ideas to increase the energy efficiency of stand-alone servers were proposed. Due to their narrow power range [BH07], i.e., the power spectrum between idle and full utilization, the goal of increased energy efficiency cannot be reached using todays hardware. Besides reducing the energy consumption of servers, other ideas like improving the cooling infrastructure and reducing its power consumption help reducing the energy footprint of data centers, although they do not decrease the energy consumption at the server level.

The original problem—reducing the energy consumption of installed servers—leads to a demand for energy-proportional hardware. *Energy proportionality* describes the ability of a system to reduce its power consumption to the actual workload, i.e., a system, that is utilized only 10% of its peak performance, must not consume more than 10% of its peak

power consumption. For specific applications, this goal can be approached by hardware-intrinsic properties, e.g., CPUs automatically entering sleep states or hard disks spinning down when idle. So far, automatically scaling systems down when idle—thus preventing high idle power consumption of todays servers—is the main focus of energy proportionality. Unfortunately, current hardware is not energy proportional for data-intensive applications as studies have shown (see [BH09]).

Several components such as CPUs are able to quickly change into sleep states, requiring less energy, when idle. Other components, especially the two main energy consumers of a DBMS, main memory and storage, exhibit bad energy characteristics. DRAM chips consume a constant amount of power—regardless of their use—and it is not possible to turn off unused memory chips in order to reduce energy consumption. Spinning down hard disks when idle conflicts with long transition times and results in slow query evaluation. For these reasons, such mechanisms are not very useful in case of DB applications where reference locality of data in large main-memory-resident DB buffers has to be preserved and low-latency accessibility of storage devices has to be guaranteed.

Todays servers are not energy efficient and approaches focusing on single machines cannot achieve energy proportional behavior either. This conclusion shifted the research focus from single-node approaches to clusters of servers, which appear more promising. Tsirogiannis et al. [THS10] observed in an extensive study based on empirical DBMS measurements that "*within a single node intended for use in scale-out (shared-nothing) architectures, the most energy-efficient configuration is typically the highest performing one*". In an independent study based on DBMS buffer management [OHS10], we came up at the same time with a similar conclusion concerning performance and energy efficiency of database systems and storage managers. Therefore, we want to improve energy efficiency of a DBMS by enabling the software side to explicitly power up/down resources as needed. Many clustered database systems exist, yet none has the ability to flexibly *scale up and down* in order to save energy.

We have shown in [SHK12] that real-world workloads usually do not stress database systems 24/7 with peak loads. Instead, the workloads alternate in patterns between high and near-idle utilization. But, database systems have to be tailored to the peak performance to satisfy incoming queries and potential users, i. e., customers. Therefore, database servers usually come with big DRAM buffers and a number of disks as external storage—both components that consume a lot of energy. The majority of these resources is only needed in rare time intervals to handle peak workloads. All other times, they lie waste, thereby substantially decreasing the overall energy efficiency of the server. During times of underutilization, overprovisioned components are not needed to satisfy the workload. By adjusting the database systems to the current workload's needs, i.e., making the system energy proportional, energy consumption could be lowered while still allowing the maximum possible performance.

This paper is structured as follows: Section 2 sketches WattDB and explains the power management algorithm. In Section 3, we describe the benchmarking model we used in this paper and explain our measurement setup for performance and energy consumption, before we discuss the results of our empirical benchmark runs in Section 4. Finally, we conclude and give an outlook in Section 5.

2 The WattDB Approach

For the reasons outlined above, we raised the hypothesis whether or not overall energy-efficiency optimization or energy-proportional system behavior could be better approached by a cluster of DB servers and redirected our work towards this research goal [GHP+10]. Hence, we want to achieve optimal DBMS energy efficiency—independent of its level of activity. For this purpose, we use a cluster of wimpy (lightweight) nodes, which can be powered on and off individually, allowing the cluster to scale. By dynamically adjusting the number of nodes in the cluster, the overall performance and energy consumption can be tailored to the current workload.

Our research project [SH11] focused on approaching energy-proportional runtime behavior for database management. So far, a commercially available DBMS does not exist, which can dynamically support powering up and down the nodes of a server cluster. Therefore, we have decided to build *WattDB* as a research prototype from scratch. The system is still under development and does not yet provide a full-fledged query execution engine. By providing limited querying capabilities and a primary-key index for tables, WattDB keeps queries close to the data. For this reason, it is not necessary to ship data pages to remote locations, which would burden the rather high network latency with each page read/write request.

2.1 Cluster Hardware

The cluster hardware consists of identical nodes, interconnected by a Gigabit-Ethernet switch. Each node is equipped with 2 GB of DRAM, an Intel Atom CPU D510 and (optionally) two storage disks. The hardware components are chosen to balance processing power and I/O bandwidth, making the nodes Amdahl-balanced [SBH+10]. All components running at 100% utilization result in a peak power consumption of about 30 Watts, hence they are ”wimpy”, compared to typical DB servers. This is the reason for choosing commodity hardware which uses much less energy compared to server-grade components. For example, main memory consumes ∼2.5 Watts per DIMM module, whereas ECC memory, typically used in servers, consumes ∼10 Watts per DIMM. Likewise, our HDDs need less power than high-performance drives, which makes them more energy efficient.

By choosing commodity hardware with limited data bandwidth, Ethernet wiring is sufficient for interconnecting the nodes. Currently, we have up to ten nodes running in the cluster. The nodes are connected to the same ethernet segment and can all communicate with each other. The total power consumption of the cluster can be broken down to roughly 20 Watts for the backplane switch, connecting the nodes; another 23 Watts for each active node, and 2.5 Watts for standby nodes. Fully utilized nodes (disks and CPU) consume about 30 Watts, mostly accounted to the CPU, as the power consumption of the disk drives is more or less steady. Replacing the magnetic storage disks with SSDs does not affect the power consumption of the storage nodes.

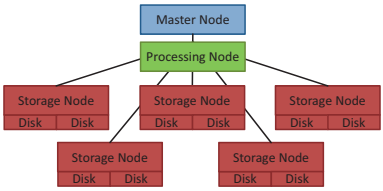


Figure 1: Overview of the cluster

2.2 Software

WattDB is a single-master DBMS, where one dedicated node accepts client connections, distributes incoming queries and keeps metadata for all cluster nodes. This node is also responsible for controlling the power consumption of the cluster by turning nodes on and off. The master is not processing queries, but distributing query plans and collecting results. This decision was made to prevent interferences between query processing and maintenance tasks on the master. Figure 1 sketches an exemplary cluster of seven nodes with the master on top. The remaining nodes can be classified as either storage nodes or processing nodes, whether they have disks attached or not. *Storage nodes* provide storage space to the cluster and act as page servers for other nodes. *Processing nodes* are executing queries by requesting pages from the storage devices, evaluating the content and writing them back. These nodes also hold the DB buffer to mitigate latency and limited bandwidth to the storage nodes. As mentioned, the query capabilities are still limited. The minimal configuration of the cluster requires the master node, at least one processing node, and also one storage node. It would be possible to allocate all three functions on a single physical node, but we decided to keep them separate for easier debugging and analysis.

2.3 Database Functionality

WattDB is based on a hybrid storage architecture. At the hardware level, the database can be considered as a shared-disk system; each processing node can access all storage disks remotely by connecting to the corresponding node and requesting pages. Characterized by its processing behavior, the shared-disk architecture is restricted to a logical shared-nothing DBMS. Each processing node has limited access to the storage layer and may only work on pages whose accessibility is previously defined by the master node. This restriction is enforced by the database software. By combining both approaches, *shared disk and shared nothing*, WattDB gains the flexibility to re-assign pages to processing nodes while avoiding synchronization cost that would come with a plain shared-disk architecture.

Database tables can span multiple processing nodes, each responsible for one of the table's partitions. Data is physically partitioned to the storage devices present, logical partitioning is currently not supported. Therefore, queries have to be evaluated on all processing nodes having partitions of the table in question. Re-distributing data blocks on storage devices is always possible and does not require any logical partitioning scheme, because it oper-

ates on physical data blocks. The mapping from logical to physical pages is done in the processing nodes, redistributing storage blocks only requires an update of the mapping.

2.4 Power Management

To approach energy-proportional processing behavior, WattDB is intended to dynamically scale the number of active nodes based on the current workload.

The master node is responsible for managing the cluster, switching nodes on and off, and redistributing the storage load. Each node monitors its disk and CPU usage and reports the readings periodically to the master to allow informed decisions based on the actual utilization of each node. The master is using the monitoring results for cluster orchestration—in particular, to estimate the overall cluster performance and to react to changing utilization.

Based on the monitoring data, the master node is running a kind of scheduling algorithm to adjust the number of nodes. This algorithm runs every minute and takes the past five minutes into account for calculating the IOPS. Listing 1 sketches this process in pseudo-code. First, all active storage devices in the cluster are examined and the current IOPS are compared to the threshold of max. allowed IOPS for this device (line 7 of the listing).[1]

Algorithm 1 Power-management pseudo code

```
1  ForEach(Storage storage in Cluster.Storages) {
2
3     If(!storage.PoweredOn) {
4        continue;
5     }
6
7     If(storage.IOPS > MAX_IOPS_PER_DISK) {
8        // Storage overloaded, acquire new storage and distribute data
9
10       Storage storageNew = PowerUpAnotherStorage();
11       Storage storageOld = GetStorageWithHighestLoad();
12
13       distributeBlocks(storageOld, storageNew);
14    }
15
16    If(storage.IOPS < MIN_IOPS_PER_DISK) {
17       // Storage underutilized, consolidate data to other active storages
18
19       consolidateStorage(storage);
20       storage.Suspend();
21    }
22 }
23 // Suspend unused nodes
24 ForEach(Node node in Cluster.Nodes) {
25    If(node.ActiveStorages == 0 &&
26       node.Partitions == 0 &&
27       !node.IsMaster) {
28       node.Suspend();
29    }
30 }
```

[1] The threshold is set to 90% of the peak IOPS for the drive, which was determined beforehand.

If the current utilization of a device exceeds the threshold, it is considered overloaded and the data is distributed to other storages devices. Not depicted in this listing is the selection of the distribution targets (line 13): The algorithm tries to move blocks to active, non-overloaded storage devices attached to the same node first, to minimize network traffic. In case these storage devices are already utilized too much, additional disks on the same node are powered up and used as a target, if possible. Lastly, when all the storage devices identified above are not sufficient to handle the load, other nodes are taken into account as well, and data blocks are shipped over the network to re-distribute the load. In case no other eligible nodes are found, additional storage nodes have to be powered up first.

After analyzing the overutilized storage disks and distributing their load, the algorithm now examines underutilized storage devices and tries to consolidate data blocks to other storage devices (line 16). This step performs the opposite work as sketched above and aims to reduce the number of storage disks, while still maintaining sufficiently high IOPS. Consolidating storage disks (line 19) follows a similar logic as before: First, disks on the same node are selected as target devices, if they are not overloaded already. Second, remote locations are involved and blocks have to be sent via the network. In both cases, *all* blocks are moved to other locations in order to shutdown the originating disk. After redistributing the disk load, the algorithm takes a final step and suspends all nodes, which do currently not serve a purpose (line 24–30).

3 Benchmarking for Energy Efficiency

So far, we have sketched the essential components of WattDB responsible for approaching our overall goal. However, testing a system for energy efficiency also requires a new benchmarking paradigm. Established procedures to measure DBMS performance cannot be used to get meaningful estimates of the system's power consumption (see below).

3.1 Benchmark Procedure

As we have shown in [SHK12], typical server workloads strongly vary in their utilization. In this study, we have examined the servers of an ERP service provider and monitored their workloads. We observed that servers are typically not fully utilized all the time. Instead, the servers are usually loaded between 20 and 30% of their peak performance; but these servers are not overdimensioned for their workloads: During some (short) period of the day, their full capacities are needed to satisfy the performance demands, either for processing the incoming OLTP workload or for generating OLAP reports. This observation was made by other studies as well. Barroso and Hölzle [BH07] have shown that a cluster of nodes is more than half of the time below 50% load. Yet, its energy consumption during underutilization is comparable to its peak.

Current database benchmarks (primarily TPC-*) lately incorporated power measurements to deliver additional runtime characteristics for their evaluation. TPC-Energy was defined as such an addition to the performance-centric measurements. Nevertheless, the power

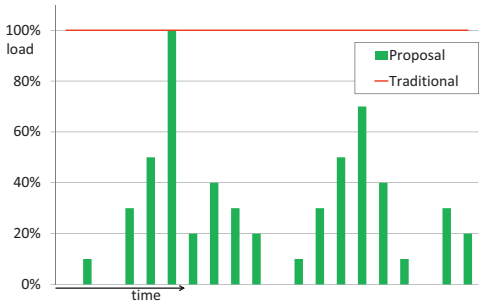


Figure 2: Traditional benchmarking compared to our proposal

consumption is only reported for idle and full utilization, leaving the typical DB server usage disregarded.

We will employ the benchmarking paradigm we proposed earlier [SHK12], which addresses typical usage patterns of databases and tries to mimic them in a benchmark run. Figure 2 depicts an exemplary benchmark run compared to traditional benchmarks. While traditional benchmarks only stress the system under test at 100%, the proposed benchmark will burden the system at different utilization levels to simulate such typical usage patterns.

Each benchmark step is scheduled to run for 30 minutes. If the DB cluster is able to finish the benchmark earlier, due to overprovisioned resources, the energy consumption of the time remaining is still recorded. In case the cluster cannot fulfill all queries in time, the total additional processing time and energy consumption is measured. These constraints are introduced in accordance to the benchmark definition in [SHK12].

3.2 Simulating OLTP Queries

Complex query capabilities are not yet implemented in WattDB, therefore, OLTP queries cannot be used to benchmark the database. Instead, the benchmark consists of a set of threads, executing an OLTP simulation. Each thread is representing one database client, running a series of OLTP queries.

Each query consists of a series of page reads, (artificial) processing steps and writes to simulate an OLTP query trace and to generate load at the storage layer; the processing nodes are not utilized much. Hence, this benchmark is heavily IO-intensive to empirically evaluate especially the storage layer and its energy-efficiency potential.

The benchmark operates on a 128 GB database with a primary-key index stored as a B*-tree. The database is preallocated on a single storage node which also contains the index. To circumvent the OS file system buffers and minimize the management overhead, no file system is used; instead, WattDB operates on raw disk devices. The database pages contain multiple records (which currently consist of an ID column and additional columns, filled with *junk* data to increase size). The inner leaves of the index fit into 2 GB of main memory, hence, after warming up, the buffer should contain a large fraction of the index.

The (simulated) OLTP clients randomly select IDs for reading records. For each request, the DBMS traverses the primary-key index to fetch the respective leaf page and locates the requested record inside the page. To emulate data processing, the threads generate CPU load by spin-locking. Finally, with a 1:4 chance, the page gets marked dirty in the buffer and, hence, has to be written back to the storage node at some time.[2] Afterwards, the benchmark thread goes to sleep for a specified time interval, before commencing the next read-process-write cycle. Such breaks are necessary when running an energy benchmark—as opposed to a performance benchmark. The system-under-test utilization can be tuned by adjusting the number of concurrently running clients, i. e., threads.

3.3 Measuring Energy Consumption

To measure the energy consumption of the cluster during the benchmark runs, we developed and installed a measurement framework. The basic power measurements are done using a custom measurement hardware. This device is capable of keeping track of each node in the cluster and the additional peripherals (i. e., the network switch). It can read the power consumption at a frequency of 300 Hz and report the values digitally over a standard USB connection. A software component is reading the measurements and aggregates all values to the cluster's total power consumption. This aggregate is then reported along with the benchmark runtime to a log file. Furthermore, the framework consists of a hardware device, capable of monitoring the energy consumption of up to ten nodes and infrastructure devices like ethernet switches as well, and a software component which aggregates and logs the measured data. Our framework can be integrated into the benchmark component and allows combining performance measurements with energy data. Although it would be possible to monitor the energy consumption of each node separately, we aggregated all energy measures to show the consumption of the cluster for the sake of clarity (and not their differing distribution in ten separate plots). A detailed description of the hardware device can be found in [HS11].

3.4 Experimental Setup

Figure 3 depicts the experimental setup. The database cluster, consisting of ten nodes and the ethernet switch[3], is powered by the measurement device, allowing us to monitor the power consumption of each server. The measured (analog) values (AC) are converted to digital readings and streamed to a connected computer executing the benchmarks based on a configuration file (Cfg) defining the specific usage patterns and sends queries to WattDB. By combining information from the benchmark runs (i. e., start and stop signals, duration) and the power measurements, the energy consumption for each benchmark run can be exactly determined. All data is written into a log file for later evaluation.

[2] The buffer decides which pages and when to write back.

[3] A single server would not need a dedicated ethernet switch, therefore we think it is fair to include it as a required hardware component into the measurements.

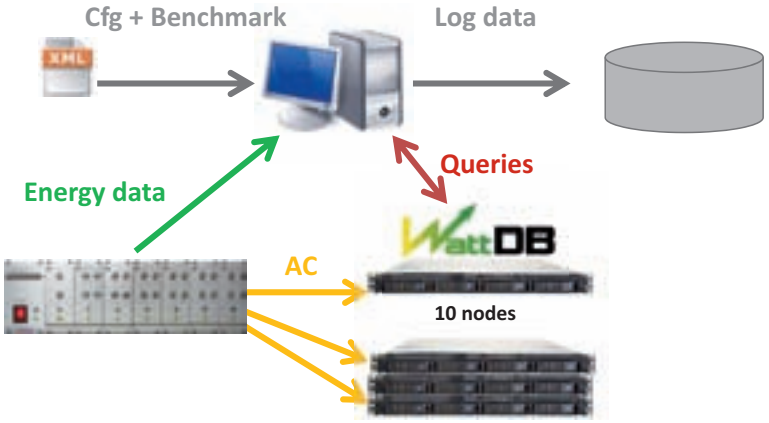


Figure 3: Experimental measurement environment

4 Measurement Results

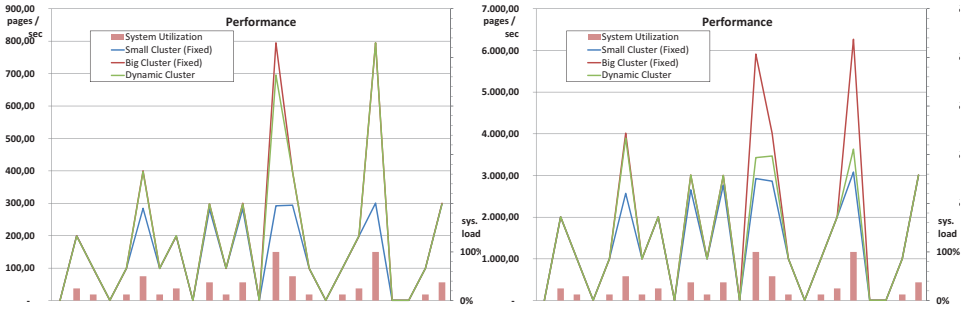
In this section, we explain the benchmarking methodology we used to verify our claims on a cluster of wimpy nodes, as previously described, and show the results of the runs. We have deployed the WattDB software with an energy management component in the cluster, connected to an energy measurement device. By running the benchmark against a cluster configuration, we expect the software to react to the changing workloads and power up/down nodes as needed. To make results comparable, we have run the identical load profile (as explained by Figure 2) three times.

For the first cluster configuration, we distributed the DB pages to two storage disks on one node and disabled the power management algorithm. Therefore, the number of nodes was fixed to the bare minimum of 3 (master, processing node and storage node) and the cluster was fixed to its most power-saving configuration delivering the lowest performance.

As next cluster configuration, we distributed the DB pages to all available disks and started the same benchmark, again with disabled power management. This time, all nodes (the master, one processing node and five storage nodes) were active and the cluster was able to work with maximum performance and, as a consequence, maximum power consumption.

The results of these two cluster configurations were used as baselines to estimate the performance and power coverage the cluster can achieve. Finally, we set up an unrestricted cluster configuration, with the power management component in full control of the cluster and its current workloads. We expected the cluster to adapt to the current workloads, as the benchmark runs proceeded.

During each of the runs, we measured the runtime and the energy consumption of the query phase from the start of the first query until the last query finished. Initially, the benchmarks were run on a cluster of seven nodes with 10 magnetic disks attached. Five nodes were acting as storage nodes, each having two disks attached, one was used as processing node and the remaining one was the coordinating master node. Later, we replaced the magnetic



(a) HDD **(b)** SSD

Figure 4: Performance: benchmark results for three cluster configurations

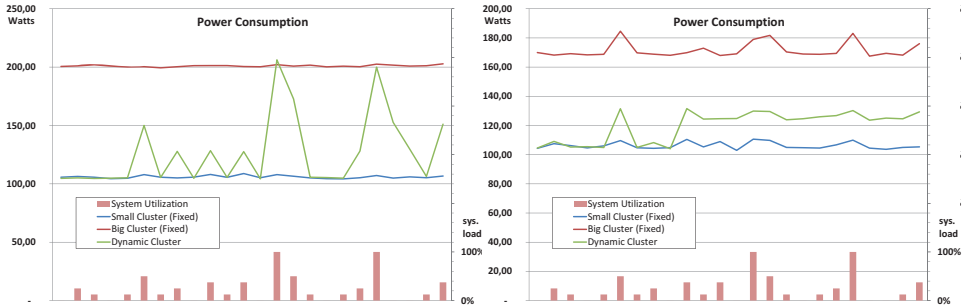
disks with 8 *Solid State Disks* (SSDs), thus reducing the number of storage nodes to four, and ran the benchmarks again. Since SSDs provide much more IOPS than traditional disks, we have increased the workload for the SSD benchmark by the factor of 10.

While running the benchmark against the three cluster configurations, energy consumption and runtime were reported to file. At the bottom of the result graphs, load profile or system utilization is depicted on the secondary axis. This is similar as in Figure 2 and only included for reference. Each graph plots the results of three cluster configurations: *Small Cluster* is showing results for the first run, where only two storage disks were active, thus forming the smallest possible configuration, *Big Cluster* is referencing to the second configuration with 10 (8) storage disks, and *Dynamic Cluster* depicts the measurements for an unrestricted run with the power management component active. In addition to the storage nodes, the master node and a single processing node were used in all three configurations.

On the left of each figure, the results of the configurations using magnetic disks are shown. The graphs on the right depict the results using SSDs. In all graphs, *sys load* refers to the utilization of the storage system, where 100% represent the maximum throughput the system could achieve (using all storage nodes and disks).

4.1 Performance

Figure 4 shows the performance graph for each run. As expected, the big cluster delivered the best performance and was even able to handle the highest utilization. The small cluster's performance broke down, due to it's constrained number of storage disks and the limited maximum IOPS. Finally, the dynamic cluster showed more or less identical performance to the big one, with small limitations in a case where dynamic adaptation caused some blocks to be moved between storage devices, which decreased the maximum performance. (It took only a few minutes to redistribute several Gigabyte of storage via Gigabit-Ethernet. By using compression, we were able to further reduce network traffic.) Compared to the total runtime of at least 30 minutes, redistribution cost was acceptable.



(a) HDD **(b)** SSD

Figure 5: Power consumption: benchmark results for the three cluster configurations[5]

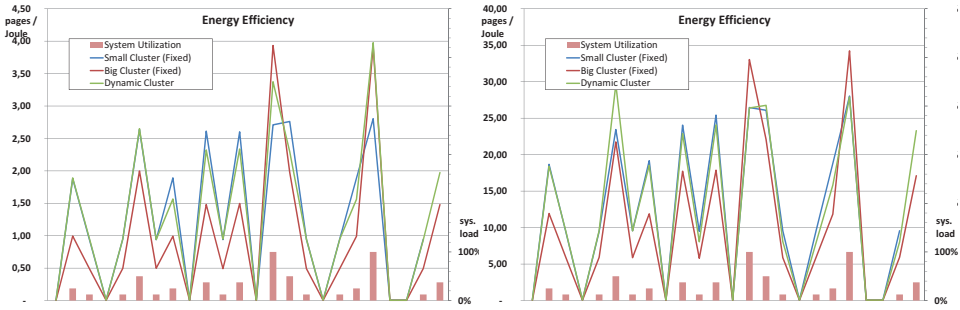
The performance using SSDs was much better, compared to magnetic disks. Still, the relative results are comparable, except for the dynamic configuration, which did not deliver the same peak performance as the pre-configured big cluster. When stressing the SSD cluster with heavy load, the performance of the cluster did not increase as expected. This might indicate optimization potential in the power management component or a bottleneck which was not monitored, e. g., CPU or network.

4.2 Power Consumption

Figure 5 visualizes the power consumption during the benchmark runs. Both fixed configurations exhibit a mainly static power consumption, because the number of nodes was fixed. The big cluster delivers no measurable difference for the HDD configuration between idle and full utilization.[4] Compared to idle, the SSDs exhibit a slightly increased power consumption under load. The dynamic configuration oscillates between the lowest and highest power consumption, as the cluster adapts to the workload. Using HDDs, the power management decided for our benchmark to use all available storage devices to share the load. For SSD configurations, however, not all storage devices were used, possibly because the storage was not over-utilized and some other component of the cluster was the bottleneck. Our power management decided to distribute the load to two storage disks on two separate nodes, instead of two disks on the same node. This is another indicator that the network was the limiting factor in the benchmarks, and not the IOPS of the SSDs.

[4]CPU-bound benchmarks might reveal different power characteristics.

[5]The use of enterprise server hardware would enlarge the relative distance between the curves of the small and big cluster. As a consequence, the overall saving of the dynamic cluster would have been amplified.



(a) HDD **(b)** SSD

Figure 6: Variation of energy efficiency for the three cluster configurations

4.3 Energy Efficiency

If we relate performance to energy consumption (which is power consumption times benchmark duration), we can calculate the energy efficiency for each of the runs, which is shown in Figure 6. Energy efficiency is expressed in pages per Joule, i. e., how many pages can be processed by consuming one Joule of energy. Not surprisingly, the small cluster exhibits the best energy efficiency during low utilizations. The big cluster is simply overprovisioned to satisfy the workload and consumes more energy to process the same amount of work, hence, its energy efficiency is worse.

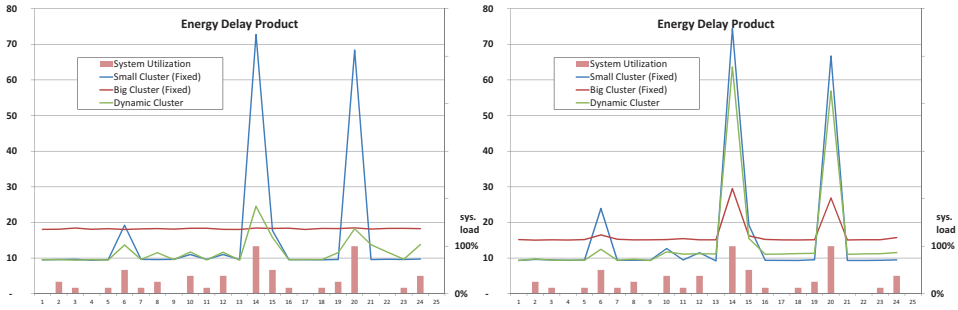
At full utilization, the situation turns in favor of the big cluster. The small cluster is not suited to handle the high utilization and needs almost 3 times as long as the big cluster to process the workload (not depicted here). As a consequence, the energy consumption of the small cluster is much higher and the energy efficiency accordingly lower.

The dynamic cluster powers storage nodes up and down according to the current workload. Therefore, under low utilization, its energy efficiency is identical to the small cluster. With rising load, the dynamic cluster powers up additional storage devices; hence, its energy efficiency gets comparable to that of the big cluster. Again, transition costs to move storage blocks decrease the energy efficiency in the dynamic case.

Using SSDs, energy efficiency is roughly 10 times better, although the difference between the small and the big cluster is not as prominent as with magnetic disks. The small cluster is still the most energy efficient one at low utilization and the big one only pays off at full load, but the reduced performance of the dynamic cluster affects its energy efficiency.

4.4 Energy Delay Product

When calculating energy efficiency, a system that is twice as slow, but consumes only half the power, will get the same score, because the same amount of work can be done using the same amount of energy. This calculation disregards the user expectation, as the



(a) HDD **(b)** SSD

Figure 7: Illustration of the EDP for the three cluster configurations

user/consumer is interested in getting results quickly. Therefore, we added another metric, which takes power consumption and query delay (i. e., the inverse of the performance) into account; the *Energy Delay Product* (EDP), a metric stemming from chip and microprocessor design [GH96]. EDP is defined as follows:

$$EDP = \text{ execution time} \times \text{ energy consumption } \textit{or}$$
$$EDP = \text{ execution time}^2 \times \text{ power consumption}$$

Hence, faster query execution times get rewarded more than power consumption, i. e., a system with twice the execution time must consume less than 1/4th of the power to get the same EDP rating. A lower EDP is generally favorable.

Figure 7 shows the EDP for the three benchmark runs. The small cluster exhibits the lowest EDP under low utilization, but does not perform well under heavy load. Starting at 50% utilization, the EDP of the small cluster outgrows the big cluster's EDP, because the load is too high for the small cluster and the execution time nearly triples. The big cluster shows a stable EDP, regardless of the workload. In most cases, the cluster is underutilized and, thus, more energy is consumed and the EDP is higher, compared to the small cluster. Only in peak-load situations, the additional performance of the big cluster pays off. The dynamic cluster shows the best overall EDP, with a similar score to the small cluster when not fully utilized and a slightly higher EDP in the peak-performance benchmarks.

Running the benchmark on SSDs, even the big cluster seems to have trouble handling the heavy workloads, hence, the rising EDP. The dynamic cluster is also unable to adjust the configuration to score a low EDP. As previously mentioned, this indicates a bottleneck beyond the reach of the current monitoring.

In summary, the measurements clearly illustrate that no fixed cluster configuration, neither a small one, nor a big one, is able to process the given workload in the most energy-efficient way. Hence, the results of the dynamic cluster can be considered as a proof of existence that, in specific cases, energy proportionality can be approximated for DBMS processing and that the increased effort pays off in terms of energy saving—without sacrificing too much performance.

5 Related Work, Conclusion & Outlook

As highlighted, a key result for energy efficiency of single-server DBMSs was published by Tsirogiannis et al. [THS10]. An exploration of a clustered DB system—close to our approach—was reported by Lang et al. [LHP+12]. Their contribution identified ways to improve energy efficiency in database applications by using a *static server cluster* instead of a single-server DBMS. Using a COTS (commercial off-the-shelf) parallel DBMS, they ran a single TCP-H query as workload and repeated their experiments on clusters where the cluster size/server configurations were set up for each test run. First, they reduced the number of beefy (powerful) nodes step-by-step from 16 to 8 nodes. Then, they gradually replaced the beefy nodes by wimpy (lightweight) ones. Both experiments resulted in decreased energy consumption, because the reduced cluster size or the lightweight nodes replacing beefy nodes needed less power, and in reduced performance. By analyzing the results, they revealed opportunities leading to improved energy efficiency. Although the experiments in [LHP+12] used only static server configurations and did not explain how the unused servers in the cluster could be turned on/off, they also delivered at least a kind of *existence proof* that research in DB clusters may lead to enhanced energy efficiency.

Our results are a step forward towards dynamically achieving energy proportionality. We have shown that energy proportionality can be approximated by using a cluster of commodity hardware and that tuning the system to a certain performance level comes with the drawbacks of limiting either the maximum processing power or the possible energy savings. By automatically adjusting the number of nodes to a running workload, which results in a balanced trade-off between performance and energy consumption, we demonstrated that it is possible to configure a cluster to process data in the most energy-efficient way. By dynamically reconfiguring the storage to fit the workload, a cluster of nodes reveals substantial energy-saving potential compared to big servers and also compared to statically configured clusters. The workload explored was rather simple and served as a starting point to reveal in our future work the data clusters and workload patterns requiring only limited reorganization/reallocation where such a storage server—while preserving its intended energy-proportional behavior—can be used. Nevertheless, our findings represent a milestone towards an energy-proportional DBMS, because the storage in traditional database systems accounts for more than half of the power consumption [PN08].

By using SSDs, IO is not the only limiting factor in the cluster. In the future, we will include CPU, main memory, and network into the monitoring scope as well to allow dynamic adjustments of all DBMS-relevant components. While scaling at the storage side was rather easy—it only required to copy/move data from one disk to another, while keeping the logical pointers up-to-date—, scaling at the processing side is more complex.

The experiments show that re-distribution of storage blocks and cluster balancing cannot be done frequently, due to the cost of shipping storage blocks via the network. By including historical measurements and forecast data, we are planning to extend the reactive power management component to become proactive [KHH12]. Workloads usually follow an easy-to-predict pattern, e.g., workdays are similar to each other, workloads in December keep rising for e-commerce back-end DBMSs, and so on. Therefore, we expect even better energy savings with a proactive cluster.

The granularity of control over performance and energy is a single server. With n nodes in the cluster, the finest grain of adaptation is $1/n$-th of the total power. In this work, we have run the experiments on a cluster of seven nodes (5 storage nodes). This configuration implies that the finest grain of control is 1/5th of the total, or 20%. For more fine-grained control, the number of nodes should be increased. Lastly, there is still optimization potential for power management, and many other factors besides forecast data can be included in the decision process such as CPU/network/memory utilization, information about the workloads, repeating patterns, etc. More specific data can lead to more intelligent use of resources and even better energy efficiency than we have shown in this paper.

References

[BH07] Luiz André Barroso and Urs Hölzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 40(12):33–37, 2007.

[BH09] Luiz Andre Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, 2009.

[GH96] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277 –1284, 1996.

[GHP+10] Clement Genzmer, Volker Hudlet, Hyunjung Park, Daniel Schall, and Pierre Senellart. The SIGMOD 2010 Programming Contest - A Distributed Query Engine. *SIGMOD Record*, 39(2):61–64, 2010.

[HS11] Volker Hudlet and Daniel Schall. Measuring Energy Consumption of a Database Cluster. In *Proc. 14-th GI-Conf. on Database Systems for Business, Technology and Web, LNI - P 180*, pages 734–737, 2011.

[KHH12] Christopher Kramer, Volker Höfner, and Theo Härder. Lastprognose für energieeffiziente verteilte DBMS. *Proc. 42. GI-Jahrestagung 2012, LNI 208*, pages 397–411, 2012.

[LHP+12] Willis Lang, Stavros Harizopoulos, Jignesh M. Patel, Mehul A. Shah, and Dimitris Tsirogiannis. Towards energy-efficient database cluster design. *PVLDB*, 5(11):1684–1695, 2012.

[OHS10] Yi Ou, Theo Härder, and Daniel Schall. Performance and Power Evaluation of Flash-Aware Buffer Algorithms. In *DEXA, LNCS 6261*, pages 183–197, 2010.

[PN08] Meikel Poess and Raghunath Othayoth Nambiar. Energy Cost, The Key Challenge of Today's Data Centers: A Power Consumption Analysis of TPC-C Results. *PVLDB*, 1(2):1229–1240, 2008.

[SBH+10] Alexander S. Szalay, Gordon C. Bell, H. Howie Huang, Andreas Terzis, and Alainna White. Low-Power Amdahl-Balanced Blades for Data Intensive Computing. *SIGOPS Oper. Syst. Rev.*, 44(1):71–75, 2010.

[SH11] Daniel Schall and Volker Hudlet. WattDB: An Energy-Proportional Cluster of Wimpy Nodes. In *SIGMOD Conference*, pages 1229–1232, 2011.

[SHK12] Daniel Schall, Volker Höfner, and Manuel Kern. Towards an Enhanced Benchmark Advocating Energy-Efficient Systems. In *TPCTC, LNCS 7144*, pages 31–45, 2012.

[THS10] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. Analyzing the Energy Efficiency of a Database Server. In *SIGMOD Conference*, pages 231–242, 2010.

Hibernating in the Cloud – Implementation and Evaluation of Object-NoSQL-Mapping *

Florian Wolf, Heiko Betz, Francis Gropengießer, and Kai-Uwe Sattler

Database and Information Systems Group
Ilmenau University of Technology
{*first.last*}@tu-ilmenau.de

Abstract: Object-relational mappers such as Hibernate are often used in applications to persist business objects in relational databases. The availability of commercial cloud-based database services opens new opportunities for developing and deploying database applications. In addition, highly scalable cloud services belong to the class of NoSQL systems promising to avoid the paradigm mismatch between the object-oriented programming model and the relational backend. In this paper, we discuss and analyze the usage of a scalable NoSQL solution such as Basho's RIAK as backend for Hibernate. We describe the necessary mapping and translation steps for an integration avoiding the detour on SQL. Finally, we present results of an experimental evaluation showing the benefits and limitations of this class of NoSQL backends for object-relational mappers.

1 Introduction

Today, modern business applications rely mainly on application server technologies such as Java EE or .NET where business entities like customers, products, or orders are represented by components or objects of an object-oriented language. In order to persist these objects, often SQL database systems are leveraged requiring an object-relational mapping. For this purpose, several frameworks exist. Most prominent examples are Hibernate [JBo], ADO.Net [ADO], and DataNucleus [Data], just to mention only a few. Although, object-relational mapping works quite well for most applications, it faces at least two drawbacks resulting from the underlying relational database technology:

Paradigm mismatch: Though, object-relational mappers hide the details and complexity of the mapping of the data structures and translation of object accesses to SQL queries, there is still a major difference. While in object-oriented applications traversing the graph of objects is the typical access pattern, operations in the relational model are set-oriented and access tuples by their attribute values. Thus, traversing an object relationship or retrieving a complex object results in complex joins and outer joins of multiple tables. This is also known as object-relational impedance mismatch [New06].

*This work was supported by the Thüringer Aufbaubank (TAB) under grant 2011 FE 9005.

Scalability issues: Relational database systems aim to provide strong consistency guarantees. In large distributed settings as required for example in Web 2.0 applications, these systems reach the limits in terms of scalability according to the CAP theorem [FGC+97].

A possible approach to overcome these shortcomings is to use so-called NoSQL systems like Amazon DynamoDB [AWS], Cassandra [Apa], RIAK [Bas], or Neo4j [Neo]. These systems favor scalability and availability over consistency. Furthermore, they provide more flexible schema support and evolution. For instance, a graph model as supported by Neo4j seems to be better suited for persisting object graphs since graph traversal is more similar to the typical access pattern of object-oriented applications than performing joins. Secondly, systems like DynamoDB do not require a schema in the classic sense: each object can have its own set of attributes which makes it easier to add or remove new object classes than the rather limited `ALTER TABLE` operations in SQL.

Basically, two approaches for persisting application data in a NoSQL system exist – *Persistence API* or *Mapping framework*. The former one is tailored to a specific NoSQL solution. Examples of this approach are Morphia [Mor], an object mapper for MongoDB [Mon], as well as HelenaORM [Hel] and Object-Cassandra-Mapper [OCM], which are object mapper for Cassandra. The latter approach can deal with different NoSQL systems. Examples are DataNucleus, DataMapper [Datb] as well as Hibernate OGM [Hib].

Obviously, the mapping framework approach is the more flexible one. Especially today, where many cloud providers like Amazon, Microsoft, or Google offer different NoSQL services with different features and pricing models, it enables application developers to write their applications without tailoring them to a specific system and allows also to migrate to other storage systems.

Although, several non-relational object mapping frameworks already exist, the most famous one – Hibernate – lacks general support for different NoSQL systems. Currently, only Infinispan [Inf] is supported. The problem of this solution is that it tries to map the relational model to the data model of the NoSQL system, which leads to the mentioned paradigm mismatch.

Hence, the question we try to answer in this paper is, whether it is possible to integrate NoSQL support in Hibernate by avoiding the intermediate mapping to the relational model and while keeping compatibility to existing applications. We address this question by presenting a general "non-relational mapping" approach and discussing the integration in the Hibernate framework. Furthermore, we describe the implementation of this approach using the RIAK system as backend for Hibernate and discuss still existing and inherent limitations. Beside the functional properties, we investigate performance and scalability of our Hibernate/NoSQL in comparison to a setup with a traditional MySQL backend.

2 Related Work

In fact, the discussion about the usefulness of relational databases as storage backends for all kinds of applications dates back to the early 80s. Already in [HL82], limitations of relational database system, for example missing support for complex types and interactive access patterns needed for efficient support of engineering design, are described.

The emerging success of the object-oriented programming language as well as the need for adequate storage backends for CAD systems [Wol91] led to the development of a variety of so called non-standard database systems. In [CM84] the authors propose GemStone – one of the first commercial object-oriented database system. Object-oriented databases try to avoid a paradigm mismatch, as existing between object-oriented applications and relational database systems, by enabling native persistence of application objects. They fully support object-oriented principles such as encapsulation, inheritance, or object relations. Further examples for non-standard database solutions are KUNICAD [HHLM87], a database system to support geometric modeling for CAD applications, or PRIMA [HMWM+87], a database system to support design applications such as VLSI design and software engineering .

With the beginning of the 90s, the trend was shifting from object-oriented database systems to object-relational database systems [SM95]. The absence of standardized data models and declarative query languages were only some reasons for this development [Bro01]. Object-relational database systems are an extension of relational database systems – combining object-oriented principles with a powerful query language. Examples are Informix [IBM] or Oracle [Ora]. However, the impedance mismatch could not be fully solved with this technology, preventing a total success of object-relational database systems.

A contrary approach to object-relational database systems is the integration of SQL into the application layer. Well known techniques here are Module Language, Embedded SQL, or Direct Invocation [VP95]. One of the goals of these developments is to solve the impedance mismatch by abandon the definition of an object-relational mapping. However, one of the problems is the tight binding of the application to a specific database. Switching the database can lead to deep changes in the application code.

With the development of CLI (Call Level Interface), the binding got more abstract. JDBC and ODBC are concrete implementations of this standard. They can be seen as important steps on the way to object-relational mapping frameworks already mentioned in the introduction.

Driven by the current trends in cloud computing, object mapping to NoSQL storage backends becomes more and more important. NoSQL systems provide the availability and scalability guarantees needed in today's business applications. Some examples for appropriate object-to-NoSQL mappers like Morphia or HelenaORM are already mentioned in the introduction. A further famous commercial solution is Google's App Engine [App]. It provides built-in mapping from Java objects to Google's BigTable [CDG+06].

Summarizing, the problem of persistent storage of application objects is an old problem. Using relational backends has the advantage of a proven, standardized data model as well as a powerful query language. However, due to different paradigms this introduces problems summarized under the term impedance mismatch [IBNW09]. Alternatives are native backends like object-oriented database systems. The current developments in storage techniques along with the increasing success of cloud computing are making native storage solutions more attractive again and provide the context for our work.

3 Hibernate ORM Framework

The Hibernate ORM framework is an Open Source project which allows for easy persistence of Java application objects in a relational fashion, using almost any available relational database system as storage backend via JDBC. Its data model follows the object-oriented approach, requiring only less modifications to the application code. Hibernate also supports relationships (associations) between objects. 1:1 associations are implemented by object references as members, whereas 1:N, N:1, N:M associations are implemented by embedded Java collections.

Basically, each Java class is mapped to a table and each object is mapped to a table row, whereby the object properties (class members) constitute table columns. Furthermore, associations between objects are stored in normalized form using foreign key relations. Necessary relation tables (join tables) for N:M associations are automatically created.

Persisting and deleting objects is supported by simple save, update, and delete operations. Retrieving objects is either performed manually using SQL-like HQL (Hibernate Query Language) queries or the Hibernate Criteria API or automatically during object access. In many applications, object retrieval is typically performed by *i)* querying an object that acts as an entry point and *ii)* traversing associations to other objects by accessing the object's references via the provided getter methods. Thereby, referenced objects are automatically retrieved by the Hibernate framework.

Under the hood, data model mapping, persisting and retrieving objects as well as traversing object associations result in relational database specific SQL statements. While this works well in case of simple object persistence and retrieval, several problems arise in case of association traversal due to the underlying relational data model (impedance mismatch), for which we give a short example in the following.

Many applications make extensive use of hierarchical data structures. In the simplest case, they implement trees, where nodes are just connected by parent-child relationships. Hibernate supports the bidirectional storage of these relationships. Every node in the tree stores a reference to its parent as well as a set of references to its children. In other words, all associations are stored with the object and hence, traversal in both directions – top down and bottom up – is well supported.

The relational data model treats trees as 1:N relationships. In normalized form, relations are stored unidirectionally at the N-side. Hence, bottom up traversal is favored. Retrieving a node's children means either joining parent and children table or performing a selection on the children table with the parent's key. If we just want to obtain the number of a node's children, we have to perform a `SELECT COUNT` on the children table. In the object oriented data model we just retrieve the size of the reference set.

In the general case, the relational data model separates objects from their associations through relation tables. This leads to additional scans and joins and hence a behavior that contradicts the object-oriented approach. Furthermore, the fixed relational schema prevents easy adaption to new associations. It is not possible to easily extend one-to-many associations to many-to-many associations. This would result in significant changes to the schema.

Since NoSQL storage solutions do not rely on fixed schemas, they seem to be well-suited candidates to build a storage backend that comes closer to the object-oriented data model. Hence, we try to solve the above stated problems by defining a non-relational object mapping described in the following sections.

4 RIAK as Hibernate Backend

Looking at the current market of existing cloud based NoSQL solutions reveals that two different storage models exist – wide-column stores and key-value stores. Wide column stores provide a schema-less and flexibly structured data model comparable to spreadsheets with access to single columns of a row. In contrast, key-value stores treat data just as BLOBs, which are accessible only as a whole via unique keys. Although, the data model is more low-level than the one provided by wide-column stores, we have chosen the key-value store approach instead of a wide-column store due to the following reasons. Firstly, wide-column stores typically rely on key-value stores in the background. Hence, we can create our own wide-column store with custom features if needed. Secondly, Hibernate only retrieves complete objects. Partial access to single properties of an object is not needed. Hence, usually it is not necessary to store data in a spreadsheet-like fashion. However, if objects shall be selected based on certain properties other than keys, knowledge of the internal structure is indeed required. In order to support these kinds of selections, most providers have introduced MapReduce support, e.g., Amazon with Elastic MapReduce [AWS], which works seamlessly with underlying key-value stores. By this means, selections on those BLOBs are as efficient as in a wide-column store.

For our work, we have chosen RIAK as a concrete key-value store backend for Hibernate. RIAK is open source and roughly comparable to Amazon S3 [AWS]. Data is organized by buckets and keys. Buckets define a virtual key space and, hence, group data logically. A value is uniquely identified by the bucket name and the key. It contains BLOB data as well as an arbitrary set of the so-called RIAK links for establishing references to other key-value pairs. A single RIAK link contains the bucket and key of the referenced key-value pair. Additionally, it is possible to name an association by specifying a tag. RIAK links are explicitly accessible through RIAK's API.

RIAK's API provides the following main operations: get/put/delete a single key-value pair, list existing buckets as well as keys within a specified bucket, and get/set RIAK links on a value. Additionally, it supports MapReduce integration, e.g., the built-in link-walk for traversing RIAK links. Starting from a given key-value pair it is possible to follow the outgoing links.

5 Hibernate on RIAK – Integration Concept

Integrating a key-value store like RIAK in an object-relational mapping framework comprises two tasks: mapping the object-oriented domain model to the data structures of the key-value store and replacing the translation to SQL statements by API calls of the key-value store. In the following sections we describe both steps for RIAK. However, it should

be noted that this approach can be easily applied also to other key-value stores, including Amazon's cloud services.

5.1 Data Model Mapping

With having both RIAK's data model and API in mind, mapping Hibernate's class and object model is a straightforward approach. Each object is mapped to a single key-value pair. For each class, a bucket is created. In this way, all instances of a certain class can be retrieved with list bucket operations. The properties of an object are stored within the value of the key-value pair. As serialization format we use JSON, since it is easy to use and supported on most platforms. Associations between objects are represented by RIAK links.

Associations between objects can be stored either unidirectionally or bidirectionally. In the latter case, two related objects link each other. Thus, RIAK linking is very flexible compared to the relational data model. Reconsider our tree example from Section 3. In RIAK, we can store both – the link to the parent within the child objects as well as the links to the children in the parent object. If just the number of children or their keys are needed, only a single lookup of the parent object has to be performed. Furthermore, we can easily extend one-to-many associations to many-to-many associations and, hence, represent general object graphs by just adding new links.

5.2 Connecting Hibernate and RIAK

For integrating Hibernate and RIAK, the most interesting parts of the Hibernate architecture [JBo] are the *Session* and *JDBC* components. The session component provides an interface containing core functionality for persisting and retrieving objects. The JDBC interface connects Hibernate to a specific relational database by loading an appropriate JDBC driver. In the following, we discuss possible anchor points where RIAK support could be plugged in.

Session-API reimplementation: A first approach is to replace the original implementation of the Session component by a RIAK-specific component. In this way, we could fully exploit all RIAK specific features. However, in this case almost the entire functionality of Hibernate, including object life-cycle management, has to be reimplemented requiring a tremendous effort.

JDBC-API reimplementation: The second approach is to provide a RIAK driver which implements the JDBC API. The disadvantage of this approach is that it requires parsing all SQL statements and rewrite them to RIAK API calls – something that we try to avoid.

Hooking into the Session component: The most promising approach is to plug in the interaction with RIAK into the Session component. In this way, we have access to SQL-independent data structures and still profit from core Hibernate functionality like object-life-cycle management. Hence, we have chosen this approach and describe the details in the following.

Based on the discussion above, RIAK backend support is integrated as shown in Figure 1. In fact, all highlighted components have been modified in order to intercept and redirect communication to our *RIAK mapping framework* instead of calling the JDBC interface.

In order to save, update, and delete objects (label 0) during object-life-cycle-management, Hibernate provides several `Persister`-classes for different purposes. If the `Persister` needs already stored objects, e.g., for resolving associations, it contacts the `Loader` (label 3). The `Loader` fetches the required objects from the persistent storage and returns them to the `Persister`. Similar to the `Persister`, several `Loader` classes for different purposes exist. In order to retrieve objects via HQL or the Criteria API (label 1 and 2) specific `QueryTranslators` are used to parse the query in an abstract syntax tree (HQL AST). In this case, the `Loader` is only responsible for redirecting the HQL AST and returning the results as application objects.

The core of our extension is the `Engine` component. It performs the mapping from objects to key-value-pairs with the help of JSON serializers and deserializers. Furthermore, it initiates RIAK API operations. Following, we describe our extension in case of saving an object, traversing an 1:N association, and executing a HQL query in more detail.

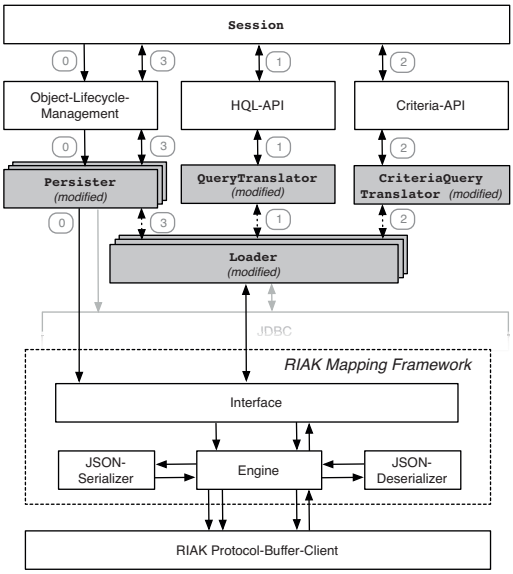


Figure 1: RIAK integration within Persister/Loader

For saving an object in RIAK, the object identifier and the object's entity type are required. Additionally, information about the attributes of the object and its associations are needed. Figure 2 depicts the resulting information flow.

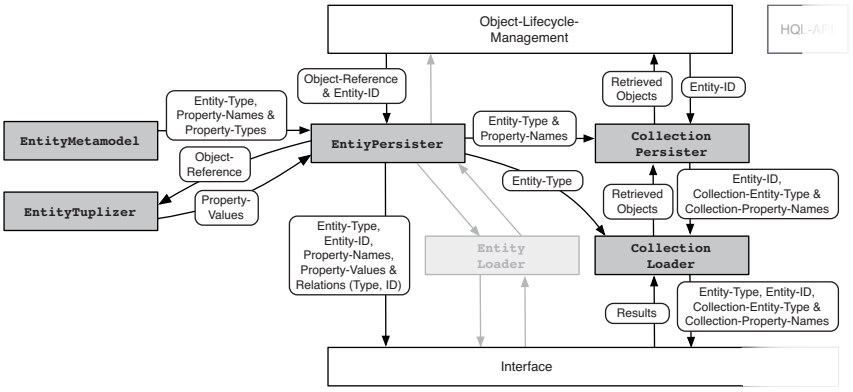


Figure 2: Saving an object and traversing 1:N associations

A Java object that has to be saved is passed to the `EntityTuplizer` by the `EntityPersister` for extracting the object's property values. The necessary information about the object type, its properties, and its associations are represented by the `EntityMetamodel`. The `EntityPersister` component has been modified to pass the information to the RIAK mapping framework which is responsible for storing the object persistently.

Performing a single traversal step during a top-down traversal in a tree leads to the information flow shown on the right hand side in Figure 2. Because in this case a 1:N association has to be resolved, Hibernate uses the appropriate `CollectionPersister` and `CollectionLoader` classes. The `CollectionLoader` has also been modified. It extracts information about the entity and the collection and forwards them to the RIAK mapping framework. The mapping framework retrieves all related objects and returns them as JDBC result set – a format natively understood by the `CollectionLoader`. In this way, we do not have to take care about proper object creation which is still done by the Hibernate framework.

Figure 3 illustrates the information flow during the processing of a HQL query. The query string is translated into a HQL AST by the `HqlParser` and passed to the `Walker` which extracts necessary information like requested entity types or property names. For interacting with the RIAK mapping framework, the `QueryTranslator` and `QueryLoader` have been slightly modified. The RIAK mapping framework retrieves the requested data from the storage backend and passes them back to the `QueryLoader` as JDBC result set.

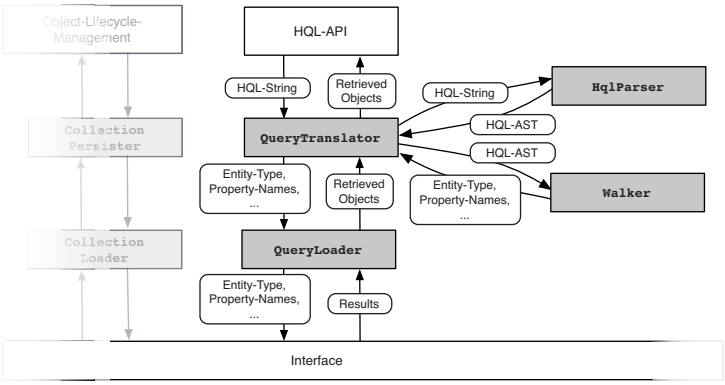


Figure 3: HQL-Query Loading

All object-related operations such as save, update, delete, and query are implemented in the RIAK mapping framework by performing appropriate RIAK API calls. The required information are gathered from the Hibernate core as described above. For saving an object, the value is obtained by serializing the object into a JSON string. The `EntityID` represents the necessary key. All associations to other objects are added as links to the meta-information of the object. For retrieving a 1:N association, the object on the 1-side is retrieved with the help of the provided `EntityID`. Using the `CollectionEntityType` ,the type (and hence the bucket) of the N-side objects can be identified. The linked objects (RIAK values) are retrieved in parallel by several threads. The results are deserialized and stored in a shared result list.

6 Evaluation

In the previous section, we have shown that the integration of RIAK in Hibernate as storage backend is basically possible. Instead of forcing a relational mapping to RIAK, we chose a storage model which is close to the object-oriented data model in a way that associations are stored with objects and not separated from them. In this section, we investigate whether this approach is beneficial in terms of performance and scalability. In order to better classify the test results, we compare them to a centralized relational backend. We chose MySQL [MyS], since it is freely available and widely used. However, we stress that the goal of this evaluation is not a comparison of both backends. Instead, MySQL just acts as a baseline to be able to assess the performance and scalability behavior of the NoSQL backend.

6.1 Micro Benchmark

To the best of our knowledge, no standardized Hibernate benchmarks exist. Hence, we perform micro benchmarking in order to compare both approaches. Our workload com-

prises three scenarios which are typical for most business applications – retrieving a single object, storing a single object, and traversing object associations. The first scenarios describe the retrieval of an existing instance and the storing of a new instance of a given class. Typical examples are requesting information regarding a given customer or creating a new customer. The latter scenario focuses on the most critical part of the object-relational mapping – the association handling. A widely spread traversal scenario is, as already mentioned, tree-traversal – either top-down or bottom-up. A typical example is: retrieving all order positions from a given order belonging to a given customer.

The test data is randomly generated. A class comprises ten attribute types – five string and five integer values. Objects are uniquely identified by a key (MD5 hash value). In MySQL, the key is used as primary key. In RIAK, the key is used to unambiguously identify a key-value pair.

The used metric is the time needed for storing and retrieving objects. The time is measured between issuing the operations and accessing the operation results. Including the time for accessing the results is necessary, because Hibernate works with lazy fetching. In detail this means, an object is not initialized till the first method invocation occurs. Based on the measured time, other metrics can be calculated, e.g., the number of queries per seconds.

During the tests, the number of objects in the data store, the number of concurrent client requests as well as the object size are changed. In order to prevent cache issues, the whole data store is cleaned between different tests. In order to get stable results, the tests are executed several times with different keys.

6.2 Test Setup

For both setups (RIAK and MySQL), we use virtual machines which are distributed over seven physical hosts connected by 1 GBit/s Ethernet. Each host has two Intel Xeon CPUs E5645 (12 cores and 24 threads; 2.40 GHz) and 40 GB of RAM. Each virtual machine has 1 CPU, 8 GB of RAM, 512 MB of Swap, and 4 GB of hard disk space.

The RIAK setup consists of 32 nodes. The number of partitions is also set to 32. Each node corresponds to a single virtual machine. The storage mechanism in RIAK is set to `riak_kv_memory_backend`. This means, all data is stored in main memory. All further configuration options are set to default values.

MySQL runs in a single virtual machine, with 40 GB of hard disk. No cluster or replication techniques are used. All configuration options are set to default values. MyISAM is used as storage engine. The primary key in each relation is indexed.

The client machine executing the tests is also virtual and has 12 cores. The used Java version is 1.6.0_26. All further configuration options are set to default values.

Of course, comparing a hard disk setup against an main memory setup is quite unfair. However, letting RIAK read/write from disk would just result in an offset shift of the access times. It does not influence the trend of the access times due to RIAK's distributed characteristics.

6.3 Hypotheses

Following, we outline the hypotheses for our evaluation. They summarize RIAK's performance and scalability behavior we expect during the execution of the test scenarios.

RIAK is based on consistent hashing [KLL+97] which distributes data across all nodes equally. Hence, we expect that the time for storing and retrieving a single object scales at most linearly with increasing number of stored objects. In MySQL, the index is used for retrieval and has to be updated in case of an insertion. Hence, we expect that times for storing and retrieving a single object increase with the increasing number of stored objects.

The time for storing and retrieving a single object depends on its size. Here, we expect a similar behavior for both backends. The times for storing and retrieving a single object increase with its increasing size due to more data traffic.

With the growing number of parallel requests, we expect at most a linear scaling in RIAK, since all requests can be equally distributed across all nodes. Since MySQL is centralized, it scales worse than RIAK.

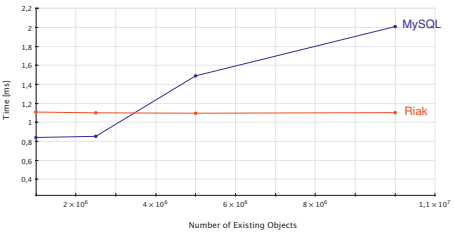
Top-down and bottom-up traversal can be performed very efficiently with RIAK. In the first case, child objects of a given parent object are identified via RIAK links and are fetched by parallel working threads. With MySQL, each child object is fetched via a select query. Although, this can also be done in parallel, we expect worse performance than with RIAK, because of the lack of horizontal scalability. In case of bottom-up traversal, MySQL performs joins. In RIAK, every child holds a RIAK link to its parent. We expect a performance drop by using MySQL with the increasing number of stored objects, whereas RIAK scales at most linearly.

In case of top-down traversal, we expect that MapReduce performs better than our externally implemented traversal algorithm, because data is directly processed in parallel on the RIAK nodes. Although, MapReduce introduces additional overhead, we expect better scaling with the increasing number of child objects compared to the externally implemented traversal algorithm.

6.4 Implementation and Analysis

Retrieving a single object A single object is retrieved by its unique key. Figure 4(a) and 4(b) show the results. RIAK introduces additional overhead due to its distributed character. This leads to worse response times compared to MySQL in case of a small number of stored objects (Figure 4(a)). However, as expected, the response times evolve almost constantly with the increasing number of stored objects. At a storage size of 500,000 objects, RIAK starts outperforming MySQL. The response times in case of an increasing object size also evolve as expected. With default configuration, MySQL supports only row sizes up to approximately 8,000 Byte without using BLOBs or text. Hence, we finished testing at this point.

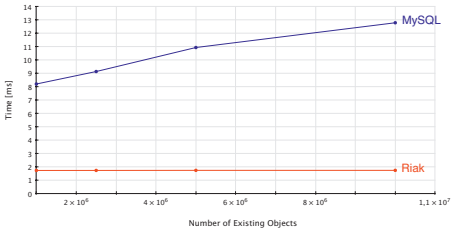
Storing a single object A new object is stored under a unique key. Figure 5(a) and Figure 5(b) show the results. As in case of object retrieval, the times for storing an object



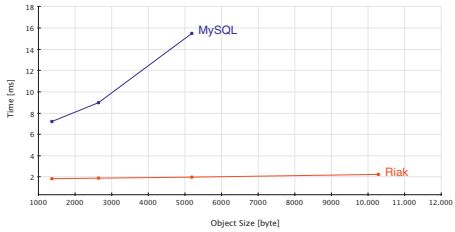
(a) Object Size 1362 Byte

(b) Number of Existing Objects 1000,000

Figure 4: Retrieving a Single Object



(a) Object Size 1362 Byte

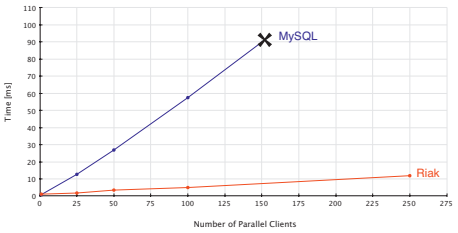


(b) Number of Existing Objects 1000,000

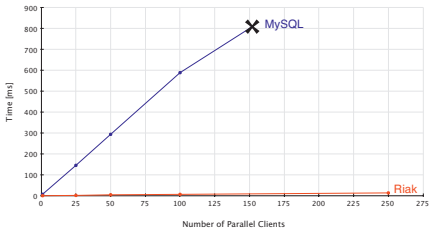
Figure 5: Storing a Single Object

in RIAK evolve almost constantly. Here, MySQL performs worse in all cases. Reasons might be, as already mentioned, additional index updates and hard disk accesses. The latter aspect has tremendous impact on the performance with an increasing object size (Figure 5(b)). With RIAK, storage times only increase slightly with an increasing object size. This is in contrast to our expectation.

Scalability In this scenario, we investigate the scalability characteristics of both backends under an increasing number of concurrently working clients. Every client retrieves or stores an object. Figure 6(a) and 6(b) show the measured times when all clients finished operation execution. As expected, RIAK scales almost constantly with an increasing load size. MySQL shows worse performance values. The highlighted cross in both charts indicates a MySQL connection problem. It was not possible to connect more than 150 clients simultaneously.

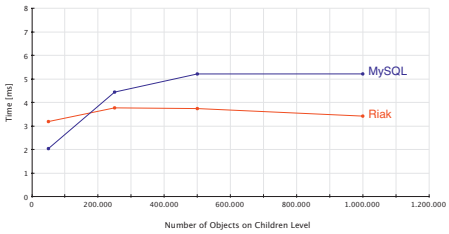


(a) Retrieving a Single Object

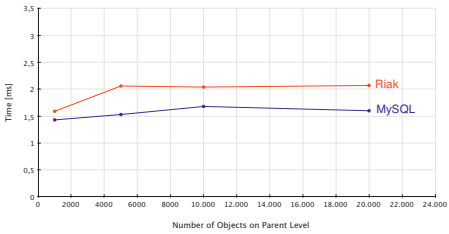


(b) Storing a Single Object

Figure 6: Object Size 1362 Byte, Number of Existing Objects 1000,000

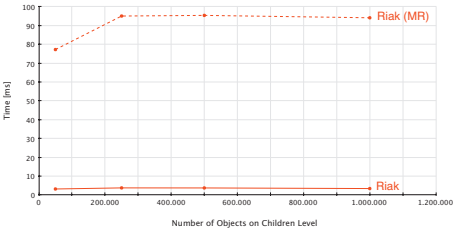


(a) Top-Down Traversal

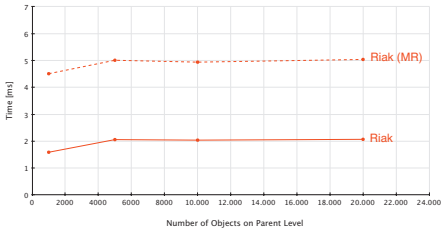


(b) Bottom-Up Traversal

Figure 7: 1 Level Traversal, 50 Children per Parent, Object Size 1362 Byte



(a) Top-Down Traversal



(b) Bottom-Up Traversal

Figure 8: MapReduce, 1 Level Traversal, 50 Children per Parent, Object Size 1362 Byte

Traversing object associations Here, we focus on top-down as well as bottom-up traversal in a tree structure. In the first case, we retrieve all child objects belonging to a given parent object. In the latter case, we retrieve a given child's parent. Figure 7(a) and Figure 7(b) depict the retrieval times for RIAK and MySQL. As expected, during top-down traversal, RIAK scales almost constantly with an increasing number of children, since all child objects can be retrieved in parallel. Due to the small amount of child objects, also MySQL shows good retrieval times. However, performance might drop if the number of child objects increases tremendously as depicted in Figure 4(a). The bottom-up traversal reflects the overhead in RIAK due to its distributed character. Here, only a single object, the parent of a given child, is retrieved. Although, MySQL executes a join in order to determine the parent of a given child, it performs better than RIAK. However, if we would increase the number of levels during traversal, this would result in more join operations which increase traversal times. In RIAK, since every child knows its parent (bidirectional RIAK links), the number of retrieved objects equals the number of traversal levels. This might have a positive effect on retrieval times.

Figure 8(a) and Figure 8(b) depict the comparison of our externally implemented traversal algorithm and the MapReduce traversal algorithm. In contrast to what we expected, our algorithm performs better than MapReduce. The reason might be, that our algorithm retrieves objects in parallel (with parallel connections) whereas in the case of MapReduce all determined child objects are returned as an entire result over a single connection.

7 Conclusion

The goal of our work was to integrate the key-value store RIAK as backend in Hibernate, avoiding an intermediate mapping to the relational model. Thereby, we tried to answer the question, whether a scalable NoSQL backend (for which RIAK is just an example sharing many common properties with, e.g., Amazon DynamoDB and S3) is feasible and fulfills the expectations in terms of performance and scalability. Section 5 shows that such an integration is possible, but requires several changes in the Hibernate core. Currently, we have only implemented a subset of Hibernate's features including save, update, and delete operations as well as a basic HQL support. Exploiting all of Hibernate's features in combination with RIAK might require substantial modifications in the Hibernate framework. Especially, enabling full HQL support on RIAK is a challenging task.

In the evaluation part of this paper, we investigated performance and scalability behavior of RIAK and MySQL with respect to typical Hibernate application scenarios. The goal was not to compare both systems, but rather to let MySQL act as a baseline in order to answer the question if and when RIAK has performance advantages. Although, RIAK introduces some overhead which hampers performance in small scenario setups, the test results show that Hibernate profits from the distribution and scalability characteristics of RIAK. This is especially the case in situations, where system load in terms of data size, the number of objects, and the number of concurrent requests is increasing tremendously. The measured times for bottom-up traversal do not reflect RIAKs potential. However, with an increasing number of traversal levels, we expect a substantial performance boost. In contrast to our expectations, the built in MapReduce support does not lead to additional speedup in our test scenarios.

References

[ADO] ADO.NET Overview. `http://msdn.microsoft.com/en-us/library/h43ks021(v=VS.100).aspx`.

[Apa] The Apache Cassandra Project. `http://cassandra.apache.org`.

[App] Google App Engine. `https://appengine.google.com`.

[AWS] Amazon Web Services. `http://aws.amazon.com/`.

[Bas] Basho: Basho Documentation. `http://wiki.basho.com`.

[Bro01] P. Brown. *Object-relational database development: a plumber's guide*. Number Bd. 1 in Prentice Hall PTR Informix series. Prentice Hall, 2001.

[CDG+06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15. USENIX Association, 2006.

[CM84] George Copeland and David Maier. Making smalltalk a database system. *SIGMOD Rec.*, 14(2):316–325, June 1984.

[Data] DataNucleaus. `http://www.datanucleus.org`.

[Datb] DataMapper. `http://datamapper.org`.

[FGC+97] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-Based Scalable Network Services. In *SOSP*, pages 78–91, 1997.

[Hel] marcust/HelenaORM. `https://github.com/marcust/HelenaORM`.

[HHLM87] Theo Härder, Christoph Hübel, Stefan Langenfeld, and Bernhard Mitschang. KUNICAD - A Database System Supported Geometrical Modeling Tool for CAD Applications (in German). *Inform., Forsch. Entwickl.*, 2(1):1–18, 1987.

[Hib] Hibernate OGM. `http://docs.jboss.org/hibernate/ogm/3.0/reference/en-US/html_single/`.

[HL82] Roger L. Haskin and Raymond A. Lorie. On extending the functions of a relational database system. In *Proceedings of the 1982 ACM SIGMOD international conference on Management of data*, SIGMOD '82, pages 207–212. ACM, 1982.

[HMWM+87] Theo Härder, Klaus Meyer-Wegener, Bernhard Mitschang, Andrea Sikeler, and Andrea Sikeler. PRIMA - a DBMS Prototype Supporting Engineering Applications. In *VLDB*, pages 433–442, 1987.

[IBM] IBM - Informix product family - Relational, embeddable and hassle-free offerings. `http://www-01.ibm.com/software/data/informix/`.

[IBNW09] Christopher Ireland, David Bowers, Michael Newton, and Kevin Waugh. A Classification of Object-Relational Impedance Mismatch. In *DBKDA*, pages 36–43. IEEE Computer Society, 2009.

[Inf] Infinispan - Open Source Data Grids - JBoss Community. `https://www.jboss.org/infinispan`.

[JBo] Hibernate - JBoss Community. `http://www.hibernate.org`.

[KLL+97] David R. Karger, Eric Lehman, Frank Thomson Leighton, Rina Panigrahy, Matthew S. Levine, and Daniel Lewin. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *STOC*, pages 654–663. ACM, 1997.

[Mon] MongoDB. `http://www.mongodb.org`.

[Mor] Morphia - A type-safe java library for MongoDB. `http://code.google.com/p/morphia/`.

[MyS] MySQL :: The world's most popular open source database. `http://www.mysql.com`.

[Neo] Neo4j: World's Leading Graph Database. `http://neo4j.org`.

[New06] Ted Neward. The Vietnam of Computer Science. `http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx`, 2006.

[OCM] charliem/OCM. `https://github.com/charliem/OCM`.

[Ora] Contents. `http://docs.oracle.com/cd/B19306_01/appdev.102/b14260/toc.htm`.

[SM95] Michael Stonebraker and Dorothy Moore. *Object Relational DBMSs: The Next Great Wave*. Morgan Kaufmann Publishers Inc., 1995.

[VP95] Murali Venkatrao and Michael Pizzo. SQL/CLI - A New Binding Style For SQL. *SIGMOD Rec.*, 24(4):72–77, December 1995.

[Wol91] Wayne Wolf. Object Programming for CAD. *IEEE Des. Test*, 8(1):35–42, January 1991.

Subspace Clustering for Complex Data

Stephan Günnemann

Data Management and Data Exploration Group
RWTH Aachen University, Germany
guennemann@cs.rwth-aachen.de

Abstract: Clustering is an established data mining technique for grouping objects based on their mutual similarity. Since in today's applications, however, usually many characteristics for each object are recorded, one cannot expect to find similar objects by considering all attributes together. In contrast, valuable clusters are hidden in subspace projections of the data. As a general solution to this problem, the paradigm of *subspace clustering* has been introduced, which aims at automatically determining for each group of objects a set of relevant attributes these objects are similar in.

In this work, we introduce novel methods for effective subspace clustering on various types of complex data: vector data, imperfect data, and graph data. Our methods tackle major open challenges for clustering in subspace projections. We study the problem of redundancy in subspace clustering results and propose models whose solutions contain only non-redundant and, thus, valuable clusters. Since different subspace projections represent different views on the data, often several groupings of the objects are reasonable. Thus, we propose techniques that are not restricted to a single partitioning of the objects but that enable the detection of multiple clustering solutions.

1 Introduction

The increasing potential of storage technologies and information systems over the last decades has opened the possibility to conveniently and affordably gather large amounts of complex data. Going beyond simple descriptions of objects by some few characteristics, such data sources range from high dimensional vector spaces over imperfect data containing errors to network data describing relations between the objects. While storing these data is common, their analysis is challenging: the human capabilities of a manual analysis are quickly exhausted considering the mere size of the data. Thus, automatic techniques supporting the user in the process of knowledge extraction are required to gain a benefit from the collected data.

The concept of Knowledge Discovery in Databases (KDD) [HK01] has been evolved as a possible solution for the above challenge and it is coherently described by a multilevel process the user has to follow (cf. Figure 1). Given the raw data, which is rarely perfect since, e.g., missing entries, inconsistencies, or uncertain values are prevalent during the data acquisition phase, the KDD process starts with a preprocessing step to clean the data. This step is often referred to data cleansing and tries to increase the data quality to support the subsequent data mining step. The goal of data mining, as the key component of the

Figure 1: Knowledge Discovery in Databases (KDD) process

KDD process, is to extract previously unknown and useful patterns from the data using automatic or semi-automatic algorithms. Finally, the KDD process concludes with the presentation and evaluation of the detected patterns, enabling the user to understand and interpret the results.

In this work we focus on the development of novel models and algorithms for the central step of the KDD process: data mining. Out of the several mining tasks that exist in the literature, this work centers on the important method of *clustering*, which aims at grouping similar objects while separating dissimilar ones. Clustering, as an unsupervised learning task, analyses data without given labels but automatically reveals the hidden structure of the data by its aggregations. For today's data, however, it is known that traditional clustering methods fail to detect meaningful patterns. The problem originates from the fact that traditional clustering approaches consider the full space to measure the similarity between objects, i.e. all characteristics of the objects are taken into account. While collecting more and more characteristics, however, it is very unlikely that two objects are similar with respect to the full space and often some dimensions are not relevant for clustering. A continuative aspect is the decreasing discrimination power of distance functions with increasing dimensionality of the data space due to the "curse of dimensionality" [BGRS99]. The distances between objects grow more and more alike, thus all objects seem equally similar based on their attribute values. Since clusters are strongly obfuscated by irrelevant dimensions and distances are not discriminable any more, searches in the full space are futile or lead to very questionable clustering results.

Global dimensionality reduction techniques, e.g., based on Principal Component Analysis (PCA [Jol02]), try to mitigate these effects, but they do not provide a solution to this problem. Since they reduce all objects to a single projection, they cannot detect clusters with locally relevant dimensions. In complex data sets, however, different groups of objects may have different relevant dimensions. In Figure 2, the objects depicted as rectangles are similar in a 2-dimensional subspace, while the objects depicted as triangles show only similar values in a single dimension.

As a solution to this problem, the paradigm of *subspace clustering* [PHL04, KKZ09] has been introduced. Subspace clustering detects clusters in arbitrary subspace projections of the data by automatically determining for each group of objects a set of relevant dimensions these objects are similar in. Thus, in Figure 2 the objects grouped in cluster C_1 would correspond to a subspace cluster in subspace $\{$fast food consumption, sport activity$\}$, while the cluster C_2 is only located in subspace $\{$sport activity$\}$. Since different subspaces may lead to different groupings, each object can naturally belong to multiple clusters as illus-

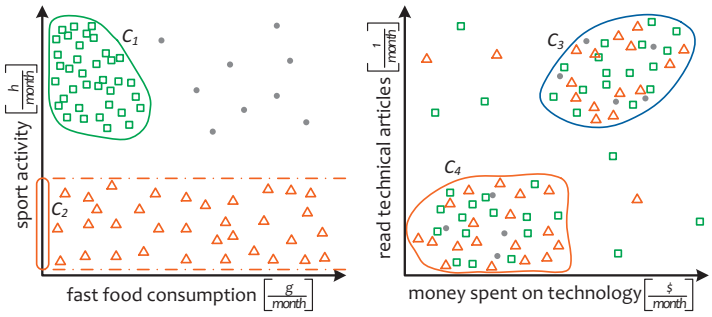


Figure 2: Exemplary subspace clustering of a 4-dimensional database

trated in Figure 2 (right). The subspaces individually assigned to each group provide the reasoning why such multiple solutions are meaningful. Thus, in the example of Figure 2, each of the four clusters $\{C_1, \ldots, C_4\}$ is useful and should be provided to the user.

In this work we describe novel methods for effective subspace clustering on complex data including high-dimensional vector spaces (Section 2), imperfect data (Section 3), and graph data (Section 4). Such clustering methods are beneficial for various applications: In customer and social network analysis, persons can be grouped according to their similarity based on some product relevant attributes. In bioinformatics, groups of genes that show similar expression levels in a subset of experimental medical treatments can be identified. In sensor network analysis, different environmental events can be described by similarly behaving sensors with respect to specific measured variables. For all of these domains objects are characterized by many attributes, while the clusters appear only in subspace projections of the data.

2 Subspace Clustering on Vector Data

In high-dimensional vector spaces, clusters rarely show up in the full dimensional space but are hidden in subspace projections of the data. Subspace clustering methods try to detect these patterns by analyzing arbitrary subspaces of the data for their clustering structure. In general, a subspace cluster $C = (O, S)$ is defined by a set of objects $O \subseteq DB$ that are similar in a subset of dimensions $S \subseteq Dim$.

Traditional subspace clustering approaches report clusters in any possible subspace projection. However, besides the high computational demand due to the exponential number of subspaces w.r.t. the number of dimensions that have to be analyzed, this principle generates results with a tremendous amount of redundant clusters [MGAS09]: often the objects grouped in a cluster $C = (O, S)$ are also similar in the subspace projections $S' \subseteq S$. In Figure 2 for example, the objects of the 2-dimensional subspace cluster C_1 are also similar in the 1-dimensional projections $\{$sport activity$\}$ and $\{$fast food consumption$\}$, resulting in already three clusters. Most of these groups, though, do not provide novel knowledge

about the data's structure. Even worse, such redundant information hinders an easy interpretation of the mining result. Consequently, traditional subspace clustering approaches fail to detect only the relevant subspace clusters.

To tackle the above challenge we propose novel subspace clustering methods avoiding redundant information in the final clustering result. In contrast to existing approaches that simply exclude lower dimensional projections of clusters, our methods perform an optimization of the final clustering to select the most interesting clusters. Furthermore, unlike to projected clustering methods, which avoid redundancy by enforcing disjoint clusters, our methods allow overlapping clusters in general.

2.1 Subspace Clustering using Combinatorial Optimization

In one line of research, we exploit the principle of combinatorial optimization to detect non-redundant subspace clustering results. The general idea can be described as follows: Assuming the set All of all possible subspace clusters according to a specific cluster definition is given (e.g. the set of clusters when applying DBSCAN [EKSX96] to any subspace projection). Since this set, however, contains highly redundant clusters, we are only interested in finding an *optimal*, *non-redundant* subset $M \subseteq All$ of clusters. To formally define the set M, we have to specify an appropriate *objective function* which should be minimized or maximized and necessary *constraints* that need to be fulfilled by M.

In our RESCU approach [MAG+09] we extend the Set Cover optimization problem to handle subspace clustering. The basic idea is that each cluster $C \in M$ of a non-redundant clustering $M \subseteq All$ needs to cover sufficiently many objects not contained in other clusters. That is, we do not include clusters whose grouped objects are already represented by the remaining clusters. To realize this aim, RESCU introduces the notion of cluster gain:

Definition 1 *Cluster gain*
Let $M = \{(O_1, S_1), \ldots, (O_n, S_n)\}$ be a clustering, $C = (O, S)$ a subspace cluster, and k a cost function for subspace clusters. The cluster gain of cluster C w.r.t. to M is:

$$clus\_gain(C, M) = \frac{|O \backslash Cov(M)|}{k(O, S)}$$

where $Cov(M) = \bigcup_{i=1}^{n} O_i$ are the objects covered by the clustering M.

The cost function k flexibly models the (un-)interestingness of clusters and can be specified by the user. For example, high-dimensional clusters are often be regarded as more interesting and therefore might get lower cost values. For a cluster to be included in the final result the cluster gain according to Definition 1 needs to be sufficiently high. Two important aspects contribute to this fact. First, the cluster covers many new objects, i.e. only few objects are already contained in other clusters. And second, the cost of the cluster is low, i.e. the cluster is interesting according to the user's rating. Based on the above definition, the optimal clustering M as specified in the RESCU model is defined as:

Definition 2 *Relevant subspace clustering (RESCU)*
Given the set All of all possible subspace clusters and a minimal cluster gain $\Delta \in \mathbb{R}^{\geq 0}$, $M \subseteq All$ is the optimal, non-redundant clustering if and only if

- *constraints:*
  - *M is redundancy-free, i.e. $\forall C \in M : clus\_gain(C, M \backslash \{C\}) > \Delta$*
  - *M is concept-covering, i.e. $\forall C \in All \backslash M : clus\_gain(C, M) \leq \Delta$*
- *objective: M minimizes the relative cost of the clustering, i.e. for all redundancy-free and concept-covering clusterings $N \subseteq All$ it holds*

$$\frac{1}{|Cov(M)|} \sum_{(O_i,S_i) \in M} k(O_i, S_i) \leq \frac{1}{|Cov(N)|} \sum_{(O'_i,S'_i) \in N} k(O'_i, S'_i)$$

The above *constraints* ensure that the optimal clustering M contains all but only non-redundant clusters. By minimizing the *objective function*, the best clustering according to the selected interesting criterion is chosen. Overall, based on this combinatorial optimization problem a small set of interesting and non-redundant clusters is determined.

In [MAG+09] we prove that the computation of the RESCU model is NP-hard, and we propose an algorithm determining an approximate solution showing high clustering accuracy. Thorough experiments demonstrate that RESCU reliably outperforms existing subspace and projected clustering algorithms. Figure 3 shows the clustering quality (computed via the F1 and accuracy measure [GFM+11]) for six real world data sets [FA10, AKMS08]. In addition to the absolute values we note the relative quality compared to the best measurement on each data set. Best 95% results are highlighted in gray. RESCU achieves top quality results for *all* data sets with respect to *both* quality measures. The competing approaches show highly varying performance. None of them achieves top quality allover. Although some of the approaches achieve slightly better results on some of the data sets, RESCU reliably shows top results on all data sets.

Detecting Multiple Clustering Views. Eliminating redundancy from subspace clustering results has to be regarded carefully: overlapping clusters are not necessarily a sufficient criterion for redundancy. Since different subspaces represent different views on the data, objects are allowed to be contained in several clusters without inducing redundancy (cf. Figure 2). The subspace clusters of each view provide novel information about the data's characteristic, and their grouping into views enables further interpretations about the clusters' interrelations. These aspects are considered in our OSCLU model [GMFS09]. In the OSCLU model we propose a global optimization of the overall clustering exploiting the clusters' similarities regarding their sets of objects as well as their subspace projections. The combinatorial optimization method of OSCLU actively includes novel knowledge of (almost) orthogonal subspaces into the final clustering result. Therefore, it overcomes major limitations of existing approaches in the detection of multiple views. The formal definition of the combinatorial optimization performed by OSCLU can be found in [GMFS09].

While the OSCLU model provides a general and flexible solution to detect subspace clusters hidden in multiple views, we prove its complexity to be NP-hard and propose an efficient algorithm to compute an approximate solution. We approximate the optimization

	Glass (214; 9)				Vowel (990; 10)				Diabetes (768; 8)			
	F1		Accuracy		F1		Accuracy		F1		Accuracy	
RESCU	60	100%	62	100%	44	100%	64	96%	71	100%	69	100%
INSCY	56	93%	54	87%	37	84%	67	100%	58	82%	65	94%
FIRES	30	50%	49	79%	10	23%	12	18%	33	46%	65	94%
SCHISM	45	75%	49	79%	24	55%	53	79%	69	97%	69	100%
PROCLUS	39	65%	54	87%	32	73%	30	45%	44	62%	65	94%
P3C	17	28%	39	63%	8	18%	16	24%	44	62%	65	94%
STATPC	19	32%	47	76%	17	39%	47	70%	39	55%	64	93%

	Shape (160; 17)				Liver-Dis. (345; 6)				Breast (198; 33)			
	F1		Accuracy		F1		Accuracy		F1		Accuracy	
RESCU	60	100%	75	100%	62	97%	61	98%	67	100%	76	97%
INSCY	56	93%	61	81%	62	97%	59	95%	65	97%	70	90%
FIRES	56	93%	62	83%	50	78%	53	85%	46	69%	75	96%
SCHISM	38	63%	59	79%	64	100%	58	94%	65	97%	71	91%
PROCLUS	60	100%	62	83%	46	72%	62	100%	47	70%	77	99%
P3C	39	65%	45	60%	36	56%	58	94%	63	94%	77	99%
STATPC	31	52%	62	83%	57	89%	58	94%	41	61%	78	100%

Figure 3: Quality (F1 & accuracy) on real world data. Captions: data set (size; dimensionality)

problem by pruning similar subspaces ensuring efficient cluster detection since only orthogonal subspaces are analyzed. Overall, our OSCLU approach is the first method for detecting multiple clustering views in subspaces of high dimensional data.

2.2 Subspace Clustering using Bayesian Generative Models

Besides combinatorial optimization, we analyzed a second line of research for multi-view subspace clustering: in [GFS12] we propose a method exploiting the principle of Bayesian generative models. We extend the established concept of mixture models to handle data containing multiple clustering views. Our MVGen model represents the data's multiple views by different subspace projections, thus, avoiding the problem of full-space clustering. Each view describes an individual partitioning of the objects. Accordingly, our model is able to represent multiple groupings and it simultaneously prevents redundant information since highly overlapping clusters in similar subspace projections are avoided. Figure 4 shows an exemplary result as reported by our method: In the example, two different clustering views are detected. The first grouping is found in the subspace $\{1, 2\}$, while a second and completely different grouping is detected in subspace $\{3, 4\}$. For each of these views an individual mixture distribution is fitted to the data. Please note that our method automatically detects the groupings as well as the dimensions supporting this grouping. Additionally, our method allows each mixture component to be located in an individual subspace. For example, as shown in Figure 4 left, the mixture component in the back is noisy in dimension 1, while the component in the front is located in both dimensions.

In [GFS12], we formally introduce the generative process that models data containing multiple views. Since views and clusters are located in subspace projections, we distinguish between relevant and irrelevant dimensions. Thus, unlike to traditional mixture model, in our model we have to tackle the challenge of model selection. In our method, we use

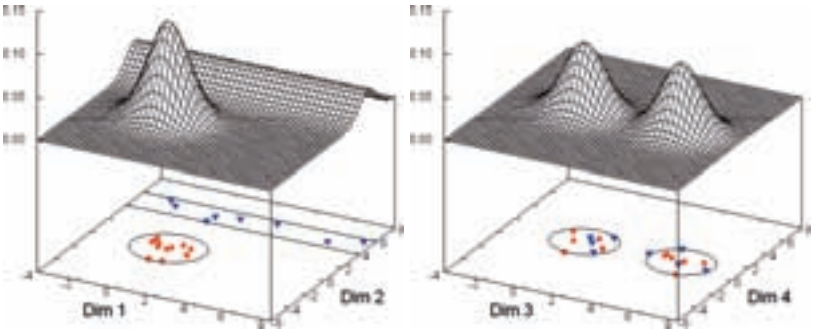


Figure 4: Mixture models located in subspace projections

Bayesian model selection to decide which sets of dimensions are relevant for the clusters and views. For an efficient learning, we exploit the principle of iterated conditional modes and we derived the required update equations.

The comparison of MVGen with competing approaches demonstrated the strengths of detecting views in multiple subspace projections. In the following we exemplarily show the results of MVGen on two datasets. In the first experiment we analyze the clustering result on the CMUFace data. This data is interesting for multi-view clustering since it consists of images taken from persons showing varying characteristics as their facial expressions (neutral, happy, sad, angry), head positions (left, right, straight), and eye states (open, sunglasses). We randomly select 3 persons with all their images and applied PCA retaining at least 90% of the data's variance as a pre-processing step. The clustering result of MVGen for two views each with three clusters is illustrated in Figure 5. The images correspond to the means of each detected cluster. By visual inspection, we can easily find the reason for detecting these two views: The first view, describes the grouping based on the different persons, while the second view, corresponds to a grouping based on their head positions.

In the second experiment, we perform image segmentation on Escher images, which are known to have multiple interpretations to the human eye. For clustering, each pixel is

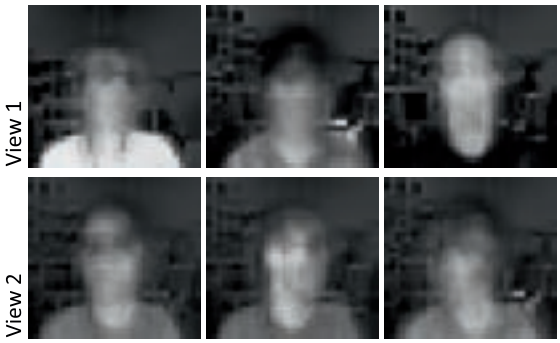
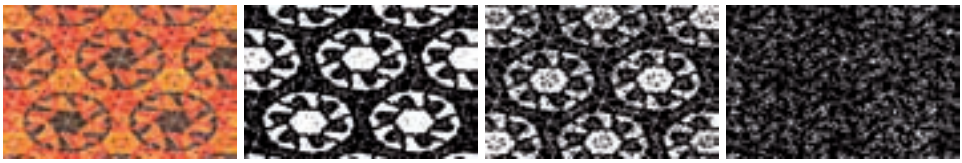


Figure 5: MVGen on face data



Original image View 1 View 2 View 3

Figure 6: Result of MVGen on an Escher image

regarded as an object with RGB and HSV values as features. In Figure 6 (left), such an image is depicted (followed by the three views detected by MVGen). Focusing on the dark regions, there is a segmentation of the image as given by the first view of MVGen. This segmentation is dominant since the dark parts clearly deviate from the orange/yellow parts. However, MVGen is also able to discover the more subtle view where the yellow parts are decoupled from the others. Most interesting is the third view detected by MVGen: it corresponds to only the background of the image.

Overall, MVGen successfully detects the multi-view clustering structure on a variety of data sets. Especially the explicit modeling of the views' relevant subspaces has proven to be very valuable for interpreting the final clustering results.

2.3 Subspace Correlation Clustering

While the previous methods focus on subspace clusters corresponding to dense areas in the data space, we introduced in [GFVS12] the novel paradigm of subspace correlation clustering: we analyze *subspace projections* to find *subsets of objects* showing *linear correlations* among this subset of dimensions. While existing correlation clustering methods are limited to almost disjoint clusters, our model allows each object to contribute to several correlations due to different subspace projections. For example, by considering the 2-dimensional subspace $\{d1, d2\}$ in Figure 7, two different (local) correlations can be detected: The objects indicated by a cross are positively correlated on a line, while the objects indicated by a circle are negatively correlated on a different line. Considering the subspace $\{d3, d4\}$, two different correlations supported by different sets of objects can be detected. Thus, objects may contribute to several correlations due to different subspace projections. In our paradigm, we permit multiple overlapping clusters but simultaneously avoid redundant clusters deducible from already known correlations originating from collinearity or induction. More details about this work can be found in [GFVS12].

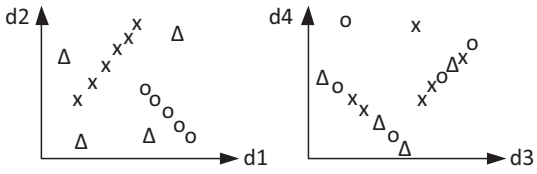


Figure 7: 4-d database with 15 objects and 4 subspace correlation clusters

3 Subspace Clustering on Imperfect Data

Most subspace clustering algorithms assume perfect data as input. Imperfect information, however, is ubiquitous where data is recorded: Missing values, for example, occur due to sensor faults in sensor networks, or uncertainty about attribute values is present due to noise or privacy issues. There is a need to handle such imperfect data for the task of subspace clustering.

Naively, traditional data cleansing techniques could be applied to preprocess the data before clustering. Data cleansing, however, has several limitations. First, data cleansing is accompanied by high cost since the methods are rarely completely automatic but the user has to be involved. Second, the storage overhead can be huge since besides the original data also the preprocessed data have to be stored. And last, preprocessing the data usually results in an information loss. On the one hand the preprocessing step is not aware of the special characteristics of the subsequent subspace clustering task as, e.g., the occurrence of objects in multiple clusters due to different subspace projections. On the other hand the mining method cannot distinguish between a precise object and an imperfect but cleaned object. Overall, valuable information is no longer available due to preprocessing.

Consequently, we propose integrated mining methods that direct handle imperfect data for the task of subspace clustering as illustrated in Figure 8. By joining the preprocessing step with the actual mining task, we are able to account for the special characteristics of subspace clustering leading to a better handling of imperfect information. Instead of mining the preprocessed data, the mining method directly analyzes the raw data and, e.g., instantiates missing values based on the currently detected subspace clusters.

Directly operating on imperfect data leads to novel requirements for subspace clustering models and definitions ranging from the accurate determination of similarity values between individual objects to the overall coherence of a cluster in an imperfect setting. The underlying challenge to be tackled by our models is their robustness against 'errors' in the data. Even for a high-degree of imperfect information, reliably detecting high quality patterns should be possible. In our work, we describe two scenarios: our first method provides a solution for the case of data containing incomplete information and our second method covers the topic of attribute uncertainty.

Figure 8: Enhanced KDD process by integrating the preprocessing step into the mining step for better handling of imperfect data

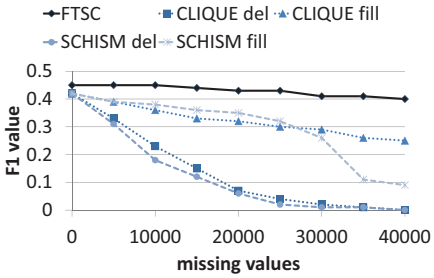


Figure 9: Clustering quality on pendigits

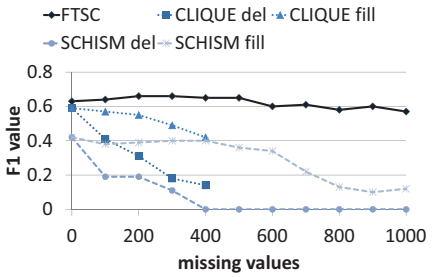


Figure 10: Clustering quality on shape

3.1 Subspace Clustering for Incomplete Data

Incompleteness describes imperfect information due to the absence of individual measurements. While the absence of specific information regarding a single object is denoted as existential incompleteness, the absence of objects as a whole is denoted as universal. In our work we tackle the challenges of existential incompleteness. Incomplete information, often referred to as data containing *missing values*, occurs for example due to faulty sensors or incomplete customer questionnaires.

In [GMRS11] we introduce a general fault tolerance definition enhancing subspace clustering models to handle missing values. Our model handles missing values based on the currently considered subspace and set of objects. Intuitively, missing values should be tolerated within a subspace cluster when the remaining objects still provide sufficient information about the relevant dimensions and the object groupings. Since a meaningful fault tolerance has to consider the varying object and attribute characteristics for each subspace cluster individually, we introduce a fault tolerance notion that adapts to the characteristics of subspace clusters. We abstract from concrete subspace clustering definitions and propose a general fault tolerance principle applicable to multiple instantiations. Thus, grid-based subspace clustering approaches as CLIQUE [AGGR98], paradigms based on the density-based clustering idea [KKK04], and several other definitions can benefit from our approach. In addition to our general model, we present a concrete instantiation – the algorithm FTSC – to the well-established grid-based subspace clustering.

As there are no competing subspace clustering approaches that handle missing values, we compare FTSC with methods working on (complete) data, cleaned by statistical preprocessing techniques. In one case we use complete case analysis and in the second case mean value imputation. To ensure a fair comparison, we apply the grid-based clustering methods CLIQUE [AGGR98] and SCHISM [SZ04] on these data since FTSC is also grid-based. In the experiments depicted in Figure 9 & 10 we analyze the methods' clustering quality on the real world datasets *pendigits* and *shape*. We increase the number of randomly distributed missing values to analyze the methods' robustness to faults in the data. For both datasets the following observations become apparent: By adding 0 missing values, the qualities of all approaches are nearly identical. The small differences can be explained by slightly different clustering definitions. Our FTSC achieves the highest

clustering qualities and shows robust behavior with increasing number of missing values. Even for a huge amount of missing values the quality is high and only for some datasets a small decrease can be observed. The methods based on pre-processing show a stronger decrease of their clustering qualities. Especially, the deletion methods (CLIQUE/SCHISM del) are consistently worse than the methods based on filling up missing values by mean value imputation (CLIQUE/SCHISM fill). Summarized, our FTSC gets highest clustering qualities even if the data is polluted by a huge amount of missing values.

3.2 Subspace Clustering for Uncertain Data

In many scenarios uncertainty about the given information does exist. In the case of uncertainty, one is just provided with an estimate how likely the observed value is equal to (or may differ from) the true value. For example, the measured GPS signal of a mobile phone is highly uncertain information for determining its true position and one is only provided with an estimate about this position by, e.g., incorporating probability distributions. Similar to incomplete information, one distinguishes uncertainty about specific attributes – so called attribute uncertainty – and uncertainty about the existence of whole objects – tuple uncertainty. We consider the case of attribute uncertainty. Besides uncertainty due to the data recording step, artificial uncertainty due to privacy issues is present, i.e. before providing a data set sensitive information is obfuscated.

Data mining on uncertain databases is critical since attribute values with a large error are less reliable for data mining purposes than ones with small errors. Our novel subspace clustering method [GKS10] considers these aspects to ensure robust clustering results. Since often uncertain objects are represented by probability density functions (*pdfs*), our subspace clustering methods analyses data objects modeled as (multi-dimensional) probability density functions. Additionally, since subspace clustering tackles the challenge of clustering objects in projections of the data space, our method has to consider for each *pdf* multiple subspace projections:

Definition 3 *Projection of an uncertain object*
Given an uncertain object i represented by the pdf p_i and a subspace $S \subseteq Dim = \{1, \ldots, d\}$, the projection of p_i to S is the marginal distribution *of p_i for S. The obtained pdf is denoted as*

$$p_i^S(x) \text{ with } x \in \mathbb{R}^{|S|}$$

For example and w.l.o.g. $S = \{1, \ldots, s\}$, then

$$p_i^S(x) = p_i^S(x_1, \ldots, x_s) = \int_{x_{s+1}, \ldots, x_d \in \mathbb{R}} \cdots \int p_i(x_1, \ldots, x_d)$$

i.e., we marginalize over the dimensions $\{s+1, \ldots, d\}$.

In our method we exploit the principle of grid-based subspace clustering [PJAM02]. Established for (certain) vector data, this principle groups objects into the same cluster if their distance to each other in a specific subspace is sufficiently small. Since our method has

to cope with uncertain objects represented by *pdf*s, we do not calculate an actual distance value but we calculate the probability that two objects are near to each other. Formally, the probability that the distance between two independent objects i and j (represented by the *pdf*s p_i and p_j) in a subspace S is smaller than a maximal distance w is

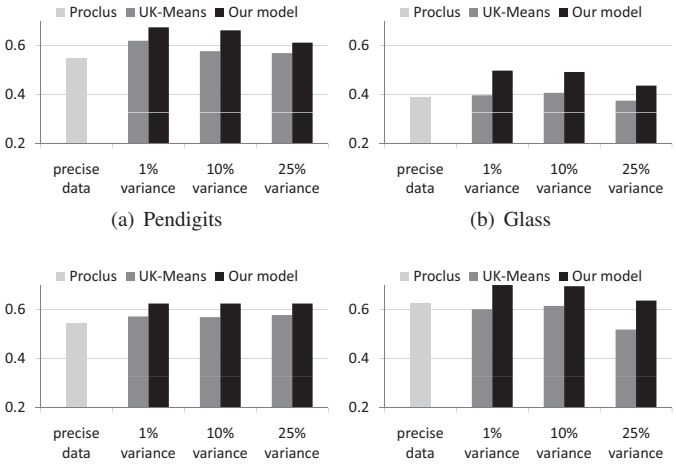
$$P_{\leq w}(p_i, p_j, S) = \int_{\substack{x,y \in \mathbb{R}^{|S|} \\ d_\infty^S(x,y) \leq w}} p_i^S(x) \cdot p_j^S(y)\, dx\, dy \tag{1}$$

We have to integrate over all possible vectors whose distance to each other is small enough and multiply the corresponding densities of the underlying *pdfs*.

Please do not confuse this probability with the values computed when considering, for example, mixture models. In mixture models, we compute for each object the likelihood of belonging to the cluster, i.e. we evaluate the density of a *single pdf* (the cluster's component) at a *given* realization (the observed data point). Here, we compute the probability that *any two* realizations of the *two given pdf*s are close to each other.

A subspace clusters in subspace S can finally be detected by randomly selecting an uncertain object m and determining all objects i whose probability for the event $P_{\leq w}(p_m, p_i, S)$ is high enough. Since in an uncertain setting each object naturally might belong to multiple clusters with different probabilities, partitioning clustering approaches are obviously out of place. Therefore, we additionally introduce a new non-partitioning clustering method by augmenting the clusters with membership degrees of their assigned objects. This improves the quality of clusterings and enables users to extract more useful information. Since our proposed model is computationally expensive, we introduce an efficient solution that uses Apriori-based pruning and heuristic sampling while still providing high quality results.

The performance of our model on real world data is analyzed in Figure 11. We present the results for the 4 datasets pendigits, glass, breast cancer, and shape. Because there exist no



(a) Pendigits (b) Glass

(c) Breast cancer (d) Shape

Figure 11: Clustering quality on real world data

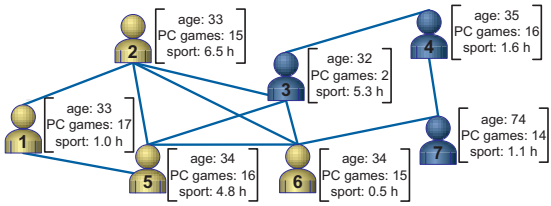


Figure 12: Exemplary social network represented by vector and graph data; highlighted in yellow: one potential twofold cluster with two relevant dimensions

direct competitors in the domain of subspace clustering for uncertain data, we compare our approach with UK-Means [CCKN06] and Proclus [AWY+99]. UK-Means is chosen as a representative for fullspace clustering on *uncertain* data while Proclus identifies *subspace* clusters on certain data. Proclus is executed on the original precise data. Our model and UK-Means use the uncertain variants of the data; the variance of the underlying Gaussian distributions is set to 1%, 10%, and 25% of the data range.

The results on the pendigits dataset are presented in Figure 11(a). We can see that our model outperforms the competing algorithms. Interestingly, the results of Proclus, operated on precise data, are worse than the results of the approaches that have to cope with uncertain information. For higher variances, however, we can see a decrease in quality; the clustering structure is obfuscated by the high variance and hence a detection of clusters is difficult. On the remaining datasets similar results are obtained. Only the shape dataset (Figure 11(d)) is an exception: the quality of Proclus is slightly better than the quality of UK-Means. Nevertheless, for every dataset the effectiveness of our model is higher compared to the competing methods.

4 Subspace Clustering on Graphs with Feature Vectors

Traditional data mining algorithms process just a single type of data; e.g., objects embedded into a vector space. Today's applications, however, can acquire multiple, diverse, and heterogeneous data sources. Besides characterizing single objects by vector data, network information, for example, is a ubiquitous source to indicate the relations between different objects. Such type of heterogeneous data can be observed in various domains including social networks, where friendship relationships are available along with the users' individual interests (cf. Figure 12); systems biology, where interacting genes and their specific expression levels are recorded; and sensor networks, where connections between the sensors as well as individual measurements are given. To realize the full potential for knowledge extraction, mining techniques should consider all available information sources.

A sequential process for heterogeneous data, which first mines each type independently and then compares the detected patterns, is problematic since the results of each source might differ or even contradict. Thus, for an effective clustering, again an integrated mining promises more meaningful and accurate results. By simultaneously mining different

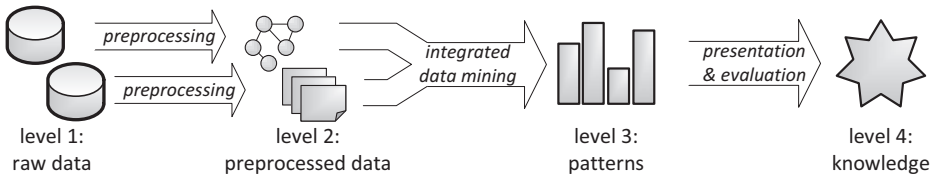


Figure 13: Enhanced KDD process by simultaneously mining multiple information types for better handling of heterogeneous data

types of information, as illustrated in the adapted KDD process of Figure 13, inaccurate information in one source can be mitigated by the other sources and an overall coherent result is possible.

In the past years, multiple integrated approaches for clustering graphs with feature vectors have been introduced. The main problem of almost all these approaches, however, is the consideration of *all* attribute dimensions for determining the similarity. As known from the previous sections, some dimensions, however, might not be relevant, which is why clusters are located in subsets of the dimensions. E.g. in social networks, it is very unlikely that people are similar within all of their characteristics. Since this aspect is not adequately considered by the existing models, we propose novel methods joining the mining task of subspace clustering and graph mining.

4.1 A Synthesis of Subspace Clustering and Dense Subgraph Mining

The GAMER approach [GFBS10] combines graph data and attribute data to identify groups according to their density of connections as well as similarity of attribute values in subsets of the dimensions. In Fig. 12 for example we are able to identify the cluster $\{1, 2, 5, 6\}$ because the objects are similar in 2 attributes and the density of the subgraph is high. A clustering procedure like this is advantageous for a variety of applications: Besides the already mentioned example of gene analysis, highly connected groups of people in social networks (graph density) can be used for target and viral marketing based on their specific preferences (attribute subset). In sensor networks, an aggregated transmission of specific sensor measurements (attribute subset) of communicating sensors (graph density) leads to improved energy efficiency and, thus, to longer lifetime of the network.

A sound combination of the paradigms *subspace clustering* and *dense subgraph mining* has to be unbiased in the sense that none of the paradigms is preferred over the other. Most integrated clustering models focus on graph properties as determining maximal sets whose density is large enough. In Fig. 12 for example the largest clique (a certain type of dense subgraphs) is $\{2, 3, 5, 6\}$; however, the vertices of this clique show similar behavior only in one of their three attributes. Even worse, preferring just high dimensional clusters leads to $\{1, 4, 6\}$; this cluster cannot be reconciled with the graph structure. Obviously the cluster properties 'density/connectedness', 'dimensionality', and 'size' are usually contradictory and a clustering model has to realize a reasonable trade-off. The challenge tackled by

our approach is the optimization of all three goals simultaneously to ensure their equal consideration. This enables each paradigm to be on a par with the other one in order to obtain meaningful and consistent clusters. Vertex group $\{1, 2, 5, 6\}$ and vertex group $\{2, 3, 5\}$ could be possible clusters for such an optimization. In both clusters all vertices have similar values in 2 attributes, and the density of the subgraphs is negligibly smaller than in cliques.

A further important observation is that overlaps between clusters are quite reasonable. While the cluster $\{1, 2, 5, 6\}$ might be of interest for video game producers, the cluster $\{2, 3, 5\}$ might be of interest for sports wear retailers. Persons thus can be assigned to more than one product target group. Also for the application of gene interaction networks and sensor networks it holds that genes can belong to more than one functional module and sensors to more than one aggregation unit. Highly overlapping clusters, however, often imply nearly the same interpretations and, thus, a strong overlap usually indicates redundancy. As shown in the previous sections of this work, considering redundancy is indispensable for subspace clustering methods. Also in the field of graph mining, avoiding redundant patterns is studied [HCS+07]. The importance of a proper treatment of redundancy is hence increased for the combined consideration of subspace clustering and subgraph mining albeit rarely treated accurately in the past. Our model successfully avoids redundancy in the clustering result, while generally allowing the clusters to overlap.

Formally, the input for our model is a vertex-labeled graph $G = (V, E, l)$ with vertices V, edges $E \subseteq V \times V$ and a labeling function $l : V \to \mathbb{R}^d$ where $Dim = \{1, \ldots, d\}$ is the set of dimensions. As an abbreviation we use $l(O) = \{l(o) \mid o \in O\}$ to denote the set of vectors associated to the set of vertices $O \subseteq V$ and $x[i]$ to refer to the i-th component of a vector $x \in \mathbb{R}^d$.

The clusters detected in GAMER should represent meaningful subspace clusters and at the same time meaningful dense subgraphs. To achieve this aim, the notion of twofold clusters is introduced:

Definition 4 *Twofold cluster*
Given a graph $G = (V, E, l)$, a twofold cluster $C = (O, S)$ with respect to the thresholds $s_{min}, \gamma_{min}, n_{min}$ is a set of vertices $O \subseteq V$ and a set of dimensions $S \subseteq Dim$ with the following properties

- $(l(O), S)$ *fulfills the subspace cluster property, i.e.*

$$\forall d \in S : \forall x, y \in l(O) : |x[d] - y[d]| \leq w$$
$$\forall d \in Dim \backslash S : \exists x, y \in l(O) : |x[d] - y[d]| > w$$

- *O fulfills the quasi-clique property, i.e.*

$$\min_{v \in O} \{deg^O(v)\} \geq \lceil \gamma_{min} \cdot (|O| - 1) \rceil$$

  where $deg^O(v)$ is the degree of vertex v within vertex set O

- *the induced subgraph of O is connected, $|O| \geq n_{min}$, and $|S| \geq s_{min}$*

With the beforehand introduced definition we are able to determine the set of all valid twofold clusters $Clusters$. Without any constraints this set can be large and would represent many redundant clusters. Similar to Section 2.1 we are interested in finding a non-redundant subset $Result \subseteq Clusters$ of the most interesting clusters. The interestingness of individual clusters is evaluated in GAMER via a quality function $Q(C)$. It incorporates the density, size and dimensionality of a cluster and, thus, realizes a sound and unbiased synthesis of subspace clustering and subgraph mining.

The quality function is important to identify the redundant clusters. A cluster C can only be redundant compared to a cluster C' if C's quality is lower. If the cluster C had a higher quality, then it should not be reported as redundant w.r.t. C'; the user is more interested in C. Thus, $Q(C) < Q(C')$ must hold for the redundancy of C w.r.t. C'. Furthermore, the cluster C induces redundancy w.r.t. C' if it does not describe novel structural information. In our context, this aspect means that the objects as well as the relevant dimensions of $C = (O, S)$ have already been covered to most parts by the cluster $C' = (O', S')$. If the fraction $\frac{|O \cap O'|}{|O|}$ is large, only a small percentage of C's objects are not contained in C'; we do not have a large information gain based on the object grouping of C. The same holds for the set of relevant dimensions. If all three conditions hold, the cluster C is redundant w.r.t. C'. We denote this by $C \prec_{red} C'$ and we formally define:

Definition 5 *Binary redundancy relation*
Given the redundancy parameters $r_{obj}, r_{dim} \in [0, 1]$, the binary redundancy relation $\prec_{red}$ is defined by:

$$\text{For all twofold clusters } C = (O, S), C' = (O', S'):$$
$$C \prec_{red} C' \Leftrightarrow \left[Q(C) < Q(C') \quad \wedge \quad \frac{|O \cap O'|}{|O|} \geq r_{obj} \quad \wedge \quad \frac{|S \cap S'|}{|S|} \geq r_{dim}\right]$$

Based on this relation, the optimal clustering can be defined as follows:

Definition 6 *Optimal twofold clustering*
Given the set of all twofold clusters $Clusters$, the optimal twofold clustering $Result \subseteq Clusters$ fulfills

- *redundancy-free property:* $\neg\exists C_i, C_j \in Result : C_i \prec_{red} C_j$
- *maximality property:* $\forall C_i \in Clusters \backslash Result : \exists C_j \in Result : C_i \prec_{red} C_j$

As in Section 2.1 we perform a combinatorial optimization to detect the non-redundant clustering result. Please note, though, that the above definition introduces *constraints* only and does not specify an *objective function* to be minimized/maximized. As shown in [GFBS10], the clustering fulfilling the above constraints is unique. Thus, any objective function would lead to the same result. Overall, also for the synthesis of subspace clustering with dense subgraph mining, a combinatorial optimization method can be used to find a non-redundant clustering solution.

Figure 14 shows the experimental results on a dataset comprising gene expressions and gene interactions [S$^+$06, S$^+$05]. The data contains 3548 vertices, 8334 edges, and 115 dimensions. For evaluating the clustering quality we use the Go-Miner tool that assigns

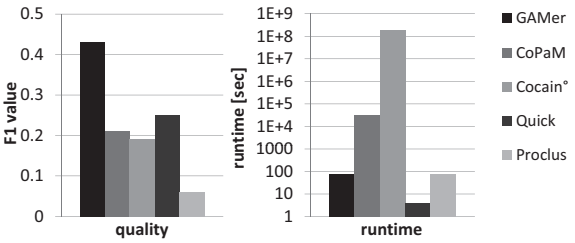


Figure 14: Clustering quality and runtime on gene data

genes to biological categories. These classes are used as hidden clusters as also done in [MCRE09]. For the experiment in Fig. 14, GAMER obtains the highest quality results. CoPaM [MCRE09] and Cocain° [ZWZK06] are not able to detect meaningful clusters. Furthermore, we calculate for this experiment the results of approaches that consider only one data source, i.e. subgraph mining (maximal quasi cliques, Quick [LW08]) or subspace clustering (Proclus [AWY+99]). The quality of these two algorithms is low, indicating that a synthesis of subspace clustering and dense subgraph mining can effectively increase the clustering quality. Considering the runtime, we see that our approach is more than 100 times faster than CoPaM and even better compared to Cocain°.

Extending the GAMER method, we propose in [GBFS13] our EDCAR model. We prove the model's complexity and identify the critical parts inhibiting an efficient execution. Based on this analysis, we develop an efficient and effective algorithm that approximates the optimal clustering solution. By interweaving the process of cluster generation and cluster selection, which both make use of the GRASP (Greedy Randomized Adaptive Search Procedure) principle, we determine high quality clusters and ensure low runtimes. Figure 15 shows that EDCAR is orders of magnitudes faster than all competing approaches.

4.2 Density-Based Subspace Clustering for Graphs with Feature Vectors

The previously proposed approaches GAMER and EDCAR successfully overcome the problems of full-space clustering when analyzing graphs with feature vectors. Though,

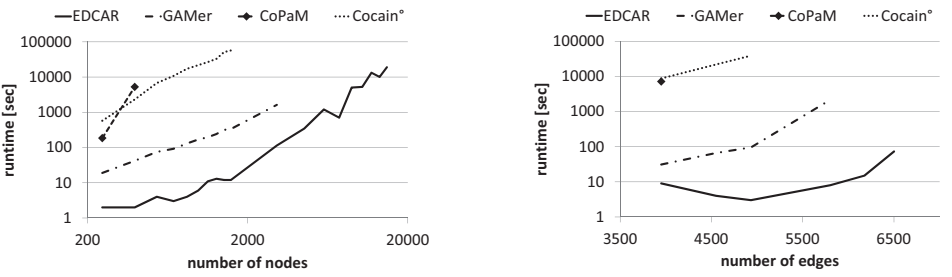


Figure 15: Scalability w.r.t. number of nodes and number of edges

the twofold cluster definition (cf. Def. 4) is restricted to clusters of certain shapes. Similar to grid-based subspace clustering [PJAM02], a cluster (w.r.t. the attributes) is defined by taking all objects located within a given grid cell, i.e. whose attribute values differ by at most a given threshold w. The methods are biased towards small clusters with little extend. This drawback is worsened by considering the used notions of dense subgraphs: e.g. by using quasi-cliques the diameter is a priori constrained to a fixed threshold [PJZ05]. For real world data, such a cluster definition might be too restrictive since clusters can exhibit more complex shapes.

In our DB-CSC model [GBS11, GBS12], we combine dense subgraph mining with subspace clustering based on a more sophisticated cluster definition; thus solving the drawbacks of previous approaches. Established for other data types, density-based clustering techniques [EKSX96, SEKX98] have shown their strength in many scenarios. They do not require the number of clusters as an input parameter and are able to find arbitrarily shaped clusters. We introduce the first approach exploiting a density-based clustering principle to join the paradigms of subspace clustering and dense subgraph mining. Our clusters correspond to dense regions in the attribute space as well as in the graph. Based on the novel notion of local densities, our DB-CSC model uses a fixed point iteration to find the desired clusters. Further pruning techniques, allow an efficient calculation of the overall clustering solution. In thorough experiments we demonstrate the strength of DB-CSC in comparison to related approaches. A more detailed discussion can be found in [GBS11, GBS12].

5 Conclusion

In this work, we proposed novel models for an effective subspace clustering of complex data. We analyzed three different types of data: vector data, imperfect data, and network data in combination with vector data. For each of these different data sources, we introduced enhanced mining models and efficient algorithms. In thorough experiments, we demonstrated the strengths of our novel clustering approaches. Overall, for the first time, meaningful subspace clustering results can be obtained for these types of complex data.

References

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 94–105, 1998.

[AKMS08] I. Assent, R. Krieger, E. Müller, and T. Seidl. EDSC: Efficient Density-Based Subspace Clustering. In *ACM Conference on Information and Knowledge Management (CIKM)*, pages 1093–1102, 2008.

[AWY+99] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park. Fast algorithms for projected clustering. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 61–72, 1999.

[BGRS99] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When Is "Nearest Neighbor" Meaningful? In *International Conference on Database Theory (ICDT)*, pages 217–235, 1999.

[CCKN06] M. Chau, R. Cheng, B. Kao, and J. Ng. Uncertain Data Mining: An Example in Clustering Location Data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 199–204, 2006.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, 1996.

[FA10] A. Frank and A. Asuncion. UCI Machine Learning Repository, http://archive.ics.uci.edu/ml, 2010.

[GBFS13] S. Günnemann, B. Boden, I. Färber, and T. Seidl. Efficient Mining of Combined Subspace and Subgraph Clusters in Graphs with Feature Vectors. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2013.

[GBS11] S. Günnemann, B. Boden, and T. Seidl. DB-CSC: A density-based approach for subspace clustering in graphs with feature vectors. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 565–580, 2011.

[GBS12] S. Günnemann, B. Boden, and T. Seidl. Finding density-based subspace clusters in graphs with feature vectors. *Data Mining and Knowledge Discovery Journal (DMKD)*, 25(2):243–269, 2012.

[GFBS10] S. Günnemann, I. Färber, B. Boden, and T. Seidl. Subspace Clustering Meets Dense Subgraph Mining: A Synthesis of Two Paradigms. In *IEEE International Conference on Data Mining (ICDM)*, pages 845–850, 2010.

[GFM+11] S. Günnemann, I. Färber, E. Müller, I. Assent, and T. Seidl. External Evaluation Measures for Subspace Clustering. In *ACM Conference on Information and Knowledge Management (CIKM)*, pages 1363–1372, 2011.

[GFS12] S. Günnemann, I. Färber, and T. Seidl. Multi-View Clustering Using Mixture Models in Subspace Projections. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 132–140, 2012.

[GFVS12] S. Günnemann, I. Färber, K. Virochsiri, and T. Seidl. Subspace Correlation Clustering: Finding Locally Correlated Dimensions in Subspace Projections of the Data. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 352–360, 2012.

[GKS10] S. Günnemann, H. Kremer, and T. Seidl. Subspace Clustering for Uncertain Data. In *SIAM International Conference on Data Mining (SDM)*, pages 385–396, 2010.

[GMFS09] S. Günnemann, E. Müller, I. Färber, and T. Seidl. Detection of orthogonal concepts in subspaces of high dimensional data. In *ACM Conference on Information and Knowledge Management (CIKM)*, pages 1317–1326, 2009.

[GMRS11] S. Günnemann, E. Müller, S. Raubach, and T. Seidl. Flexible Fault Tolerant Subspace Clustering for Data with Missing Values. In *IEEE International Conference on Data Mining (ICDM)*, pages 231–240, 2011.

[HCS+07] M. A. Hasan, V. Chaoji, S. Salem, J. Besson, and M. J. Zaki. Origami: Mining representative orthogonal graph patterns. In *IEEE International Conference on Data Mining (ICDM)*, pages 153–162, 2007.

[HK01] J. Han and M. Kamber. *Data mining: Concepts and techniques*. Morgan Kaufmann, 2001.

[Jol02] I. T. Jolliffe. *Principal Component Analysis*. Springer, 2nd edition, 2002.

[KKK04] K. Kailing, H.-P. Kriegel, and P. Kröger. Density-Connected Subspace Clustering for High-Dimensional Data. In *SIAM International Conference on Data Mining (SDM)*, pages 246–257, 2004.

[KKZ09] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1):1–58, 2009.

[LW08] G. Liu and L. Wong. Effective Pruning Techniques for Mining Quasi-Cliques. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 33–49, 2008.

[MAG+09] E. Müller, I. Assent, S. Günnemann, R. Krieger, and T. Seidl. Relevant Subspace Clustering: Mining the Most Interesting Non-redundant Concepts in High Dimensional Data. In *IEEE International Conference on Data Mining (ICDM)*, pages 377–386, 2009.

[MCRE09] F. Moser, R. Colak, A. Rafiey, and M. Ester. Mining Cohesive Patterns from Graphs with Feature Vectors. In *SIAM International Conference on Data Mining (SDM)*, pages 593–604, 2009.

[MGAS09] E. Müller, S. Günnemann, I. Assent, and T. Seidl. Evaluating Clustering in Subspace Projections of High Dimensional Data. *PVLDB*, 2(1):1270–1281, 2009.

[PHL04] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: a review. *SIGKDD Explorations*, 6(1):90–105, 2004.

[PJAM02] C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. M. Murali. A Monte Carlo algorithm for fast projective clustering. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 418–427, 2002.

[PJZ05] J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 228–238, 2005.

[S+05] R. Shyamsundar et al. A DNA microarray survey of gene expression in normal human tissues. *Genome Biology*, 6, 2005.

[S+06] C. Stark et al. BioGRID: a general repository for interaction datasets. *Nucleic acids research*, 34, 2006.

[SEKX98] J. Sander, M. Ester, H.-P. Kriegel, and Xiaowei Xu. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. *Data Mining and Knowledge Discovery Journal (DMKD)*, 2(2):169–194, 1998.

[SZ04] K. Sequeira and M. J. Zaki. SCHISM: A New Approach for Interesting Subspace Mining. In *IEEE International Conference on Data Mining (ICDM)*, pages 186–193, 2004.

[ZWZK06] Z. Zeng, J. Wang, L. Zhou, and G. Karypis. Coherent closed quasi-clique discovery from large dense graph databases. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 797–802, 2006.

SQLScript: Efficiently Analyzing Big Enterprise Data in SAP HANA

Carsten Binnig, DHBW Mannheim
carsten.binnig@dhbw-mannheim.de

Norman May, SAP AG Walldorf
norman.may@sap.com

Tobias Mindnich, SAP AG Walldorf
tobias.mindnich@sap.com

Abstract: Today, not only Internet companies such as Google, Facebook or Twitter do have Big Data but also Enterprise Information Systems store an ever growing amount of data (called Big Enterprise Data in this paper). In a classical SAP system landscape a central data warehouse (SAP BW) is used to integrate and analyze all enterprise data. In SAP BW most of the business logic required for complex analytical tasks (e.g., a complex currency conversion) is implemented in the application layer on top of a standard relational database. While being independent from the underlying database when using such an architecture, this architecture has two major drawbacks when analyzing Big Enterprise Data: (1) algorithms in ABAP do not scale with the amount of data and (2) data shipping is required.

To this end, we present a novel programming language called *SQLScript* to efficiently support complex and scalable analytical tasks inside SAP's new main-memory database HANA. *SQLScript* provides two major extensions to the SQL dialect of SAP HANA: A functional and a procedural extension. While the functional extension allows the definition of scalable analytical tasks on Big Enterprise Data, the procedural extension provides imperative constructs to orchestrate the analytical tasks. The major contributions of this paper are two novel functional extensions: First, an extended version of the MapReduce programming model for supporting parallelizable user-defined functions (UDFs). Second, compared to recursion in the SQL standard, a generalized version of recursion to support graph analytics as well as machine learning tasks.

1 Introduction

Today, not only Internet companies such as Google, Facebook or Twitter do have Big Data but also Enterprise Information Systems of companies in other industries store an ever growing amount of mainly structured data (called Big Enterprise Data) that needs to be analyzed. For example, large SAP systems hold more than 70TB of structured data in a single database instance. Moreover, looking at complex system landscapes with multiple SAP systems the total data volume that needs to be analyzed is even much higher while the total amount of data is constantly growing.

In a classical SAP system landscape a central data warehouse (SAP BW) based on a stan-

dard off-the-shelf relational database is used to integrate and analyze all enterprise data. In SAP BW most of the business logic for complex analytical tasks (e.g., a complex currency conversion) is implemented in the application layer on top of the database using the imperative language ABAP in order to be independent from a certain database product. However, this architecture has two major drawbacks when analyzing Big Enterprise Data: First, algorithms implemented in ABAP do not automatically scale with the amount of data that needs to be analyzed. Second, data transfer time is growing with the amount of data that needs to be transferred from the database into the application layer.

In order to implement scalable data warehousing solutions today, MapReduce, in particular its open-source implementation Hadoop [DG08, HAD12], is often used instead of a classical data warehouse on top of a relational database. A major reason for the wide adoption of Hadoop is its simple but scalable programming model consisting of two higher-order functions (i.e., map and reduce), that allow complex user-defined functions that can be parallelized efficiently. High-level programming languages for composing MapReduce programs (e.g., Hive [TSJ+10], PigLatin [ORS+08]) as well as further extensions for iteration and recursion (e.g., HaLoop [BHBE10]) further quelled arguments in favor of Hadoop.

However, compared to relational databases Hadoop is inherently inefficient:

- First, Hadoop does not natively support efficient relational operations such as parallel joins in an efficient manner. Instead it supports only a strict sequence of map and reduce functions. This often leads to complex workarounds (e.g., for expressing joins).
- Second, Hadoop always executes full table-scans and does not provide indexes to selectively read data.
- Third, Hadoop does not provide sophisticated cost-based optimizations as relational databases typically do. Instead analytical tasks are often executed as implemented by the user instead of re-ordering operations in the execution plan.
- Finally, Hadoop has an inefficient execution model which materializes and re-partitions all intermediate results even if this is not required in many cases.

In this paper, we present a novel programming language called *SQLScript* that is currently provided by SAP HANA to support complex analytical tasks inside the database. In contrast to other existing work (e.g., HadoopDB [ABPA+09], Hadoop++ [DQRJ+10]) which mainly focuses on fixing the above mentioned shortcomings of Hadoop by integrating ideas from the database world into Hadoop, we directly integrate complex scalable analytical functions into a commercial main-memory database system (SAP HANA) by extending its query language SQL. Thus, we can directly benefit from the maturity of the database and its efficient query optimization and execution techniques.

In its current version that is commercially available *SQLScript* [SQL12] provides two major extensions to the SQL dialect of SAP HANA: A functional and a procedural extension. The functional extension allows the definition of optimizable (side-effect free) functions which can be used to express and encapsulate complex data flows on Big Enterprise Data.

The procedural extension provides imperative control flow constructs like cursors or exceptions as they are defined for SQL stored procedures. While the functional extension is designed to be highly optimizable and parallelizable to efficiently analyze large amounts of enterprise data, the procedural extension is designed to implement orchestration logic (i.e., to pre- and post-process data for the execution of analytical tasks).

As the main contributions, this paper presents two novel language constructs of the functional extension of *SQLScript* that are currently available as an internal prototype: First, we present the integration of a more flexible and efficient version of the MapReduce programming model into *SQLScript* for supporting parallelizable user-defined functions (UDFs) which avoids the above-mentioned drawbacks of its original version in Hadoop. Second, we present an extension to support a generalized version of recursion when compared to recursion in the SQL standard. These two extensions help to implement complex but scalable business functions inside the database. Both extensions are driven by real world use cases to support complex data analytics for Big Enterprise Data. We show an experimental evaluation based on these use cases to show the efficiency of *SQLScript*.

The outline of this paper is as follows: Section 2 introduces the novel programming language of SAP HANA called *SQLScript*. Section 3 then presents the integration of an extended version of the MapReduce programming model into *SQLScript* to support efficient and parallelizable UDFs. Section 4 discusses the second novel extensions to *SQLScript* to support recursion. Finally, the remaining two Sections show an experimental evaluation using two use cases of SAP and discuss related work.

2 SQLScript

2.1 Main Idea

As already mentioned in the introduction, in this paper we present a language called *SQLScript* which integrates complex scalable analytical functions into a SAP's main-memory database system HANA. Therefore, we first discuss why the existing programming models of relational databases (i.e., SQL and SQL stored procedures) are not well suited for analyzing Big Enterprise Data.

Relational databases traditionally offer two approaches to ship its code to the data: (1) declarative SQL statements or (2) stored procedures implemented using a dialect of SQL stored procedures (e.g. PL/SQL or T-SQL) which embed SQL statements for accessing the data. While SQL statements without SQL stored procedures do not allow to implement complex business logic, imperative language extensions such as SQL stored procedures cannot be efficiently optimized and parallelized.

In order to tackle the before-mentioned issues, *SQLScript* provides two major extensions to the declarative SQL dialect of SAP HANA: A functional and a procedural extension. While the functional extension allows the definition of declarative and optimizable (side-effect free) functions[1] to analyze Big Enterprise Data, the procedural extension provides

[1] Created as read-only procedures in the database.

imperative constructs to implement orchestration logic (i.e., to pre- and post-process data for the execution of an analytical task). Consequently, procedures in SAP HANA are either typed as functional (i.e., as read-only) and have a bag-oriented semantics or they are of a procedural type (i.e, with side-effects) and have a tuple at a time semantics [SQL12].

While procedural code is allowed to call functional code in *SQLScript*, this is not allowed vice versa (see Figure 1). The reason is that the functional extension is designed to be scalable to work on large amounts of data (see Section 2.2) while the procedural extension supports more complex language constructs which do not scale as well. Thus calling procedural code from the functional code would mitigate the scalable execution of functional procedures.

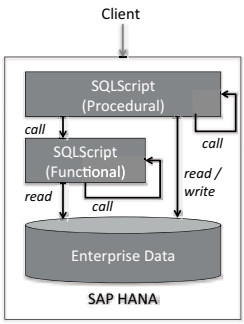


Figure 1: *SQLScript*: functional and procedural extension

The procedural extension provides control flow constructs as they are defined for SQL stored procedures including conditional statements as well as loops over result sets. Moreover, data definition and data manipulation statements (i.e., inserts, updates, deletes) are supported in the procedural extension.

The functional extension supports the definition of declarative read-only procedures (i.e., the side-effect free functions). Such a procedure can have multiple input and output parameters which can either be of a scalar type (e.g., INTEGER, DECIMAL, VARCHAR) or of a table type (as defined in the database catalog). Basic language constructs inside a procedure are single assignments and calls to other read-only procedures. Single assignments can be used to bind the result of a SQL statement (i.e., a table type) or a SQL expression (i.e., a scalar type) to a variable.

Figure 2 shows an example of a read-only procedure, which has two scalar input parameters and returns two output tables (of types `tt_publishers` and `tt_years`) to the caller. The underlying database schema is a simple star schema with a fact table called `orders` and a dimension table called `books`.

The first statement in the procedure assigns a list of identifiers of publishers that publish more books as given by the input parameter `cnt` to the variable `big_pub_ids`. This list of publishers is then used to select those orders of books that have a publisher which is in the given list of big publishers. The result is assigned to the variable `big_pub_books`. Finally, the two final assignments compute the results for the two output parameters: the revenue by publisher `output_pubs` as well as the revenue by year `output_years` (for

the last 10 years).

A complete reference of the current version of SQLScript as it is available in the commercial version of SAP HANA can be found in [SQL12].

```
CREATE PROCEDURE analyzeSales(IN cnt INTEGER, IN year INTEGER,
OUT output_pubs tt_publishers, OUT output_year tt_years)
LANGUAGE SQLSCRIPT READS SQL DATA AS
BEGIN
  big_pub_ids =   SELECT pub_id FROM books -- Query Q1
                  GROUP BY pub_id HAVING COUNT(isbn) > :cnt;
  big_pub_books = SELECT o.price, o.year, o.pub_id -- Query Q2
                  FROM :big_pub_ids p, orders o
                  WHERE p.pub_id = o.pub_id;
  output_pubs =   SELECT SUM(price), pub_id -- Query Q3
                  FROM :big_pub_books
                  GROUP BY pub_id;
  output_year =   SELECT SUM(price), year -- Query Q4
                  FROM :big_pub_books
                  WHERE year BETWEEN :year-10 AND :year
                  GROUP BY year;
END;
```

Figure 2: *SQLScript*: functional extension

The functional extension of *SQLScript* addresses the following drawbacks of the SQL dialect in HANA which also hold for many other SQL dialects in relational databases:

- Decomposing an SQL query can only be done using views. However when decomposing complex queries using views, all intermediate results are visible and must be explicitly typed. Moreover SQL views cannot be parameterized which limits their reuse. *SQLScript* supports decomposition by assignments and parameterization.
- An SQL query can only return one result at a time. As a consequence the computation of related result sets must be split into separate, for the database independent, queries which prevents optimization potentials. *SQLScript* supports multiple input and output parameters.
- Purely declarative SQL queries do not have features to express complex business logic (e.g. the currency conversion of SAP). Only calls to UDFs in a SQL query (as defined in the SQL standard) enable complex business logic. However, these procedures are implemented using imperative SQL stored procedures and thus can not be optimized and parallelized efficiently. The functional extension of *SQLScript* is declarative and thus supports efficient optimization and parallelization.

Moreover, the functional extension of *SQLScript* also addresses the following shortcomings of MapReduce programs in Hadoop: The declarative nature of *SQLScript* allows for optimizations inside the database which are not available for Hadoop programs. Moreover, the integration of *SQLScript* into a relational database provides a streaming execution model instead of an always materializing execution model with efficient relational operators as well as index structures for selectively reading data from tables. Details about the

optimization and execution of *SQLScript* read-only procedures of the functional extension are discussed in the following Section.

2.2 Optimization and Execution

For execution, a read-only procedure is compiled into a data-flow graph (consisting of relational operators) and optimized. For optimization, novel rules have been added to the rewriting phase of the optimizer of SAP HANA to rewrite graph-based plans instead of tree-based plans only (see [BRFR12] for more details). An execution plan for the example of Figure 2 is shown in Figure 3 (left hand side). For simplification, the plan shows boxes which represent the individual query fragments of the procedure instead of single relational operators.

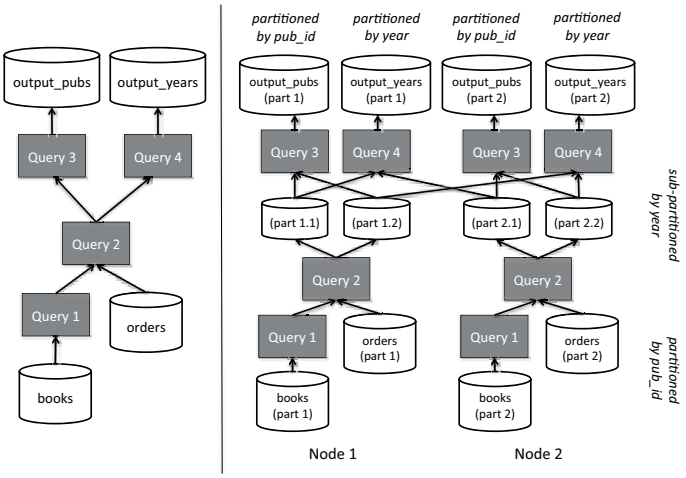


Figure 3: *SQLScript*: execution plan of a read-only procedure

During the execution, SAP HANA materializes intermediate results that are consumed by more than one operator. In the example before, the intermediate result produced by Query Q2 gets materialized since it is consumed by the operators of Query Q3 and Query Q4. Materialization in SAP HANA is also used to re-partition data for better parallelism.

Assume, that the tables `book` and `orders` in the example before are co-partitioned by the attribute `pub_id`. In this case, all queries (Query Q1, Query Q2 and Query Q3 shown in Figure 3 on the right hand side) can be executed in parallel using the same partitioning scheme. However, Query Q4 needs to repartition its input by the attribute `year`. Therefore, the intermediate result of Query Q2 is materialized using a partitioning scheme which partitions the result by `pub_id` and sub-partitions each partition by the attribute `year`. If the plan is executed on different nodes (as in the example), Query Q3 can read all local sub-partitions from one node while Q4 must read sub-partitions from different nodes (see Figure 3 on the right hand side).

3 Generalized MapReduce

3.1 Main Idea

MapReduce is a programming model introduced by Google to analyze big data [DG08]. MapReduce is often applied in use cases with unstructured and semi-structured data (e.g., log analysis) but can also be found as a replacement for classical warehouse solutions on structured data. Originally, the interfaces of both functions *map* and *reduce* are defined as follows:

$$map(k_1, v_1) \rightarrow list([k_2, v_2])$$
$$reduce(k_2, list([v_2])) \rightarrow list([k_3, v_3])$$

Logically, both functions *map* and *reduce* work on tuples with a key and a value. The *map* function processes each incoming tuple $[k_1, v_1]$ separately and produces a list (i.e., a table) of tuples $[k_2, v_2]$. Therefore, each individual map call could be executed in parallel without synchronizing. In a subsequent shuffle step, the output of the *map* function is grouped by the distinct key values of k_2. This step is implicitly executed by the framework. The result is then passed as input to the *reduce* function. Finally, the *reduce* function typically aggregates the values in $list([v_2])$ with the same group key k_2 and returns one or a several tuples $[k_3, v_3]$ as output (i.e., again a table).

In SAP HANA, we use the MapReduce programming model only for structured data (i.e., tables). Thus, we define the interfaces based on structured table types instead of key-value pairs. The table types define the structure of the input and output data. Moreover, to allow parameterization we extend the original interface definitions of both functions to support multiple input tables as well as multiple scalars (e.g., this enables the implementation of joins in both functions). Thus, in SAP HANA both functions *map* and *reduce* are logically defined as follows:

$$map(P, [T_1, ..., T_k], [s_1, ..., s_l]) \rightarrow Q$$
$$reduce(R \text{ GROUP BY } [a_1, ..., a_x], [T'_1, ..., T'_m], [s'_1, ..., s'_n]) \rightarrow S$$

The *map* function gets a table P (with a given table type) as input and applies the user-defined *map* function to each tuple $p \in P$ individually. For each tuple p, the *map* function can append multiple tuples to the output table Q (with a given table type) . The *reduce* function gets a table R (with a given table type) as input and applies the user-defined *reduce* function to each group in table R. The grouping specification is given by a list of group-by attributes $[a_1, ..., a_x]$. For each group, the *reduce* function can append multiple tuples to its output table S (with a given table type).

As mentioned before, compared to the classical MapReduce framework, the *map* and the *reduce* function has additional input parameters: (1) a list of input tables $[T_1, ..., T_k]$ respectively $[T'_1, ..., T'_m]$ and (2) a list of scalar values $[s_1, ..., s_l]$ respectively $[s'_1, ..., s'_n]$

that can be used to parameterize the code. While the input table P of the map function is processed row-wise and the input table Q of the reducer is processed group-wise, the additional input tables can be read completely by both functions. A typical example of such an additional input table T_i is a currency conversion table that is be used to lookup exchange rates inside the *map* function for each row in P.

Another major difference to the classical programming model of the MapReduce framework is that there is no strict sequence of *map* and *reduce* functions (i.e., the output of a *map* function does not need to be consumed by a *reduce* function in *SQLScript*). Instead, any arbitrary sequence of operations can be used (e.g., the output of a *map* function could be used by another *map* function or an SQL query). Thus, complex user-defined functions expressed as mappers or reducers can be mixed with any other SQL statements. Thus a mapper can also be seen as a row-level UDF in SQL with the difference that it can take additional parameters as input.

An example which extends the function in Figure 2 by a *map* and a *reduce* function is given in Figure 4 and Figure 5. The call of the *map* function in Figure 4 calculates a currency conversion (before the aggregations in Q3 and Q4). The *map* function in Figure 5 is defined as a separate read-only procedure which has a special type (i.e., type `MAPPER`). The *map* function is applied to each tuple of its input table `big_pub_books` and the output is assigned to the output parameter `big_pub_books_conv`. Additionally, the function gets a constant input table `conv_rates` and a constant scalar value `target_curr` to implement a currency conversion. The *reduce* function (which replaces aggregation Query Q4) is defined in a similar way as type `REDUCER` in Figure 5.

```
1  CREATE PROCEDURE analyzeSalesConv(
2  IN cnt INTEGER,  IN conv_rates tt_convrates
3  OUT output_pubs tt_publishers, OUT output_year tt_years)
4  LANGUAGE SQLSCRIPT READS SQL DATA AS
5  BEGIN
6    big_pub_ids =   SELECT pub_id FROM books -- Query Q1
7                    GROUP BY pub_id HAVING COUNT(isbn) > :cnt;
8    big_pub_books = SELECT o.price, o.year, o.pub_id, o.curr -- Query Q2
9                    FROM :big_pub_ids p, orders o
10                   WHERE p.pub_id =o.pub_id;
11
12   CALL mapConv(:big_pub_books, [:conv_rates], ["EUR"], :big_pub_books_conv);
13
14   output_pubs =   SELECT SUM(price), pub_id -- Query Q3
15                   FROM :big_pub_books_conv
16                   GROUP BY pub_id;
17
18   CALL reduceBooksByYear(:big_pub_books_conv GROUP BY year,
19     :output_year); -- Query Q4
20 END;
```

Figure 4: *SQLScript*: calling a *map* and a *reduce* function

Logically the user-defined code which is implemented by the mapper refers to one tuple in the input table `big_pub_books` (by calling the method `currentTuple()`). The additional constant input table `conv_rates` is referred as a complete table. In a similar

```
CREATE PROCEDURE mapConv(IN big_pub_books tt_big_books,
[IN conv_rates tt_conv_rates], [IN  target_curr CHAR(3)],
OUT big_pub_books_conv  tt_big_books_conv)
LANGUAGE C++ TYPE MAPPER AS
BEGIN
  //Pseudo code
  string srcCurr = big_pub_book.currentTuple().getColumn("currency");
  decimal price = big_pub_book.currentTuple().getColumn("price");
  decimal rate = getRate(conv_rates, srcCurr, target_curr);
  Tuple big_pub_book_conv = new Tuple();
  big_pub_book_conv.setColumn("convPrice", price*rate);
  //set other columns
  ...

  big_pub_books_conv.appendRow(big_pub_book_conv);
END;

CREATE PROCEDURE reduceBooksByYear(
IN big_pub_books_conv tt_big_books_conv GROUP BY year, [], [],
OUT books_by_year  tt_years)
LANGUAGE C++ TYPE REDUCER AS
BEGIN
  //Pseudo code
  decimal price = 0;
  int year = -1;
  for(Tuple big_pub_book_conv: big_pub_books_conv.currentGroup()){
    price += big_pub_book_conv.getColumn("price");
    if(year==-1)
      year = big_pub_book_conv.getColumn("year");
  }

  Tuple books_by_year = new Tuple();
  books_by_year.setColumn("price", price);
  books_by_year.setColumn("year", year);
  books_by_year.appendRow(book_by_year);
END;
```

Figure 5: *SQLScript*: *map* and *reduce* function

way, the user-defined code which is implemented by the reducer in the example refers to one group of tuples in the input table `big_pub_books_conv` with the same values for the group-by attribute `year`. However, as described in the next Section, the physical execution of both functions is different.

3.2 Optimization and Execution

For compilation a call to a *map* or a *reduce* function is translated into a *map* or a *reduce* operator in the plan. Figure 8 shows the compiled plan for the function in Figure 5. Compared to the logical execution of a *map* function, a *map* operator physically does not process a tuple at a time as input of its input table P. Instead, the *map* operator processes partitions of its input table P (i.e., a bag of tuples) and applies the *map* function to each

input tuple in its partition separately. Each input partition of table P can be processed in parallel by separate *map* operators. A *map* operator can produce multiple output tuples for each input tuple that are appended to its output.

A similar execution model is used for the *reduce* operator. Instead of processing only one input group of its input table Q at a time, the *reduce* operator gets an input partition which can contain multiple groups. Before execution, the *reduce* operator thus groups its input by the given grouping attributes. The *reduce* operator then applies the given user-defined code to each group individually. The *reduce* operator can also produce multiple output tuples for each input group that are appended to its output.

For both functions, the additional input parameters (i.e., the tables and scalar values) are logically replicated to all operators which process an input partition of table P and table Q. If the two operators which refer to the same input parameter are executed on the same node, no physical replication is necessary. In this case, both operators refer to the same input data. However, if two operators which refer to the same input parameter are executed on different nodes, the data must by physically replicated.

Figure 6 shows the execution of a *map* operator and a *reduce* operator which result from the procedures in Figure 5. In this example we see two instances of the *map* operator that are applied in parallel to each row of the two different input partitions. We do not show the two additional input parameters (i.e., the currency conversion table and the target currency) for simplicity. These two input parameters are logically replicated to all instances of the *map* and *reduce* operator.

Moreover, on the right hand side of Figure 6, we see one instance of the *reduce* operator which processes the union of the two output partitions of the two *map* instances. The *reduce* operator has to group its input by the attribute `year` and then processes the two resulting groups separately producing one output tuple per group.

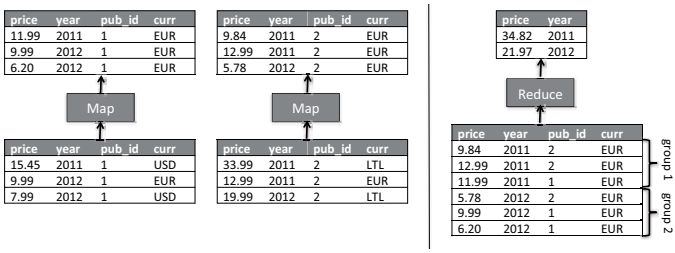


Figure 6: *SQLScript*: execution model for the *map* and the *reduce* operator

For the parallel execution, *SQLScript* allows annotations to define a partitioning specification for the input and output tables of both operators. Figure 7 shows an example of this annotations for both types of functions. The semantics of this partitioning specification is as follows: If the input of a *map* or a *reduce* operator is partitioned by the annotated partitioning specification, then the output is guaranteed to satisfy the given output partitioning specification as well. This helps to avoid irrelevant repartitioning in the plan which is expensive in a parallel distributed execution environment.

For example, if the input of the *map* function in Figure 7 is partitioned by the attribute

pub_id then the operator guarantees that the output satisfies the same partitioning specification. Defining the partitioning specification for a *map* or a *reduce* is optional. For the *reduce* operator, the partitioning schema must be compatible to the given grouping attributes (i.e., it has to guarantee that groups with the same group value are in the same partition).

```
1  CREATE PROCEDURE mapConv(
2  IN big_pub_books tt_big_book PARTITIONED BY pub_id,
3  IN conv_rates table_conv_rates, IN targetCurr STRING,
4  OUT big_pub_books_conv tt_big_books_conv PARTITIONED BY pub_id)
5  LANGUAGE C++ TYPE MAPPER AS ...
6
7  CREATE PROCEDURE reduceBooksByYear(
8  IN big_pub_books tt_big_books_conv GROUP BY year PARTITIONED BY year,
9  OUT books_year tt_big_books_year PARTITIONED BY year)
10 LANGUAGE C++ TYPE REDUCER AS ...
```

Figure 7: *SQLScript*: partitioning specification for a *map* and a *reduce* function

Using this partitioning specification, the optimizer can dynamically detect the need to re-partition the input tables of a *map* or a *reduce* operator. Consequently, the implicit grouping step which is executed in the original MapReduce framework before each *reduce* step can be avoided in *SQLScript* if the partitioning specification of the output of the previous step matches the input partitioning specification. The number of partitions as well as the partitioning method (e.g., by hashing) that are actually used for query processing is determined by the optimizer and depends on several factors (e.g., the partitioning scheme of the input tables, the degree of parallelism, the intermediate result sizes).

In Figure 8, we see a parallelized execution plan for the procedure in Figure 5. In this example, the input tables are partitioned by the attribute pub_id into two partitions. This partitioning scheme is kept for the input of the *map* operator. Thus, the output of the *map* operator is also partitioned by the attribute pub_id (as defined by the interface). This output can be consumed directly by Query Q3 without repartitioning. However, before the output can be consumed by the *reduce* operator (i.e., Query Q4) it must be re-partitioned since Q4 needs to group its input by the attribute year. This re-partitioning can be achieved either by a simple union of both output partitions or by sub-partitioning the output partitions by the attribute year (as described in Section 2 before).

Currently, no other optimizations (like selection-pushdown) are applied for a *map* or a *reduce* operator. However, additional annotations could help to find out which rewrite rules can be applied to these operators. Adding annotations for the rewriting phase is one avenue of future work.

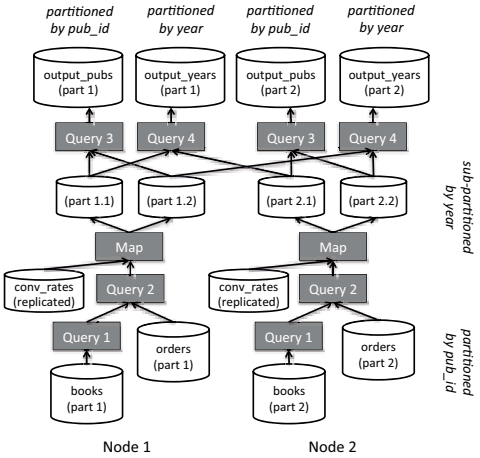


Figure 8: *SQLScript*: execution plan including a *map* operator

4 Generalized Recursion

4.1 Main Idea

Recursion enables different kinds of use cases in data analytics such as graph analysis and machine learning tasks. Major use cases for Big Enterprise Data include the ability to traverse hierarchical data (e.g., hierarchies of product groups, employee hierarchies) and to process graph algorithms (e.g., shortest paths or convex hulls). Moreover, algorithms for machine learning (e.g., k-means) are also require this language construct to examine enterprise data.

Compared to the classical definition of recursion in the SQL standard, *SQLScript* supports a generalized version of recursion. In the SQL standard the definition of a recursive view (using a `WITH RECURSIVE`-statement) is not parameterizable and does not support multiple output parameters. In *SQLScript*, recursion is defined on the procedure level (i.e., read-only procedures can call themselves) instead on the statement level. Thus, scalars and tables can be used as input parameters and a recursive procedure can produce multiple output parameters. Iterative problems can often be re-formulated using recursion. Thus, in the functional extension, we currently do not explicitly support a language construct for iteration. However, the procedural extension of SQLScript [SQL12] supports iteration (e.g., loops over result sets).

Inside a recursive procedure any other read-only procedure (i.e., also *map* or *reduce* functions) can be called. Thus, iterative machine learning algorithms as supported by HaLoop [BHBE10] are supported directly in *SQLScript* by calling *map* functions and *reduce* functions inside the recursion.

The recursive call in *SQLScript* is implemented using a `IF-ELSE` statement at the end of a function (i.e., we support only tail-recursive calls). The termination condition is given

by the predicate of the `IF` clause. The recursive call must be the first statement in the `IF` clause while the subsequent statements must assign results to all output parameters (using a simple assignment, a UNION or a UNION ALL statement). The `ELSE` block is executed if the termination condition holds. That block is only allowed to assign results to the output parameters (again using a simple assignment, a UNION or a UNION ALL statement).

Figure 9 shows an example table which describes the connections between customers (e.g., in a CRM system). A typical task on this graph structure is to compute a list of customers that are connected to a key customer only by edges which have a weight which exceeds a certain threshold. A possible input parameter to such a procedure is the depth, i.e., the distance of customers in the graph that are analyzed when using one certain customer as a starting point.

Frm	To	Weight
1	2	3
1	3	2
2	4	3
2	5	1
2	6	4
3	6	2
6	7	5

Figure 9: Table `CustomerConnections`

This task can be implemented in *SQLScript* using a recursive read-only procedure as the one shown in Figure 10. The procedure has the following input parameters: the maximal depth (parameter `depth`), the current depth (parameter `currDepth`) and a list of connections (parameter `current`) that resulted from the last recursion step (i.e., a table with a `from` and a `to` column). In the first call of the procedure, the parameter `current` holds the customer which is used as starting point.

The first assignment of the procedure filters the relevant connections that exceed a certain threshold on the weight attribute. This intermediate result is an invariant for all recursion steps. The intermediate result table `relevant` is then used to calculate a list of customers that are connected to the given list of customers (i.e., to the input table `current`). If the maximal depth is reached, the recursion stops. Otherwise the list of customers for the next depth in the graph is calculated.

4.2 Optimization and Execution

A recursive procedure is compiled into a cyclic data flow graph as already described in [BRFR12]. Figure 11 shows the data flow graph of the recursive procedure in Figure 7 (left hand side).

In order to optimize recursion, our extension to SAP HANA supports the following rewrites:

- **Materialize Invariants:** Invariants (i.e., partial plans that create intermediate results which are static over different recursion steps) are separated and executed only once.

```
CREATE PROCEDURE convexHull(IN depth INTEGER, IN currDepth INTEGER,
IN current tt_fromto, OUT hull tt_fromto)
LANGUAGE SQLSCRIPT READS SQL DATA AS
BEGIN
  relevant = SELECT Frm, To -- Query Q1
             FROM CustomerConnections
             WHERE weight >= 2;

  temp =     SELECT c.Frm, r.To  -- Query Q2
             FROM :current c, :relevant r
             WHERE c.To = r.Frm;

  currDepth = currDepth  + 1;

  IF(currDepth < depth)  -- Recusive Call C3
    CALL convexHull(depth, currDepth, temp, temp2)
    hull = :temp UNION :temp2;
  ELSE
    hull = :temp;
END;
```

Figure 10: *SQLScript*: recursive procedure

This optimization is shown in Figure 7 on the right hand side: the invariant which is stored in the intermediate result `relevant` is computed by a partial plan only once. Compared to HaLoop materializing invariants is not implicitly hidden in the execution model by caching but explicitly applied during the optimization phase.

- **Internal Rewrites:** Inside a recursive procedure, we can use all normal rewrites such as selection- and projection-pushdown.
- **Cross-Procedure Rewrites:** If the results of a recursive procedure are consumed by other procedures, we can apply the following rewrites: selection- and projection-pushdown of the calling procedure are supported if the respective operator can be pushed over the complete recursive procedure over an input table which is defined recursively. For example, if in Figure 7 a selection `c.frm=1` is executed on top of the result `hull` then this selection can be pushed over the input `current`.

For parallelization, we analyze the plan dynamically for possible partitioning schemes and add repartitioning operations into the plan as described before.

5 Experimental Evaluation

In this Section, we present the experimental evaluation of the two novel functional extensions for *SQLScript* based on use cases of SAP: the generalized versions of MapReduce as well as recursion. Both ideas are implemented at SAP as a prototype in SAP HANA to extend the commercially available version of *SQLScript*.

As hardware we used a single machine with 512GB of main memory and four Intel Xeon

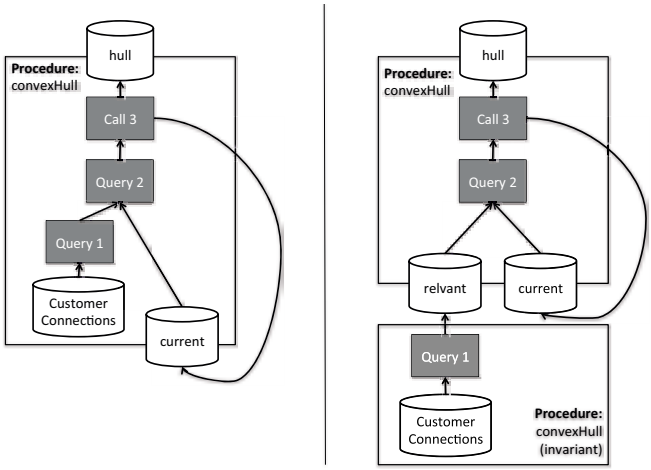


Figure 11: *SQLScript*: execution plan of a recursive procedure

X7560 processors each with eight cores (i.e., 32 cores in total). The software stack consisted of SUSE Linux 11 running a database instance of SAP HANA.

5.1 Experiment: Currency Conversion

The first experiment is based on the Shipping Priority Query (Q3) of the TPC-H benchmark [TPC12] which returns the first 10 selected rows. This query retrieves the unshipped orders with the highest value. Figure 12 shows the original Query Q3:

```
SELECT
   l_orderkey, o_orderdate, o_shippriority
  SUM(l_extendedprice*(1-l_discount)) as revenue,
FROM customer, orders, lineitem
WHERE c_mktsegment = '[SEGMENT]'
   and c_custkey = o_custkey
   and l_orderkey = o_orderkey
   and o_orderdate < date '[DATE]'
   and l_shipdate > date '[DATE]'
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY revenue desc, o_orderdate
LIMIT 10;
```

Figure 12: TPC-H: query Q3

For the experiment, we extended this query to use a simplified version of the SAP currency conversion before the aggregation on the attribute `extended_price`. Therefore, we first pre-aggregated the data using the currency as an additional group-by attribute. Then, we applied the currency conversion using different implementations (as described below).

Finally, we post-aggregated the result removing the currency from the group-by attribute.

The simplified version of the SAP currency conversion is based on a currency conversion table as shown in Figure 13.

SCurreny	TCurrency	RefDate	Rate
EUR	USD	2012-10-15	1,3
USD	EUR	2012-10-15	0,769
LTL	EUR	2012-10-15	0,289

Figure 13: Table `CurrConv`

The currency conversion is based on the date of the conversion (i.e., the attribute `RefDate`) and has three cases:

- **Direct Conversion:** There exists a conversion rate from the given source to the target currency (e.g., as from EUR to USD or vice versa).
- **Inverted Conversion:** There exists only a conversion rate from the given target to the source currency. Thus, the inverted rate is used (e.g., as from EUR to LTL).
- **Indirect Conversion:** There does neither exist a direct nor a inverted conversion. In this case the conversion must be done using a reference currency (e.g., from LTL to USD we have to use EUR as reference currency).

For the experiment, we executed the Shipping Priority Query (Q3) of the TPC-H benchmark in three variants. (1-SQL: No currency conversion) the original version of Q3 using one SQL query, (2-SQLScript: Generalized MapReduce) a variant of Q3 including the currency conversion implemented as a *map* function which takes the currency conversion table as an additional input parameter and (3-SQLScript: Procedural) a variant of Q3 including the currency conversion implemented using SQLScript procedural code which is called for each row on the pre-aggregated result. Version (1) and (3) thus represent the baselines (lower and upper limit).

For the variant (2) and (3), we extended the original `lineitem` table in the TPC-H schema by a currency column `curr` and used the attribute `o_orderdate` as reference date for the conversion. We generated additional data for the new column `curr` in the table `lineitem` such that each case of the currency conversion must be executed with the same probability. Additionally, we generated a currency conversion table (as shown in Figure 13) holding information for all currencies and reference dates in the `lineitem` table. As target currency for the function call, we used *EUR*.

Figure 14 shows the result of the execution of the three variants mentioned before on different scaling factors (SF) for the TPC-H benchmark (up to SF 25). The variants (1) and (3) have been executed using 32 threads. For variant (2) which uses the *map* operator for the currency conversion, we used 16 and 32 threads (while all other operators of Q3 were still using 32 threads).

As we can see in Figure 14, the variant with the *map* operator is much faster than the procedural SQLScript implementation. The reason is that the procedural SQLScript variant

issues multiple SQL queries for the currency conversion while variant (2) only requires one (complex) user-defined *map* operator which internally builds a hash index on the currency conversion table to do fast lookups of the exchange rates. As a result, the complex user-defined function implemented as a mapper adds only 200ms with 32 threads and 250ms with 16 threads to the runtime of $Q3$ for each scaling factor (since the pre-aggregated result has the same size for all scaling factors). The SQLScript procedural extension adds additional 13s to the runtime of $Q3$.

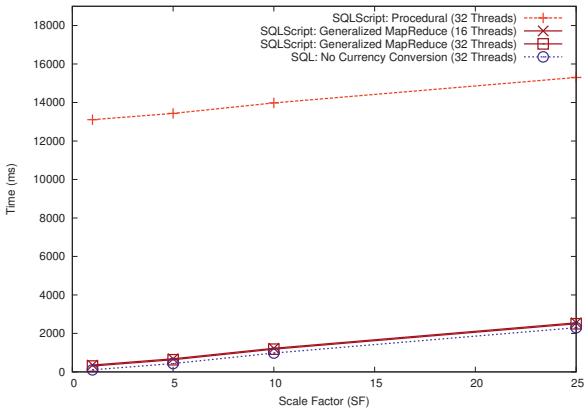


Figure 14: TPC-H query Q3 with and without currency conversion

5.2 Experiment: Graph Analysis

For this experiment we used a recursive procedure which is similar to the one shown in Figure 10 already. As data we used the table `Customer` of the TPC-H benchmark and a table `Livejournal` (which has a similar schema as the table in Figure 9). The table `Livejournal` comes from the Stanford Network Analysis Project [Les12] which provides data from social networks. The table `Livejournal` has approximately $68m$ entries with approx. $5m$ distinct nodes. For the table `Customer`, we used the SF 35 (i.e., approximately $5m$ customers) .

In order to select relevant entries from the table `Livejournal` we join this table with the table `Customer` based on the customer key. Moreover, we select customers from certain nations only to reach a selectivity from 10% to 80% of the `Livejournal` table. The result of this join corresponds to the table `relevant` in Figure 10.

We executed the recursive procedure with a maximal depth of 3 using different optimization variants[2]: one variant which materializes invariants and another variant without this optimization (i.e., the invariant is computed for each iteration). Moreover, we also varied the number of partitions used from 1 to 8 for each of these variants to exploit parallelism (while each operator was configured to use at most 8 threads). Figure 15 shows the runtime for the different variants using selectivities from 10% to 80% for the selection operator on the table `Customer`.

[2]We also executed the procedure with a maximal depth of 5 and 7 but the results looked similar.

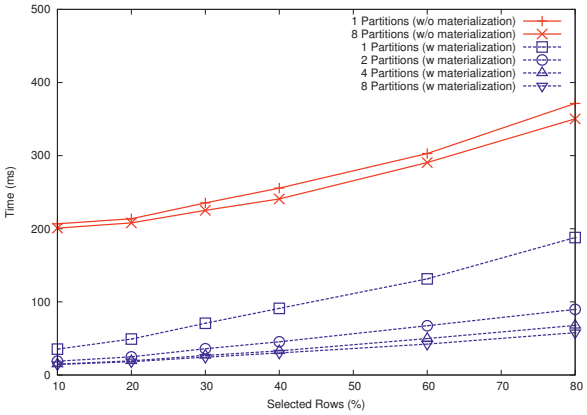


Figure 15: Recursive query with and without materialization of invariants

As we can see in Figure 15, the runtime (of the variants which do not materialize the invariant) is dominated by the redundant execution of the sub-plan which produces the invariant. Moreover, partitioning the plan (on one machine) speeds-up query processing due to parallelism. The results when using 4 and 8 partitions (for the variants with materialization of the invariant) does not show a huge difference since the CPUs of the machine were already saturated using 4 partitions (i.e., 8 threads per partition have been used). Thus, increasing the parallelism to 64 threads on 8 partitions did not show a huge difference in the resulting runtime.

6 Related Work

Most related to *SQLScript* are extensions to Hadoop to tackle its inefficiencies of query processing in Hadoop in different areas such as new architectures for big data analytics, new execution and programming models but also in the field of integrating systems like MapReduce and databases.

HadoopDB [ABPA+09] turns the slave nodes of Hadoop into single-node database instances. However, HadoopDB relies on Hadoop as its major execution environment (i.e., joins are often compiled into inefficient map and reduce operations). Only in its commercial version [BPASP11], HadoopDB presents a component called SideDB, which replaces the Hadoop execution environment by a database to execute operations like joins more efficiently.

Hadoop++ [DQRJ+10] and Clydesdale [KST12] are just two out of many other systems also trying to address the shortcomings of Hadoop, by adding better support for structured data, indexes and joins. However, like other systems, Hadoop++ and Clydesdale cannot overcome Hadoop's inherent limitations (e.g., not being able to execute joins natively).

PACT [ABE+10] and ASTERIX [BBC+11] suggest new execution models, which provide a richer set of operators than MapReduce (i.e., not only two unary operators) in order to deal with the inefficiency of expressing complex analytical tasks in MapReduce. Although

promising, *SQLScript* explores a different design, by focusing on existing databases and novel query optimization techniques.

HaLoop [BHBE10] extends Hadoop by recursive and iterative analytical tasks and improves Hadoop by certain optimization (e.g, caching loop invariants instead of producing them multiple times). *SQLScript* supports a more general version of recursion than HaLoop while optimizations are not implicitly hidden in the execution model (by caching) but explicitly applied during the optimization phase.

In the area of programming languages for big data analytics there are a lot of proposals as well. For example, Hive [TSJ+10] and PigLatin [ORS+08] have been proposed as high-level programming languages for defining map-reduce jobs in Hadoop. Those programs are optimized and then executed using Hadoop as execution environment. *SQLScript* extends these approaches for a better UDF support in databases so that the data-flow graphs including user code can be holistically optimized.

Moreover, major database vendors currently include Hadoop as a system into their software stack and optimize the data transfer between the database and Hadoop e.g. to call MapReduce tasks from SQL queries. Greenplum and Aster Data are two commercial database products for analytical query processing which support MapReduce natively in their execution model. However, to our knowledge they do not support the extended version as we do.

Finally, there has also been a lot of research work on the field of recursion in the context of SQL. In this paper, we extend many known techniques for single SQL queries to procedures with multiple in- and output parameters. For example, we extend the optimization rules presented in [Ord05] (i.e., selection pushdown) to work for recursive SQLScript procedures with multiple in- and output parameters.

7 Conclusions and Outlook

In this paper, we presented a novel programming language called *SQLScript* to support complex analytical tasks in the distributed in-memory database SAP HANA. *SQLScript* provides two major extensions to SQL: A functional and a procedural extension. The functional extension allows the definition of optimizable side-effect free functions which can be used to express and encapsulate complex data flows. Moreover, we presented two novel language constructs of the functional extension of *SQLScript*: First, an extended version of the MapReduce programming model to support parallelizable user-defined functions (UDFs). Second, an extended version of recursion (i.e., iteration) compared to recursion in the SQL standard, which takes multiple input parameters and can produce multiple output parameters. For both extensions, we showed optimization and execution strategies that were analyzed in an experimental evaluation to show their efficiency.

As future work, we plan to extend the static optimization and execution by adaptive techniques (i.e., changing the plan parallelism dynamically). Moreover, we also plan to add better rewrite techniques for the *map*- and *reduce* operators by further annotations. Another major issue includes the debugging and testing of these complex functions on the database side.

References

[ABE+10] Alexander Alexandrov, Dominic Battré, Stephan Ewen, Max Heimel, Fabian Hueske, Odej Kao, Volker Markl, Erik Nijkamp, and Daniel Warneke. Massively Parallel Data Analysis with PACTs on Nephele. *PVLDB*, 3(2), 2010.

[ABPA+09] Azza Abouzeid, Kamil Bajda-Pawlikowski, Daniel J. Abadi, Alexander Rasin, and Avi Silberschatz. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. *PVLDB*, 2(1):922–933, 2009.

[BBC+11] Alexander Behm, Vinayak R. Borkar, Michael J. Carey, Raman Grover, Chen Li, Nicola Onose, Rares Vernica, Alin Deutsch, Yannis Papakonstantinou, and Vassilis J. Tsotras. ASTERIX: towards a scalable, semistructured data platform for evolving-world models. *Distributed and Parallel Databases*, 29(3):185–216, 2011.

[BHBE10] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. HaLoop: Efficient Iterative Data Processing on Large Clusters. *PVLDB*, 3(1), 2010.

[BPASP11] Kamil Bajda-Pawlikowski, Daniel J. Abadi, Avi Silberschatz, and Erik Paulson. Efficient processing of data warehousing queries in a split execution environment. In *SIGMOD*, pages 1165–1176, 2011.

[BRFR12] Carsten Binnig, Robin Rehrmann, Franz Faerber, and Rudolf Riewe. FunSQL: it is time to make SQL functional. In *EDBT/ICDT Workshops*, pages 41–46, 2012.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[DQRJ+10] Jens Dittrich, Jorge-Arnulfo Quiané-Ruiz, Alekh Jindal, Yagiz Kargin, Vinay Setty, and Jörg Schad. Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). *PVLDB*, 3(1):518–529, 2010.

[HAD12] Apache Hadoop. http://hadoop.apache.org, 2012.

[KST12] Tim Kaldewey, Eugene J. Shekita, and Sandeep Tata. Clydesdale: structured data processing on MapReduce. In *EDBT*, pages 15–25, 2012.

[Les12] Jure Leskovec. Stanford Large Network Dataset Collection. http://snap.stanford.edu/data/, 2012.

[Ord05] Carlos Ordonez. Optimizing recursive queries in SQL. In *SIGMOD Conference*, pages 834–839, 2005.

[ORS+08] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD*, pages 1099–1110, 2008.

[SQL12] SAP HANA SQLScript Reference. http://help.sap.com/hana/hana_dev_sqlscript_en.pdf, 2012.

[TPC12] TPC-H. http://www.tpc.org/tpch/, 2012.

[TSJ+10] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Anthony, Hao Liu, and Raghotham Murthy. Hive - a petabyte scale data warehouse using Hadoop. In *ICDE*, pages 996–1005, 2010.

Seamless Integration of Archiving Functionality in OLTP/OLAP Database Systems Using Accelerator Technologies

Knut Stolze* Oliver Köth* Felix Beier†* Carlos Caballero* Ruiping Li‡

Abstract:

The recent version of the IBM DB2 Analytics Accelerator introduces the High Performance Storage Saver as a new product feature. It paves another part of the way towards integrating OLTP and OLAP into a single database system. We present the technical details of this approach, which integrates archiving functionality into the DB2 relational database systems with seamless and transparent access to the archive data. The IBM DB2 Analytics Accelerator for DB2 for z/OS offers the storage area for the archive and also delivers exceptional performance for querying the data online, archived and non-archived data alike.

In this paper, we describe the administrative interfaces controlling which table partitions shall be archived (or restored) and the associated monitoring interfaces. Enhancements in the DB2 optimizer provide control whether archive data shall be considered for query processing or not. Strong focus was laid on using simple interfaces, and we present our approach taken during product design and development.

1 Introduction

The ever-growing size of data warehouse systems requires new and innovative approaches to address performance issues. Star and snowflake schemas [Leh03] are not always the best data model for reporting and analytical systems. An example is the IBM® data warehouse industry model for banking and financial markets [IBM12]. That raises the bar for solving performance problems in real-world analytical products and systems, and new technologies have to be adopted into the database system kernel for solving them.

Customers desire to run analytical queries directly on OLTP systems to base business decisions on the most recent data and to discover future trends. Thus, we see a tendency in the market to gradually merge OLTP and OLAP systems – not only by using the same software products, but also at the data level itself [Inm99]. The benefits are reduced maintenance and operations overhead as well as hardware consolidation opportunities. However, this trend increases the data volume in OLTP systems quite significantly – although the majority of the data is now read-only and kept as history for analytical workload. The OLTP system becomes an operational data store (ODS) [Inm99] with an emphasis on analytics.

*IBM Germany Research & Development, Böblingen, Germany
†Ilmenau University of Technology, Germany
‡IBM Silicon Valley Lab, San Jose, USA

So far few customers of relational database systems have adopted such an integrated approach because available database system products have a hard time to cope with the diverse requirements in an integrated fashion. The IBM DB2 Analytics Accelerator (IDAA) [BBF+12] is a system that delivers extremely high performance for analytical and reporting queries. IDAA is a hardware and software appliance, which comes with its own internal storage structures. It is tightly integrated with DB2® for z/OS® already. While its primary use case is for data warehouses, it can also be applied to other database schemas. The integration into the DB2 optimizer offers the base line to add a new use case for the accelerator: serving as a high performance online archive solution.

In most analytical systems in ODSs and enterprise data warehouses, tables with significantly large amounts of data are horizontally partitioned to accommodate parallel processing and to support very easy roll-in and roll-out of whole partitions. If a datetime-based partitioning column is used to establish the partition ranges, there are typically only a few partitions on which data modification activities occur, i. e., partitions that hold current data. Data in the other partitions does not change at all or only extremely rarely due to business reasons or legal requirements. Such static data is a very good candidate for archiving, especially if it is archived into a system that provides high performance, transparent, and online access to the archived data.

Archiving data into IDAA results in a reduction of the data volume in the DB2 table, which implies smaller indexes (potentially saving multiple levels in a B-Tree) and smaller materialized query tables[1]. There is even the potential to do away with some access paths (indexes) – either because they are no longer needed and a table scan may be sufficiently fast now, or because multiple access paths with (partially) overlapping columns can now be combined. Durability of the archive data is still guaranteed based on backup/recovery strategies already established in customer environments. Another benefit of the solution is the reduction of disk storage required for the DB2 table, which means that less storage on high quality disks is needed. Hence, the IDAA online archiving functionality is called *High Performance Storage Saver* (HPSS), a term which emphasizes the latter user case.

Figure 1 illustrates that some table partitions remain in DB2 *and* in the accelerator for operational workload, i. e., the *recent* or *non-archived* data. A copy of the non-archived data exists in IDAA as well to facilitate query processing. Other partitions with history data reside on the accelerator only and that data was purged from DB2, i. e., it was *archived* to the accelerator. The DB2 optimizer will always direct any queries that touch the archived data to the accelerator, while queries against non-archived data only follow the usual query routing criteria of IDAA.

One of the main requirements for the IDAA HPSS feature was to avoid changes or rewrites of an application's SQL statements. By default, all queries access only the non-archived data, and such queries may be routed to the accelerator if the DB2 optimizer determines that it is more beneficial to do so. In case an application wants to include archived and non-archived data for the processing of a query, it has to set a special register to convey this fact to the DB2 optimizer, The optimizer will then always route the query to the accelerator. The SQL statement texts of queries do not have to be changed at all – only the setting of the special register has to be triggered by the application.

[1] Materialized query tables are also known as *materialized views*.

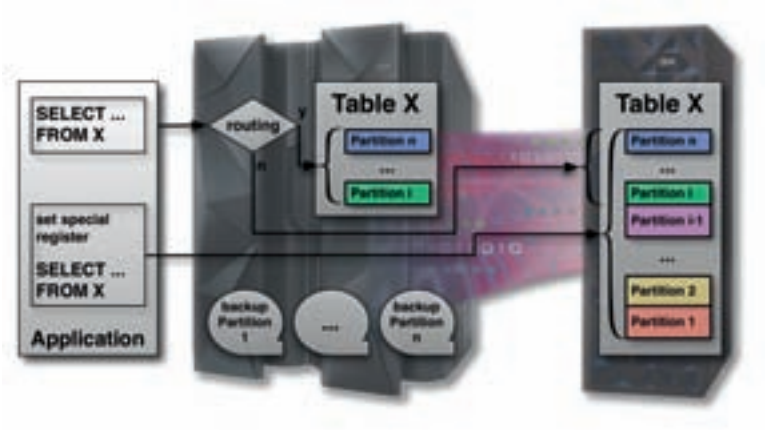


Figure 1: Principles of the High Performance Storage Saver

The remainder of the paper is structured as follows. Section 2 gives an overview on related technology, products, and solutions that provide multi-temperature, archiving, or near-line storage (NLS) functionality. We briefly touch on the differences between those solutions and the online archiving functionality in IDAA, for which an overview on the architecture is given in section 3. The concepts and the specific implementation details applicable to HPSS are described in section 4. We cover the set of stored procedures that perform archiving and restore operations, and exposed monitoring interfaces. The behavior and semantics of the new special register and its impact on the DB2 optimizer are outlined. Performance measurements on the archiving process itself and also for query processing with and without archive data have been conducted, and the results are summarized in section 5. Finally, the paper concludes in section 6 with a summary and general outlook to future direction for the development of this new product feature.

2 Related Work

Database archiving [Sch01] is the process of moving records that are not expected to be referenced from an operational database to an archive data store. Basically, the data is partitioned into operational and archive data. The partitioning in different data stores is typically done depending on the "hotness" of the data. This supports the placement of very frequently used data on high-quality, fast (and more expensive) storage, while never or rarely used data is to be put on less expensive but slower storage. Naturally, there can be many different layers in between those extremes.

The technique of automatically moving data between different storage media is commonly referred to as hierarchical storage management (HSM). Such systems monitor data usage and decide what data can be moved to slower/cheaper storage media and which data should be moved to faster/expensive media. A HSM can decide at runtime if data has to be moved

from one level to another and applies necessary actions. HSM is conceptually analogous to cache hierarchies in CPUs where we may have several levels of high speed SRAM for caches, external DRAM, and SSDS, slower hard disk, and even slower tape devices for persistent storage.

HSM usually work at the file level. While this approach works well for individual files, deploying such a solution in the context of a relational DBMS at a higher abstraction level – like relational tables or partitions – is challenging. Originally, HSM could take backups (or copies of the data) of the entire database, but the database had to be taken offline (for consistency reasons) for that time window. In order to avoid this, most RDBMS have special APIs that allow creating backups while the database is kept online. The DBMS ensures the consistency of the backup image by considering the currently running transactions and the state of the buffer pool. These approaches usually use the transaction log for capturing changes that occur while the backup is created. There are various software solutions that perform HSM with such a database integration, for example, IBM Tivoli Storage Manager [HBB+01] and Oracle SAM-QFS [Ora11].

The way to create such database backups is often referred to as *archiving*. Archived data is typically not directly accessible by users via the relational interface and, thus, cannot be used for analytical evaluation. This is a significant difference with regards to IDAA HPSS where we have an *online archive*, i. e., data in the archive can be still queried and querying the archive data comes with the exceptional query performance for which IDAA is known.

Many DBMS vendors provide what is called multi-temperature data solutions. These so-called near-line storage repositories have many things in common with archives, but the key difference is, that the data they hold – although being used less frequently – is still accessible for query processing. Querying this data will usually have no penalties for users accessing the online database. As an example of such an approach we have SAP NetWeaver BW NLS. [IBM10] NLS maintains two different databases (possibly in different database systems): (1) the online database with the operational data, and (2) and the near-line storage database. The BW OLAP processor splits queries run against the system into two sub-queries and aggregates the partial results returned by each of them. This approach is based on the fact that clients issue their queries against the SAP BW OLAP processor and not against the underlying database system itself. Queries for the underlying DBMS are generated within SAP. In contrast, IDAA HPSS is deeply integrated into DB2 for z/OS at SQL level. So accessing the archive is transparent to *any* application that connects to it. In addition we get the full accelerator performance benefits for such queries.

3 IBM DB2 Analytics Accelerator Overview

IDAA [BBF+12] is based on the Netezza appliance [Fra11], which is used as backend. It provides the data storage and querying capabilities to manage large amounts of data and to provide exceptional performance for querying the data.

Figure 2 illustrates the IDAA high-level architecture. An additional process (called *DWA*) – that implements the integration layer with DB2 for z/OS – runs on the Netezza hardware

and operating system. This integration layer is the entry point for all requests originating either from DB2 or the IDAA stored procedures, which run on System z. The DRDA protocol [DRD03] is used for the communication between both hardware platforms. Requests to execute queries are passed to the Netezza backend by translating DRDA to CLI/ODBC. Administrative requests, e. g., to provide a list of accelerated tables, are handled in DWA itself. If necessary, SQL queries against the Netezza are executed to collect backend-related meta data and/or statistical information.

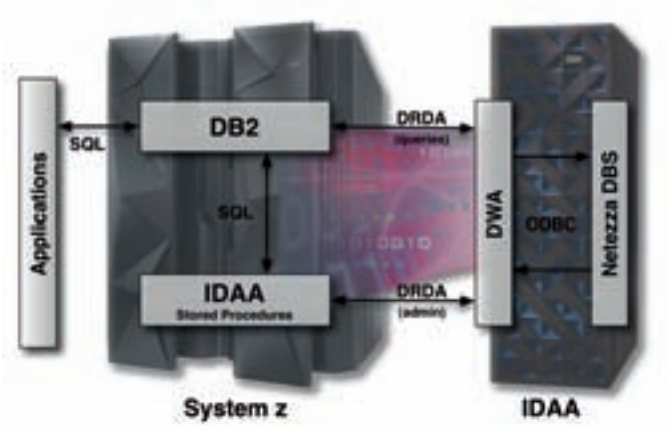


Figure 2: IDAA Architecture

It is possible to associate multiple accelerators with a single DB2 system in order to establish an environment that supports high availability and disaster recovery. The data on all accelerators is managed independently. Similarly, a single accelerator can be connected to multiple DB2 systems. The computing resources of the accelerator are then shared by all DB2 systems. It is possible to set a capping for the resources on the accelerator for each DB2 system individually. An appropriate workload balancing is applied by the Netezza backend if resource limits are reached.

It is the responsibility of the database user/administrator to trigger the refresh of the data in each accelerator individually (or to set up incremental update where appropriate). For query processing, DB2 picks any of the available accelerators that has all the tables accessed by the query. Thus, keeping the accelerated data synchronized on all accelerators is important to guarantee consistent and correct query results.

3.1 Query Processing

For each query, the DB2 optimizer decides whether to execute the query locally in DB2, or to pass it to IDAA. Multiple levels influencing this routing decision exist:

1. A system-wide configuration (zparm) can be set to enable query acceleration, i. e., the usage of an accelerator.

2. The connection between DB2 and the accelerator can be started or stopped. Only connected accelerators are considered by the DB2 optimizer.

3. A SQL session-level setting can be used to control query acceleration using the DB2 special register `CURRENT QUERY ACCELERATION`. The special register value can be changed by a SQL `SET` statement at any point in time. Possible values for the special register are:

   `NONE` Queries will only be evaluated locally by DB2 and no accelerator is considered.

   `ENABLE` Eligible queries, i. e., queries that can be executed syntactically and semantically correct by IDAA, are routed to the accelerator if the DB2 optimizer's heuristics consider the usage of IDAA as beneficial.

   `ENABLE WITH FAILBACK` Same as `ENABLE`, but if the query fails on the accelerator for whatever reason at run-time, DB2 executes the query itself to recover.

   `ELIGBLE` Eligible queries are always routed to the accelerator, but no heuristic checks are applied. Non-eligble queries, e. g., queries using unsupported expressions or accessing non-accelerated tables, are executed locally in DB2.

   `ALL` All queries are routed to the accelerator. In case a query is not eligible, the query will fail with an appropriate SQL code.

If the decision is made to pass on the query to IDAA, DB2 translates the DB2 SQL dialect to the IDAA/Netezza SQL dialect. A new DRDA connection is established between DB2 and IDAA, and DB2, acting as application requestor, sends the translated query. IDAA, acting as application server, returns the corresponding result set to DB2.

3.2 Data Maintenance

The data of accelerated tables in IDAA is a snapshot of the data in DB2. The snapshots have to be refreshed in case the data in DB2 changes. IDAA offers 3 options for the data refresh, which are explained in more detail.

The IBM DB2 Analytics Accelerator comes with an extremely fast loader for whole tables or a set of partitions thereof. Refreshing the data of an entire table (Figure 3) is typically done for rather static data and for non-partitioned tables. Partition-based load (Figure 4) is targeted at partitioned tables, where updates are performed rather infrequently and only to a small subset of partitions. Parallelism is exploited for tables using partition-by-range and also for tables that are defined as partition-by-growth.

If a low latency for the data currency in IDAA is not very important and queries return acceptable results, even if the data is slightly outdated by a few minutes or hours, both

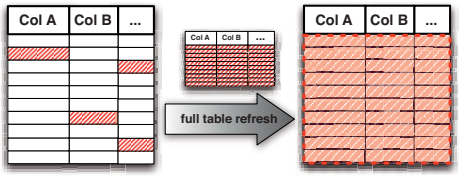


Figure 3: Full Table Refresh

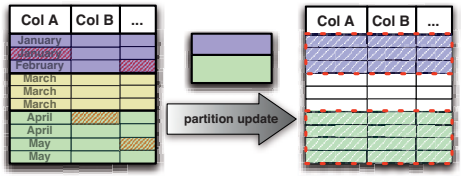


Figure 4: Partition Update

options are viable. Additionally, the initial load of the data into the accelerator is accomplished that way with very good performance, reaching a maximum throughput of 1.5 TB/h. Internally, multiple threads of the `ACCEL_LOAD_TABLES` unload the data from DB2 in the DB2 internal format and send that to the accelerator, i. e., only a minimum of CPU resources is needed on System z. The accelerator parses the DB2 format, converts it, and inserts it in parallel into a shadow table in the Netezza backend.

For tables with a higher update frequency and where a high data currency on the accelerator is desired, a third option for data maintenance is the Incremental Update feature (cf. figure 5). Incremental update is based on replication technology [BNP+12], which reads DB2 logs and extracts all changes to accelerated DB2 tables. Those changes are transmitted to IDAA, where they are applied to the Netezza backend tables. Since Netezza is tailored towards very efficient and high-performing execution of complex queries, and less so for manipulating single rows, batching is used when applying the changes. A latency of about 1 min is achieved for applying the changes to the shadow tables due to this change batching. For reporting systems and data warehouses, 1 min is usually fast enough, especially if complex accelerated queries may take several minutes (or hours) in DB2 and still a few seconds with IDAA.

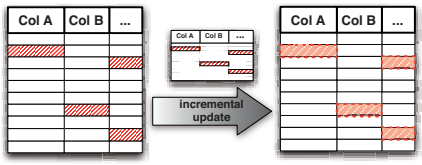


Figure 5: Refreshing Table Data With Incremental Update

4 High Performance Storage Saver

4.1 Basic Design Principles

The design of IDAA HPSS was governed by very few principles. DB2 for z/OS is and remains the one and only point to access and administer the data – regardless of the physical storage location. The DB2 optimizer chooses the best access plan, which is either a local execution in DB2, or an execution of the whole query in the accelerator.

DB2 is the owner of the data and all the administrative tasks like backup/recovery strategies are placed there. In particular, no backup/recovery mechanisms were introduced on the accelerator itself so that customers can rely on their already available DB2 skills.

A primary design goal we had for new interfaces was simplicity and ease of use, e. g., to archive or restore table partitions and to retrieve monitoring information. Since IDAA is an appliance, the amount of information needed for tuning by a database administrator is small to begin with. The accelerator does not have a large tuning layer, which means only distribution and organizing keys can be changed on a table, but no indexes or materialized views or other techniques can be applied. HPSS does not increase the complexity unless absolutely necessary.

In short, the overall guiding rule was to further strengthen the IDAA idea of a fully autonomous appliance.

4.2 Overview

Two new stored procedures establish the main functionality for HPSS (cf. figure 6). The first, `ACCEL_ARCHIVE_TABLES`, handles all the steps necessary for moving of data from DB2 to the accelerator. This includes the removal of the data from the DB2 table and the maintenance of related indexes. `ACCEL_RESTORE_ARCHIVE_TABLES` is the second stored procedure. It implements the reverse process, i. e., restoring the data and rebuilding index and support structures in DB2. For monitoring purposes, the output of the `ACCEL_GET_TABLES_INFO` and `ACCEL_GET_TABLES_DETAILS` stored procedures has been extended to return necessary archiving-related information. The final piece is the newly introduced DB2 special register `CURRENT GET_ACCEL_ARCHIVE`. It is set by DB2 client applications and used by the DB2 optimizer – together with the information in the catalog table `SYSACCELERATEDTABLES` – to decide whether to include archive data in the query (if available) or not.

Inherent to this design is that all interfaces related to HPSS are accessible via SQL. While calling stored procedures is well established in every major relational database system and very beneficial for automation purposes, it is by no means very end-user friendly. The IBM DB2 Analytics Accelerator Studio is the graphical user interface (GUI) for IDAA. Most administrative operations that are necessary for IDAA can be done via the GUI, and moving data to the IDAA online archive has been integrated, too.

Archiving and restoring with HPSS operate on table partitions. Only all the data of a partition can be moved to the accelerator and is pruned from the DB2 table. This granularity was chosen in order to simplify the implementation, improve performance, and to reduce locking and logging overhead that would be inherent to row-level granularity. DB2 utilities are exploited with their low-level access to the DB2 data manager.

DB2 for z/OS supports two types of partitioning: range-based partitioning assigns data rows to partitions based on the values of a partitioning key column (e. g., a column of type `DATE`) while growth-based partitioning automatically adds storage as new partitions to the table as the volume of data grows. With growth-based partitioning, no fixed association of

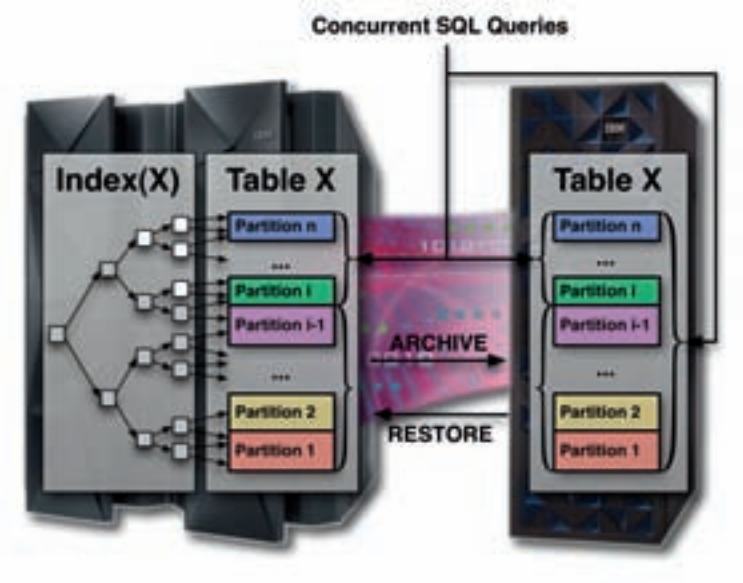


Figure 6: HPSS Interfaces

rows to partitions exists. The database system is free to shuffle rows between partitions to fill gaps in the physical layout, e. g., when the table is reorganized. For HPSS, we need an explicit, data-dependent and deterministic decision whether a row should be archived or not. Therefore, support is limited to range-partitioned tables.

In the following, we describe in more detail how the different pieces of the solution play together.

4.3 Moving Data to HPSS

All tables handled by an accelerator must have been previously registered to it. The registration copies schema information (columns, data types, ...) from the DB2 catalog to the accelerator. In fact, the registration triggers the creation of a schema-compatible shadow table in the Netezza backend system that is managed solely by IDAA. So exploiting an accelerator A as online archive for a DB2 table T requires that T has already been registered with A.

The process of archiving data has to prepare for an eventual loss and subsequent recovery of the archived data. Since there are no backup/recovery mechanisms on the accelerator itself, mechanisms in DB2 for z/OS and System z itself are put to the task. Furthermore, the design is such that all archived data is copied from DB2 to the accelerator, even if that data has previously been loaded in the accelerator. This *guarantees* that the archive data is exactly the same as it was in DB2.

The input for the ACCEL_ARCHIVE_TABLES stored procedure is an XML document with a sequence of tables and a list of partitions for each of them. The partitions list allows abbreviations like "first n partitions" or "all partitions except last m". A typical use case would be a table partitioning based on a single DATE column with ascending key ordering, such that the first partition contains the oldest data and the last partition the newest. In this case, the "first n" criterion corresponds to a SQL predicate WHERE *partitioning_column <= end_date_for_partition_n*. An example XML document is shown in listing 1. It specifies to archive partitions 1 thru 9 (inclusive) for table HPSS.STORE_SALES.

```
<dwa:tableSetForArchiving version="1.0"
         xmlns:dwa="http://www.ibm.com/xmlns/prod/dwa/2011">
     <table schema="HPSS" name="STORE_SALES">
          <partitions>1:9</partitions>
     </table>
</dwa:tableSetForArchiving>
```

Listing 1: XML Document for Archiving

The data for each of the partitions identified in the XML document is archived. If a partitions has already been archived before (and was not restored in the mean time), that partition is skipped because the accelerator already has the archive data. This silent toleration of archived partitions simplifies automation processes, e. g., for a monthly archiving of all partitions, except the partitions for the last 3 by using a fixed range $1 : -4$.

DB2 utilities are employed to achieve very high performance for the archiving process by by-passing much of the relational processing inside DB2 and to reduce the logging and locking overhead. The following major steps are applied for each partition:

1. Lock the table partition in shared mode to prevent concurrent updates on the data.
2. Create a new backup (called *image copy*) of the partition using DB2's COPY utility.
3. Copy the data from the partition to the accelerator using the UNLOAD utility.
4. Commit the archive data in the accelerator.
5. Prune the partition in DB2 using the LOAD REPLACE utility with an empty input data set.

DB2 customers have well-established procedures for backing up their data. The introduction of IDAA into the customer environment in general (and HPSS in particular) should not impact these procedures. Backups are always based on the data in DB2. Thus, recovering any data (should that become necessary) is always based on those backups and not on the accelerator. So even after data has been purged from the DB2 table, DB2 for z/OS and System z with its superior security and reliability qualities guarantees that data can been recovered in case of system, storage, or site failures.

While several mechanisms and third-party utilities exist for backing up DB2 databases on System z, the most common mechanism is the COPY utility. So we based our approach

for HPSS on that. The COPY utility takes a backup of DB2 table data at data set level and logs that action in a DB2 catalog table to allow an automated restore in combination with log replay. Of course, the COPY utility provides a wide variety of options to influence the placement of backups and the processing behavior.

In the context of IDAA, we wanted to avoid cluttering the interface of the stored procedure ACCEL_ARCHIVE_TABLES with a multitude of (possibly never needed) options. Therefore, the only input that has to be specified is a *high-level qualifier* (HLQ), identifying the location within the file system where the backup data sets are to be placed. The HLQ is specified as a global environment property that must be set by the administrator. A naming convention is automatically applied to merge the names of the DB2 subsystem, the database, the tablespace, and the partition identifier to build the final data set name for the backup image. It is mandatory that the automated disk space management facility of System z, i. e., system managed storage (SMS) with corresponding automatic class selection (ACS) routines, is set up to correctly handle the backup data sets, which will be allocated under the defined HLQ. In particular, to realize the space saving potential of HPSS, SMS must be set up such that these backups are placed on less expensive disk storage.

An important aspect for the administrator is that the backup images created by HPSS must exist independently of backup copies that are created by other automated regular backup processes. Usual backup procedures of customers retain a certain number of backup levels. At some point, old backups are discarded by deleting the data set and purging the DB2 catalog table using the MODIFY RECOVERY utility. The backup images created by HPSS must not be discarded, however. These backups have a different semantics because they are the source for restoring the actual customer data back into DB2, while other backups are only needed to recover the DB2 table to a certain point in time in case of system or storage failures.

After backup creation, the data of an archive partition is transferred to the accelerator. Data transfer uses the same mechanisms as the ACCEL_LOAD_TABLES stored procedure, i. e., the DB2 UNLOAD utility is employed to read the data in parallel using multiple threads. The archive data is read from the backup data set so that the same data can be send to multiple accelerators. Since archiving is a destructive operation on the DB2 table, it is not possible to read the data again from the DB2 table; but the backup data set holds the master data and can be used as primary source.

On accelerator side, the archive data is inserted into a different shadow table in the Netezza backend database than regular, non-archived data. A view combines the rows of both tables, and DB2 uses this view in rewritten queries. This gives IDAA the flexibility to choose a different suitable storage structure in the future. Both shadow tables (cf. figure 7) use a hidden column identifying the DB2 partition from which each row was loaded, thus allowing efficient detection and management of all data that corresponds to a specific DB2 partition.

Finally, archived data is pruned from the DB2 table using the LOAD REPLACE utility to overwrite the partition data from an empty data set. The utility provides a very efficient means of deleting an entire partition, by-passing the DB2 transaction log. An effect of the utility is that the data set for the partition is redefined in the file system, i. e., deleted,

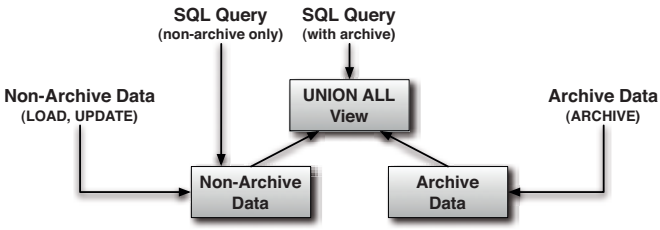


Figure 7: Shadow Tables for Archived and Non-Archived Data

re-allocated, and re-initialized. The re-allocation effectively reduces the size of the data set to the minimum possible allocation (primary extent), so that the disk space for the archived partition is actually freed. The partitioning definition of the table remains unchanged in the DB2 catalog, including the archived partitions, which are now empty. An unchanged partition definition is required for a potentially needed restore of the partition backup image.

The partition is then marked as archived in the accelerator meta-data and subsequent attempts to load or archive this partition in the future will be ignored. The content of archived partitions is effectively frozen. For DB2, those partitions still exist without any data, so SQL statements like `UPDATE` or `DELETE` have no effect. If any data is subsequently inserted into these partitions in DB2, it will not be reflected on accelerator side. Setting the partition to read-only state in DB2 prevents such situations. Of course, the policies and ETL logic of the data warehouse should already guarantee that archived data ranges remain unchanged. Otherwise, using HPSS is not appropriate.

For changing the partitioning of a table, DB2 supports the addition of new partition and the rotation of existing ones. Adding new partitions is straight-forward since any data in new partitions is treated as non-archived data, which can later be archived. Rotating a partition is logically equivalent to deleting the partition (typically the one containing the oldest data) and then adding an new partition at the end of the existing partition key range. If the partition being rotated has previously been archived, a call to the `ACCEL_ARCHIVE_TABLES` stored procedure synchronizes the data in the accelerator and deletes the matching rows in the archive shadow table.

4.4 Restoring Data from HPSS

The anticipated, typical usage scenario for HPSS is the movement of data from DB2 to the accelerator – not vice versa. Historical and archived data that will not be changed and is not needed any more for most of the query processing (at least for transactional queries). It can be backed up on inexpensive storage and moved to the accelerator where it remains available for occasional analytic processing. However, there are some situations where it may become necessary to restore the archived data back into DB2 tables.

It must be possible to restore data when a system or site failure involving the accelerator has destroyed the archive data. By design, IDAA does not capture backups on accelerator side and rather relies on DB2 for z/OS. Another accelerator may have been used to store a second copy of the archive data to avoid such situations. However, not all customers may use a second IDAA. In addition, it may turn out that the decision to archive some or all partitions of a table in HPSS was premature and direct access in DB2 is required again. For example, business reasons may demand modifications to history data, or queries may have to be applied to the archive data, which are not (yet) supported by IDAA.

The stored proceduce ACCEL_RESTORE_ARCHIVE_TABLES can be used to restore data of archived partitions. Its main task is to automate the DB2 utility calls that are needed for recovering the image copy and to purge the archive data in IDAA. This includes executing utilities to check data consistency (i. e. re-validate constraints like unique and referential constraints) and to rebuild indexes. On the accelerator, the archived data is directly moved back into the shadow table for the non-archived data and the corresponding catalog information is updated. The stored procedure is currently in prototype stage and will be made generally available in IDAA in the near future.

4.5 Monitoring Archived Data

For monitoring purposes, the output of stored procedures ACCEL_GET_TABLES_INFO and ACCEL_GET_TABLES_DETAILS has been extended to return archiving-related information at table and partition level, respectively. At table level, HPSS provides information whether an accelerated table has an archive, and how much DB2 data it contains, measured by the number of rows and bytes. A sample output is shown in listing 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<dwa:tableInformation version="1.2" xmlns:dwa="http://www.ibm.com/
    xmlns/prod/dwa/2011">
<table schema="HPSS" name="STORE_SALES">
   <status loadStatus="Loaded" accelerationStatus="true"
         integrityStatus="Unimpaired" archiveStatus="true" />
   <statistics usedDiskSpaceInMB="1" rowCount="2000"
         archiveDiskSpaceInMB="100" archiveRowCount="10000"
         skew="0.3" organizedPercent="95.00"
         lastLoadTimestamp="2012-09-20T11:53:27.997141Z" />
</table>
</dwa:tableInformation>
```

Listing 2: Sample Output of ACCEL_GET_TABLES_INFO

At partition level, the output lists the timestamp when a partition was archived, how much archive data was transferred from DB2 to IDAA, and the name of the backup data set on System z. Listing 3 shows an example for a single table with just 4 partitions, which represent the quarters of 2012. The partitions with logical partition number 1 and 2 have been archived, while partition 3 holds non-archived data only and partition 4 is actually empty. The GUI visualizes the information provided by both stored procedures (cf. figure 8).

```
<?xml version="1.0" encoding="UTF-8"?>
<dwa:tableSetDetails version="1.0"
         xmlns:dwa="http://www.ibm.com/xmlns/prod/dwa/2011">
<table name="HPSS" schema="STORE_SALES">
   <partInformation type="BY_RANGE">
      <column name="SS_TICKET_NUMBER"/>
   </partInformation>
   <part logicalPartNr="1" dbmsPartNr="1" endingAt="40000">
      <archiveInformation dataSizeInMB="120"
             archiveTimestamp="2012-10-02T03:18:03.120943Z">
        <backupImage>ARCHIVE.HPSSDB.SALES.P0001</backupImage>
      </archiveInformation>
   </part>
   <part logicalPartNr="2" dbmsPartNr="2" endingAt="60000">
      <archiveInformation dataSizeInMB="2200"
             archiveTimestamp="2012-10-02T03:19:08.983261Z">
        <backupImage>ARCHIVE.HPSSDB.SALES.P0002</backupImage>
      </archiveInformation>
   </part>
   <part logicalPartNr="3" dbmsPartNr="3" endingAt="80000">
      <changeInformation category="NONE" dataSizeInMB="18"
             lastLoadTimestamp="2012-10-10T11:53:27.997141Z"/>
   </part>
   <part logicalPartNr="4" dbmsPartNr="4" endingAt="100000">
      <changeInformation category="NONE" dataSizeInMB="0"
             lastLoadTimestamp="2012-10-12T11:53:27.997141Z"/>
   </part>
</table>
```

Listing 3: Sample Output of ACCEL_GET_TABLES_DETAILS

4.6 Query Processing

For each query entering DB2, the DB2 query optimizer checks if all of the following conditions hold true (in the non-archiving context):

- Query acceleration is enabled.
- All tables referenced by the query are loaded on an accelerator.
- The query qualifies for routing (e. g., only constructs supported by the accelerator's query processor are used).
- Heuristics (cf. section 3.1) indicate that the query will be executed faster on the accelerator (in particular, it's an analytic query and not a transactional one).

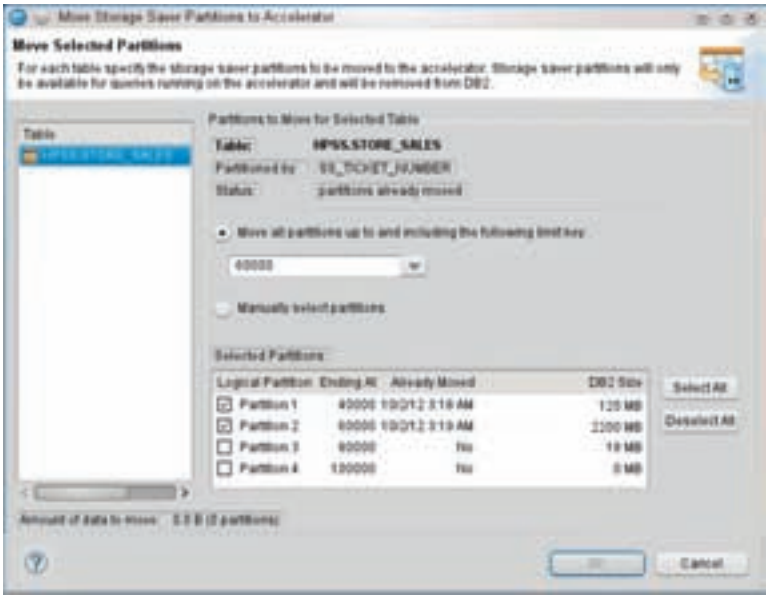


Figure 8: GUI Screenshot Highlighting Archive Information

If at least one condition is not satisfied, the query is handled in DB2. If all checks are passed, the query is rewritten into the SQL dialect supported by the accelerator (the Netezza dialect), and the query is passed on to the accelerator and executed there. The accelerator solely relies on the data in the shadow table(s) in Netezza. Query results are passed back to DB2, which returns it as-is to its client application.

In short, where the whole query is executed is a binary decision. Admittedly, those rules are simple, but that makes them very robust and attractive for customers. This is a very light-weight variation of federated technologies [ISO03]. In particular, DB2 – as the federated server – does not attempt to compensate missing functionality. Also, only heuristics are used to come up with a routing decision and no fully-fledged costing is applied. The advantage of this approach is that applying heuristics is much faster and, thus, any impact on OLTP workload is significantly reduced.

With HPSS, query processing needs to take into account whether the query should include archived data of the involved tables. For queries that involve non-archived data only, the query optimizer usually has the freedom to decide whether the query should be routed to the accelerator or not, as outline above. Assuming the data in the accelerator has been maintained properly, both execution paths return the same result. On the other hand, queries involving archive data must *always* be executed on the accelerator; running them in DB2 would produce incorrect results due to missing rows.

At first sight, it may seem desirable to let the DB2 optimizer and/or DB2 runtime decide whether a query may involve archived data and handle it accordingly. However, for many cases it cannot be easily determined *a-priori* that archived data is or is not in scope for the

query. For example, predicates on functionally dependent columns may apply the necessary filtering, or the scanning of a dimension table may reduce the qualifying rows on a fact table to the non-archived data only, possibly over a cascade of joins. Another aspect are index-only queries, for which no access to the archived partitions of the tablespace occurs. The index itself has no knowledge about values for archived rows, especially if the index is non-partitioning. The consequence is that any query where such doubts arise, would have to be routed to the accelerator. Since this applies to a majority of the queries, the optimizer would essentially loose the freedom to determine the "optimal" place for query execution (DB2 or accelerator). While it would be possible to require specific filtering predicates involving the partitioning key columns, this is not only unrealistic, it also voids the key design principle of transparency for existing applications.

Therefore, the decision whether a query should or should not include archived data is explicitly made by exposing the new special register `CURRENT GET_ACCEL_ARCHIVE` in DB2 with possible values YES and NO. The solution is not fully transparent to client code, but is much more flexible since it allows explicit control at several levels by configuring a system-wide default value, setting it at the connection level (e. g., in the JDBC connection string), or allowing application developers to switch the semantics within an established SQL session at will.

If `CURRENT GET_ACCEL_ARCHIVE` is set to YES, then any query that includes a table which has archive data *must* be executed on the accelerator. The DB2 optimizer heuristics do not even have to be checked. To easily detect the presence of archive data from DB2 query processing, the DB2 catalog table `SYSACCELERATEDTABLES` carries a new column `ARCHIVE`. The values in this column indicate the presence or absence of archive data for this table, and the IDAA stored procedures maintain that. If the query cannot run on the accelerator, e. g., because it references tables that are not accelerated or because it uses SQL constructs that are not yet understood by IDAA, then the query will fail with a dedicated SQL code.

If the query can be routed and needs to access archive data, DB2 rewrites the query against the view (cf. figure 7) in the accelerator, which combines the data from the shadow table holding non-archived data with the shadow table for the archive data. The view employs a `UNION ALL` operator, which is evaluated in the Netezza backend only. Thus, the physical separation of the data is transparent to DB2. Furthermore, we retain the binary decision for the location of query processing. This avoids the inherent complexity of partial query offloading if, e. g., only the archived data would reside in IDAA while the non-archived data remains in DB2 only.

If `CURRENT GET_ACCEL_ARCHIVE` is set to NO, query processing remains as before without HPSS and only works on the shadow tables containing the non-archived data. That is true even if archive data for the same tables is present on the accelerator.

4.7 Imposed Limitations

Our implementation intially imposes several restrictions for tables that are moved to IDAA HPSS. A fundamental restriction is that DB2 is unable to handle referential constraints if part of the referenced data is not available. Therefore, tables that are referred to by a foreign key in the same or another table may not be archived. Other restrictions are rooted in the backup strategy that we have chosen. Since the "master data" for archived partitions are the backups, tables that have LOB and XML data columns are not allowed. Those objects are stored in separate tablespaces, which are not yet included in backup copies that the `ACCEL_ARCHIVE_TABLES` procedure creates.

No table or tablespace modifications may be performed that prevent restoring the backup images. For example, modifications of physical table space properties like DSSIZE would cause the `RECOVER` utility to fail. Such modifications require that all archived data is restored first and archived again after the modification is done.

Naturally, any restrictions that IDAA has in general also apply to HPSS. So if a table has columns with data types that cannot be loaded into the accelerator, a projection of the table is used. This applies, for example, to the data types `DECFLOAT` and `ROWID`. While the data of such tables can be archived, those columns are omitted and queries including the archive must not involve those columns.

4.8 HPSS and Data Maintenance

Section 3.2 described the options that can be used to maintain data of accelerated tables. Archiving introduces another option, even if it only applies to a part of the table data. For any given table, only one of the data maintenance options can be used concurrently. Thus, if some data shall be archived, it is not possible to load all or parts of the non-archived data using the `ACCEL_LOAD_TABLES` stored procedure at the same time. Likewise, replication has to be stopped or disabled for the table while an archive or restore operation is in progress.

Incremental update via replication merits a closer look. It is supported to archive some part of a DB2 table to IDAA and to apply replication on the non-archived data. Using replication for the archive data is not necessary since the archive data won't change. Purging the archive data in DB2 uses the `LOAD REPLACE` utility. This avoids the logging of deleted rows. The replication setup is configured to ignore log entries for the utility executions. So, after an archiving operation completes, replication can be re-started and will continue to propagate changes on the non-archived data from the DB2 log position where it left off when it was stopped before archiving. That means, changes to non-archived data that occurred while data was being moved to HPSS will be propagated correctly. The purged data will not be propagated, however.

5 Evaluation

We conducted a set of performance tests during the development of HPSS. The new functionality should deliver the same results and acceleration factors that customers have come to expect from IDAA.

An important aspect was the impact of the two shadow tables and the UNION ALL view for query processing. For example, the optimizer of the Netezza backend has to detect the union and push down filtering predicates into both branches of the set operation if possible. It may also have an influence on join order, broadcasting (intermediate) tables, and so on. Some performance issues have been identified in that area, and improvements were made. As figure 9 illustrates, the resulting query performance is very close to the performance of the same queries without any data being archived, i. e., all the data residing in a single table in the accelerator. The LINEITEM table of the TPC-H benchmark was used, and about half of the table's data was archived. Since the data is evenly distributed across the table partitions, 50% of all partitions was in the archive. The total TPC-H data volume was 200 GB. The relative differences (cf. figure 10) are comparatively small. Some queries actually run slightly faster because less data resides in the base tables and more parallel processing in the Netezza backend can be exploited. Other queries take a bit longer, e. g., in case the push-down of predicates into both branches of the UNION ALL is not always applied and also due to the the increased query complexity. Overall, the known advantages of IDAA with acceleration factors of up to 2000x can be achieved with HPSS as well.

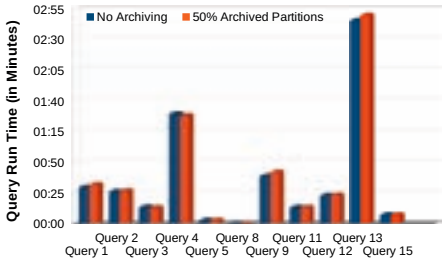


Figure 9: Query Performance Comparison

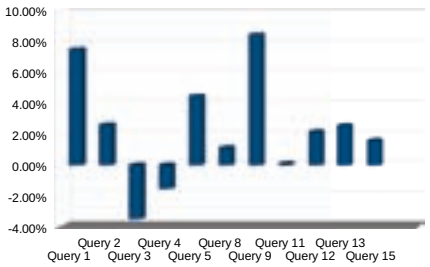


Figure 10: Relative Difference

We also measured the performance for moving data from DB2 to the accelerator. The upper limit for the throughput is defined by `ACCEL_LOAD_TABLES`, which just copies data from DB2 to the accelerator. Typically, a throughput of 1 TB/h can be achieved; in lab environments, we measured an even higher throughput of 1.5 TB/h. For archiving, additional steps need to be performed. A commit point occurs after each partition, which implies that at most 64 GB of data can be streamed and then the data flow pipeline stalls. This has a very minor and negligible impact only. However, additional DB2 utilities are executed in order to create the backup and to prune the data from the DB2 table. The data pruning maintains indexes in DB2, and figure 11 shows the impact of this. Archiving 50% of the LINEITEM table in a TPC-H schema is only slightly slower than loading the data if no indexes are defined. The overhead for the executing the additional utilities is between a

few percent only. Those indexes are maintained by the `LOAD REPLACE` utility. The more indexes are defined on a table the higher the overhead gets. The 200 GB scenario with just 3 indexes – one index is a partitioning index (includes the partitioning key column) while the other two indexes are non-partitioned – shows that the execution time for 50% of the data jumps from 10 min to nearly 30 min. All additional time is spent by the utility.

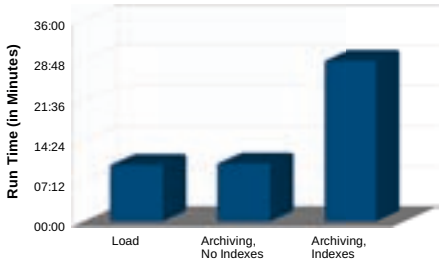


Figure 11: TPC-H 200 GB

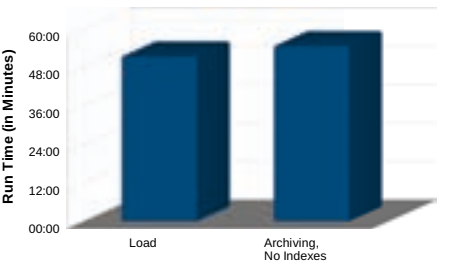


Figure 12: TPC-H 1 TB

The archiving performance with a (more typical, but still small) data volume of 1 TB is depicted in figure 12. Loading 50% of the table data scales linearly, and so does archiving. The overhead for committing after each partition and the additional utility execution stays in the single-digit percentage range. The figure does not show the case with indexes being defined on the table because the behavior with indexes is the same as in figure 11. We only want to highlight the small gap between loading and archiving the data.

6 Summary and Outlook

In this paper we have presented the High Performance Storage Saver, a new feature of the IBM DB2 Analytics Accelerator. It enhances the product to use IDAA as Online Archive, with extremely good query performance on querying the archived and non-archived data. Thus, a completely new use case for accelerator technology is established, which takes the next step towards an integration of OLTP and OLAP systems, at schema and data level. We described the interfaces to archive data from DB2 for z/OS to IDAA and to restore it back. Query processing relies on a new DB2 special register for the decision whether archive data shall be considered in the query or not. The just released IDAA Version 3 offers this functionality to customers.

Since this is a new feature, there is still room for improvements. The idea is to largely base further enhancements on feedback we already have received and will receive from customers. For example, the restore functionality is rather basic and its scope shall be broadened, e. g., to restore data back into DB2 based on the data in the accelerator. Similarly, the support for multiple accelerators is still in a prototype stage and will be productized. Long term, it is desirable to automate the decision for exploiting the IDAA online archive based on query semantics.

7 Trademarks

References

[BBF+12] P. Bruni, P. Becker, W. Favero, R. Kalyanasundaram, A. Keenan, S. Knoll, N. Lei, C. Molaro, and PS Prem. *Optimizing DB2 Queries with IBM DB2 Analytics Accelerator for z/ OS*. IBM Redbooks, 2012. http://www.redbooks.ibm.com/abstracts/sg248005.html.

[BNP+12] A. Beaton, A. Noor, J. Parkes, B. Shubin, C. Ballard, M. Ketchie, F. Ketelaars, D. Rangarao, and W.V. Tichelen. *Smarter Business: Dynamic Information with IBM InfoSphere Data Replication CDC*. IBM Redbooks, 2012. http://www.redbooks.ibm.com/abstracts/sg247941.html.

[DRD03] The Open Group. *DRDA V5 Vol. 1: Distributed Relational Database Architecture*, 2003.

[Fra11] P. Francisco. *The Netezza Data Appliance Architecture: A Platform for High Performance Data Warehousing and Analytics*. IBM Redbooks, 2011. http://www.redbooks.ibm.com/abstracts/redp4725.html.

[HBB+01] D. Hewgill, A. Balingit, M. Bruegger, W. Postl, and J. Thompson. *Backing Up DB2 Using IBM Tivoli Storage Management*. IBM Redbooks, 2001. http://www.redbooks.ibm.com/abstracts/sg246247.html.

[IBM10] IBM. IBM DB2 Near-Line Storage solution for SAP NetWeaver BW. Technical report, 2010. ftp://public.dhe.ibm.com/common/ssi/ecm/en/nis03001usen/NIS03001USEN.PDF.

[IBM12] IBM. IBM Banking and Financial Markets Data Warehouse V8.5. Technical report, 2012. http://www.ibm.com/software/data/industry-models/financial-markets/.

[Inm99] W. Inmon. *Building the Operational Data Store*. John Wiley & Sons, 1999.

[ISO03] ISO/ IEC 9075-9:2003. *Information Technology – Database Languages – SQL – Part 9: Management of External Data (SQL/ MED)*, 2nd edition, 2003.

[Leh03] W. Lehner. *Datenbanktechnologie für Data-Warehouse-Systeme – Konzepte und Methoden*. Dpunkt Verlag, 2003.

[Ora11] Oracle. Creating a Hierarchical Database Backup System using Oracle RMAN and Oracle SAM QFS with the Sun ZFS Storage Appliance. Technical report, 2011. http://www.oracle.com/technetwork/server-storage/sun-unified-storage/documentation/db-backup-samfs-rman-457121.pdf.

[Sch01] R. Schaarschmidt. *Archivierung in Datenbanksystemen – Konzept und Sprache*. PhD thesis, Database and Information Systems Group, University of Jena, Germany, 2001. (in German).

The Graph Story of the SAP HANA Database

Michael Rudolf[1], Marcus Paradies[1], Christof Bornhövd[2], and Wolfgang Lehner[1]

[1]SAP AG; Dietmar-Hopp-Allee 16; Walldorf, Germany
[2]SAP Labs, LLC; 3412 Hillview Avenue; Palo Alto, CA, 94304
eMail: {michael.rudolf01, m.paradies, christof.bornhoevd, wolfgang.lehner}@sap.com

Abstract: Many traditional and new business applications work with inherently graph-structured data and therefore benefit from graph abstractions and operations provided in the data management layer. The property graph data model not only offers schema flexibility but also permits managing and processing data and metadata jointly. By having typical graph operations implemented directly in the database engine and exposing them both in the form of an intuitive programming interface and a declarative language, complex business application logic can be expressed more easily and executed very efficiently. In this paper we describe our ongoing work to extend the SAP HANA database with built-in graph data support. We see this as a next step on the way to provide an efficient and intuitive data management platform for modern business applications with SAP HANA.

1 Introduction

Traditional business applications, such as Supply Chain Management, Product Batch Traceability, Product Lifecycle Management, or Transportation and Delivery, benefit greatly from a direct and efficient representation of the underlying information as data graphs. But also not so traditional ones, such as Social Media Analysis for Targeted Advertising and Consumer Sentiment Analysis, Context-aware Search, or Intangible and Social Asset Management can immensely profit from such capabilities.

These applications take advantage of an underlying graph data model and the implementation of core graph operations directly in the data management layer in two fundamental ways. First, a graph-like representation provides a natural and intuitive format for the underlying data, which leads to simpler application designs and lower development cost. Second, the availability of graph-specific operators directly in the underlying database engine as the means to process and analyze the data allows a very direct mapping of core business functions and in turn to significantly better response times and scalability to very large data graphs.

When we refer to data graphs in this paper, we mean a full-fledged property graph model rather than a subject-predicate-object model, as used by most triple stores, or a tailored relational schema, for example in the form of a vertical schema, to generically store vertices and edges of a data graph.

A property graph [RN10] is a directed multi graph consisting of a finite (and mutable) set

Category

ID	Name
1	Books
2	Literature & Fiction
3	Movies & TV
4	Movies

Rating

ID	Value	Product
1	4	1
2	3	2
3	4	3
4	5	3

part of

C_ID1	C_ID2
2	1
4	3

Product

ID	Title	Year	Type
1	Romeo and Juliet	2012	Book
2	Romeo and Juliet	1997	DVD
3	Shakespeare in Love	1999	DVD

in

P_ID	C_ID
1	2
2	4
3	4

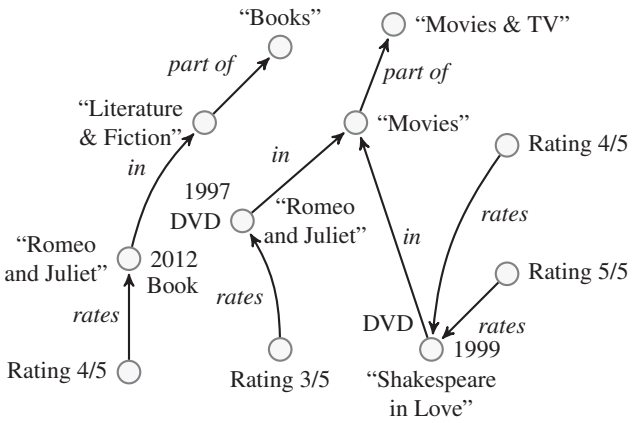


Figure 1: Example data expressed in the relational and the property graph data model

of vertices (nodes) and edges (arcs). Both, vertices and edges can have assigned properties (attributes) which can be understood as simple name-value pairs. A dedicated property can serve as a unique identifier for vertices and edges. In addition, a type property can be used to represent the semantic type of the respective vertex or edge. Properties of vertices and edges are not necessarily determined by the assigned type and can therefore vary between vertices or edges of the same type. Vertices can be connected via different edges as long as they have different types or identifiers.

Figure 1 shows a very small example data set both expressed in the relational model and the property graph model, which could be the basis for a Targeted Advertisement application. Customers can rate products, which are organized in categories. If, additionally, the relationships between customers, the products they bought, and their ratings are stored, the application can easily recommend products that might be of interest to the customer based on what other customers bought and rated.

The property graph model provides the following key characteristics, which distinguish it, in particular, from the classical relational data model.

- **Relationships as First Class Citizens.** With the property graph model relationships between entities are promoted to first class citizens of the model with unique identity, semantic type, and possibly additional attributes. The relational model focuses on the representation of entities, their attributes and relational consistency constraints between them and requires the use of link tables to represent n-to-m relationships or additional attributes of relationships. In contrast, the concept of an edge provides an explicit and flexible way to represent interrelationships between entities which is essential if relationships between entities are very important or even in the center of the processing and analysis of the data.

- **Increased Schema Flexibility.** In a property graph edges are specified at the instance and not at the class level, i.e., they relate two specific vertices, and vertices of the same semantic types can be related via different types of edges. Similarly, properties of edges and vertices are not necessarily determined by the semantic type of the respective edge or vertex, which means that edges or vertices of the same semantic type can have assigned different sets of properties.

  With this the schema of a data graph does not have to be predefined in the form of a rigid schema that would be cumbersome and expensive to modify but rather evolves as new vertices are created, new properties are added, and as new edges between vertices are established.

- **No Strict Separation between Data and Metadata.** Vertices and edges in a graph can have assigned semantic types to indicate their intended meaning. These types can be naturally represented as a tree (taxonomy) or graph (ontology) themselves. This allows their retrieval and processing as either type definitions, i.e., metadata, or (possibly in combination with other vertices) as data. By allowing to treat and use type definitions as regular vertices we can give up a strict and for some applications artificial separation of data from metadata.

  For example, in the context of context-aware search a given search request can be extended or refined not only by considering related content (i.e., vertices that are related to vertices directly referred to by the request) but also related concepts or terms (i.e., vertices that are part of the underlying type system used in the search).

In recent years, another graph model has gained a lot of popularity: the Resource Description Framework (RDF [CK04]). At its core is the concept that statements about resources can be made in the form of triples consisting of a subject, a predicate and an object. The subject and the predicate are always resources, whereas the object of such a statement can be either a resource or a literal. This simple concept, with almost no further constraints, offers an extremely flexible way of representing information – and hence heavily depends on what conventions individual applications use to encode and decode RDF data. All triples of a dataset form a labeled graph, which represents a network of values. An entity is decomposed into a set of statements and application logic is required to reassemble them

upon retrieval. In contrast, the property graph model provides intrinsic support for entities by permitting vertices and edges to be attributed. RDF therefore does not offer inherent means to represent an entity as a unit and requires applications to provide this semantics.

The use of a dedicated set of built-in core graph operators offers the following key performance and scalability benefits.

- **Allow Efficient Execution of Typical Graph Operations.** An implementation of graph operators directly in the database engine allows the optimization of typical graph operations like single or multi-step graph traversal, inclusive or exclusive selection of vertices or edges, or to find the shortest or all paths between vertices. Such optimizations are not possible in for example relational database systems since the basic operators are unaware of concepts like vertex and edge. In particular, depending on the physical representation of graph data in the system vertices can act like indexes for their associated vertices which allow the performance of graph traversals to be independent of the size of the overall data graph. In contrast, the realization of traversal steps in a relational database system requires join operators between tables whereby the execution time typically depends on the size of the involved tables.
- **Provide Support for Graph Operations Difficult to Express in SQL.** Similarly, the direct implementation of graph-specific operations in the database allows the support of operations that otherwise are very hard or even impossible to express for example in standard SQL. Relational databases are good at straight joins but are not good or are unable to execute joins of unpredicted length that are required to implement transitive closure calculations in graph traversals. Another example is sub-graph pattern matching, which is very difficult to express in general with the means of standard SQL.

In this paper we describe how we extended the SAP HANA [FCP+12] database with native graph data support. In the following section we present different classes of business applications and how they benefit from a dedicated graph support in the database engine. The key components of our technology in the context of the SAP HANA database architecture are introduced in Section 3. Section 4 details the graph data model and our declarative query and manipulation language WIPE, and Section 5 presents the underlying graph abstraction layer and the graph function library. In Section 6 we exemplary evaluate the performance of our approach compared to the traditional SQL-based implementation. Finally, Section 7 summarizes the presented work.

2 Use Cases

In the following paragraphs we illustrate the use of the property graph model and a dedicated graph database management system by different business applications.

Transportation and Logistics. Transportation and logistics are important components of supply chain management. Every company that sells goods relies on materials or products being transported via motor carrier, rail, air or sea transport from one location to another. Therefore, accurate representation and management, as well as visibility into their transportation options and logistics processes are vital to businesses. A typical scenario would include both inbound (procurement) and outbound (shipping) orders to be managed by a transportation management module which can suggest different routing options. These options are evaluated and analyzed with the help of a transportation provider analysis module to select the best route and provider based on cost, lead-time, number of stops, risk, or transportation mode. Once the best solution has been selected, the system typically generates electronic tendering and allows to track the execution of the shipment with the selected carrier, and later supports freight audit and payment. A graph data model supports a flexible and accurate representation of the underlying transportation network. Efficient graph operations enable the fast execution of compute intensive graph operations like identification of shortest or cheapest paths or multi-stop transportation routes.

Product Batch Traceability. End-to-end product traceability is key in global manufacturing to monitor product quality and to allow efficient product recall handling to improve customer safety and satisfaction. It supports complete product batch tracing of all materials purchased, consumed, manufactured, and distributed in the supply and distribution network of a company. Backward traceability allows companies to identify and investigate problems in their manufacturing process or plants as well as in their supply chain. Forward traceability, on the other hand, allows to respond fast to encountered problems to comply with legal reporting timelines, and to minimize cost and corporate risk exposure. A graph data model allows for a direct and natural representation of the batch relation network. Graph processing capabilities in the data management layer are a prerequisite to guarantee fast root cause analysis and to enable timely product recalls and withdrawals as required by law in many industries.

Targeted Advertisement. The goal of targeted advertising is to deliver the most relevant advertisement to target customers to increase the conversion rate of customers who see the advertisements into actual buyers. Decisions of which advertisements to send to which customers can be done based on user profile, behavior, and social context. This matching process includes, in particular, customer segmentation or the creation of personas (like "sports car fan") based on social and interest graphs that describe who the respective user knows or follows and what the user has shown interest in or likes. This information can be derived from publicly available sources that people volunteer or captured by opt-in applications, like Facebook interests, product reviews or blogs, or what they tweet or re-tweet. A data graph model and data graph processing capabilities support the flexible combination of data from the multitude of relevant sources and allows an efficient representation and management of large and frequently changing social graphs. Fast graph analytics operations on this data are a prerequisite to enable large-scale real-time targeted advertisement.

Bill of Materials. Complex products are usually described with the help of a hierarchical decomposition into parts, sub-components, intermediate assemblies, sub-assemblies and raw materials together with the quantities of each, a so-called bill of materials (BOM [ISO12]). Manufacturing industries, such as the automotive and aeronautics sectors, use BOMs to plan the assembly processes. Two important operations on BOMs are linking pieces to assemblies ("implosion") and breaking apart each assembly into its component parts ("explosion"). Since hierarchies are directed acyclic graphs with a single start node, applications working with BOMs can benefit from a natural graph representation and fast graph processing.

3 Architecture Overview

The SAP HANA database [SFL+12] is a memory-centric database. It leverages the capabilities of modern hardware, in particular very large amounts of main memory, multi-core CPUs, and SSD storage, to increase the performance of analytical and transactional applications. Multiple database instances may be distributed across multiple servers to achieve good scalability in terms of data volume and number of application requests. The SAP HANA database provides the high-performance data storage and processing engine within the SAP HANA Appliance product.

The Active Information Store (AIS) project aims at providing a platform for efficiently managing, integrating, and analyzing structured, semi-structured, and unstructured information. It was originally started as an extension to SAP's new in-memory database technology [BKL+12] and has now evolved into a part of it. By tightly integrating the graph processing capabilities into the SAP HANA database rather than providing a separate system layer on top of it, we can directly leverage the fast infrastructure and efficiently combine data from the relational engine and the text engine with graph data in one database query. We tried to build on the existing database engine infrastructure for the new graph capabilities by re-using or extending existing physical data structures and query execution capabilities as much as possible. This helped to keep complexity manageable, both in terms of the number of new system components and in terms of new concepts introduced.

Figure 2 shows the integration of the different AIS components in the architecture of the SAP HANA database. WIPE is the declarative query and manipulation language of the AIS and uses a property graph model extended with semantic information. Database clients can pass in WIPE statements via ODBC or JDBC. Both the language and the AIS data model are described in more detail in the following section. For the execution of WIPE relational operations are re-used where applicable. The basic graph abstractions, operations, and the library of built-in graph processing functionality used to realize the non-relational aspects of WIPE are presented in Section 5. Complex processing tasks are encapsulated as operators, which are implemented on top of the in-memory column store primitives with very little overhead and therefore profit from the efficient information representation and processing of compression and hardware-optimized instructions, respectively.

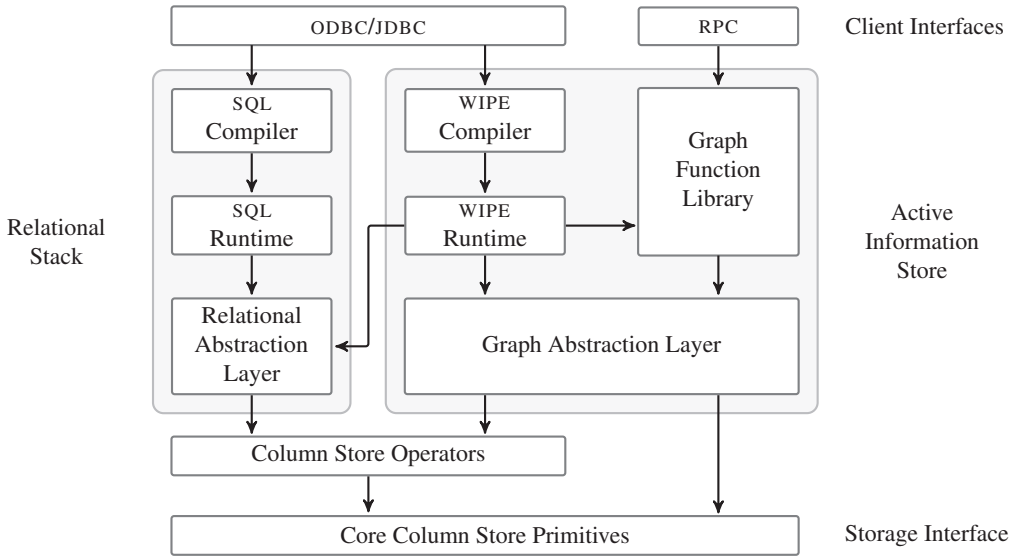


Figure 2: Integration of the Active Information Store in the SAP HANA database

4 The Active Information Store Runtime and WIPE

The AIS data model has been designed such that it permits the uniform handling and combination of structured, irregularly structured, and unstructured data. It extends the property graph model [RN10] by adding concepts and mechanisms to represent and manage semantic types (called *Terms*), which are part of the graph and can form hierarchies. Terms are used for nominal typing: they do not enforce structural constraints, such as the properties a vertex (called *Info Items*) must expose. Info Items that have assigned the same semantic type may, and generally do, have different sets of properties, except for a unique identifier that each Info Item must have. Info Items and Terms are organized in *workspaces*, which establish a scope for visibility and access control. Data querying and manipulation are always performed within a single workspace and user privileges are managed on a per-workspace basis. Finally, Terms can be grouped in domain-specific *taxonomies*.

A pair of Info Items can be connected by directed associations, which are labeled with a Term indicating their semantic type and can also carry attributes. As for Info Items, the number and type of these attributes is not determined by the semantic type of the association. The same pair of Info Items can be related via multiple associations of different types. Figure 3 visualizes the relationships between these concepts as a UML diagram.

WIPE is the data manipulation and query language built on top of the graph functionality in the SAP HANA database. "WIPE" stands for "Weakly-structured Information Processing and Exploration". It combines support for graph traversal and manipulation with BI-like data aggregation. The language allows the declaration of multiple insert, update, delete, and query operations in one complex statement. In particular, in a single WIPE statement

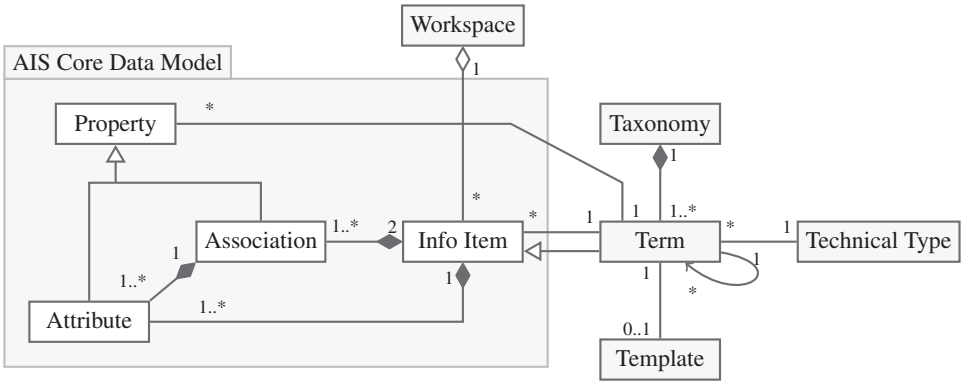


Figure 3: UML [Obj11] class diagram of the Active Information Store data model

multiple named query result sets can be declared and are computed as one logical unit of work in a single request-response roundtrip.

The WIPE language has been designed with several goals in mind [BKL+12]. One is the ability to deal with flexible data schemas and with data coming from different sources. Not maintaining metadata separately from the actual data, the AIS permits introspecting and changing type information in an intuitive way. WIPE offers mass data operations for adding, modifying, and removing attributes and associations, thereby enabling a stepwise integration and combination of heterogeneous data. While navigational queries can be used for data exploration, WIPE also supports information extraction with the help of grouping and aggregation functionality. A rich set of numerical and string manipulation functions helps in implementing analytical tasks.

Like the other domain-specific languages provided by the SAP HANA database, WIPE is embedded in a transaction context. Therefore, the system supports the concurrent execution of multiple WIPE statements guaranteeing atomicity, consistency, durability, and the required isolation.

Listing 1 shows an example WIPE query on the data set presented in Figure 1 returning all books that have received the highest rating at least once. In the first step the graph to operate on is chosen. Thereafter, the set containing the single Info Item representing the “Books” category is assigned to a local name for later use. The third line computes the transitive closure over the “partOf” associations starting from the set specified in the previous step and thereby matches all subcategories of the “Books” category. From there, all Info Items connected via “in” associations are selected and assigned to another local name. Finally, a result is declared that consists of the Info Items matched by the existence quantification, which accepts all Info Items having a “rated” association with a “rating” attribute of value 5.

Listing 1: Example WIPE statement

```
//Tell WIPE which graph data to consult
USE WORKSPACE uri:AIS;

//Save a reference to the "Books" category in a local variable
$booksCategory = { uri:books };

//Traverse to all products in the "Books" category
//The transitive closure (1, *) reaches all arbitrarily nested categories
$allBooks = $booksCategory<-uri:partOf(1, *)<-uri:in;

//Return the books with at least one highest rating using a quantification
RESULT uri:bestBooks FROM $b : $allBooks WITH ANY $b<-uri:rated@uri:rating = 5;
```

5 The Graph Abstraction Layer and Function Library

Modern business applications demand support for easy-to-use interfaces to store, modify and query data graphs inside the database management system. The graph abstraction layer in the SAP HANA database provides an imperative approach to interact with graph data stored in the database by exposing graph concepts, such as vertices and edges, directly to the application developer. Its programming interface, called Graph API, can be used by the application layer via remote procedure calls.

The graph abstraction layer is implemented on top of the low-level execution engine of the column store in the SAP HANA database. It abstracts from the actual implementation of the storage of the graph, which sits on top of the column store and provides efficient access to the vertices and edges of the graph. The programming interface has been designed in such a way, that it seamlessly integrates with popular programming paradigms and frameworks, in particular the Standard Template Library (STL, [Jos99]).

Figure 4 shows the basic concepts of the Graph API and their relationships as a simplified UML class diagram. Method and template parameters as well as namespaces have been omitted for the sake of legibility.

Beside basic retrieval and manipulation functions, the SAP HANA database provides a set of built-in graph operators for application-critical operations. All graph operators interact directly with the column store engine to execute very efficiently and in a highly optimized manner. Well-known and often used graph operators, such as breadth-first and depth-first traversal algorithms, are implemented and can be configured and used via the Graph API. Beside the imperative interface, all graph operators can also be used in a relational execution plan as custom operators.

In the following, we summarize the key functions and methods that are being exposed to the application developer.

- **Creation and deletion of graphs.** The graph abstraction layer allows to create a new graph by specifying minimal database schema information, such as an edge store name, a vertex store name, and a vertex identifier description. This information is

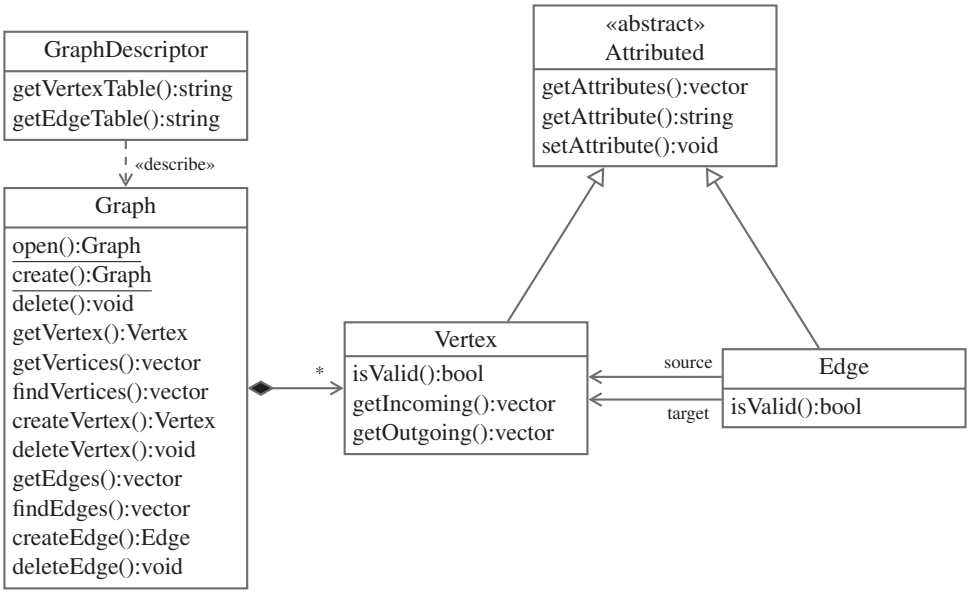


Figure 4: UML [Obj11] class diagram of the Graph API

encapsulated in a graph descriptor object. The creation of a graph is atomic, i.e., if an error occurs during the creation of the graph store object, an exception is thrown and the creation of the graph is aborted.

A graph can be deleted by specifying the corresponding graph descriptor. The deletion process removes the vertex and the edge store from the database system and invalidates the corresponding graph object in the graph abstraction layer.

- **Access to existing graphs.** An existing graph can be opened by specifying the edge store and the vertex store of the graph. All missing information, such as the edge description, are automatically collected from the store metadata. If the graph does not exist in the database management system, an exception is thrown.

- **Addition, deletion, and modification of vertices and edges.** Vertices and edges are represented by light-weight objects that act as an abstract representative of the object stored in the database. The objects in the graph abstraction layer only point to the actual data of the object and hold the internal state during processing. If the graph abstraction layer executes a function call that requests data from the objects, it gets loaded on demand.

- **Retrieval of sets of vertices based on a set of vertex attributes.** Vertices can have assigned multiple properties. These properties can be used to filter vertices, for example, in graph traversal operations.

- **Retrieval of sets of edges based on a set of edges attributes.** Similarly, properties on edges can be leveraged to select possible paths to follow in a graph traversal.

Listing 2: Example pseudo code showing how old ratings can be purged from the data set

```
//Open an existing graph specified with a description object
GraphDescriptor descriptor("VERTEX_TABLE", "EDGE_TABLE");
Graph graph = Graph::open(descriptor);

//Find a specific vertex assuming that the product title is the unique identifier
Vertex vertex = graph.getVertex("Shakespeare in Love");

//Iterate over all incoming edges (those from ratings)
for (Edge incoming : vertex.getIncoming()) {
    Vertex source = incoming.getSource();

    //Find old ratings and delete them
    if (source.getAttribute("created") < threshold) {
        //All incoming and outgoing edges will be removed as well
        graph.deleteVertex(source);
    }
}
```

- **Configurable and extensible graph traversals.** Efficient support for configurable and extensible graph traversals on large graphs is a core asset for business applications to be able to implement customized graph algorithms on top of the Graph API. The SAP HANA database provides native and extensive support for traversals on large graphs on the basis of a graph traversal operator implemented directly in the database kernel.

  The operator traverses the graph in a breadth-first manner and can be extended by a custom visitor object with user-defined actions that are triggered during defined execution points. At any execution point, the user can operate on the working set of vertices that have been discovered during the last iteration. Currently, only non-modifying operations on the working set of vertices are allowed to not change the structure of the graph during the traversal.

Listing 2 illustrates the use of these functions in a C++-like pseudo code. Header file includes, qualified identifiers, exception handling, and STL iterators have been deliberately omitted from the example for the sake of simplicity. In the first line a graph descriptor object is created; it consists of the names of the vertex and edge tables to work with. This is passed to the static `open` method to obtain a handle to the graph in the next line. Thereafter, a handle to the vertex with the identifier "Shakespeare in Love" is retrieved. The for-loop then iterates over all incoming edges of that vertex and for each edge obtains a handle to the source vertex. The value of the "created" attribute of that vertex is compared to some threshold and if it is less, the vertex is removed. All edges connecting the vertex are removed automatically as well.

The graph abstraction layer has to be used from within a transaction context. All modifying and non-modifying operations on the graph data are then guaranteed to be compliant to the ACID properties offered by the SAP HANA database. To achieve this goal, multi version concurrency control (MVCC) is used internally.

Many applications using the graph abstraction layer are built around well-known graph algorithms, which are often only slightly adapted to suit their application-specific needs. If each application bundles its own version of the algorithms it uses, a lot of code will be duplicated. Furthermore, not every application always provides an implementation that is optimal with regards to the data structures the graph abstraction layer offers.

To avoid these problems, the SAP HANA database also contains a graph function library built on top of the core graph operations, which offers parameterizable implementations of often-used graph algorithms specifically optimized for the graph abstraction layer. Applications can reuse these algorithms, which are well-tested, to improve their stability and thereby reduce their development costs.

For example, the graph function library currently contains implementations of algorithms for finding shortest paths, vertex covers, and (strongly) connected components, amongst others. As new applications are built in the future, more algorithms will be supported.

6 Evaluation

In this section we present the first experimental analysis of the integration of the AIS and its query and manipulation language WIPE into the SAP HANA database. In our experiments we show that the AIS is an advantageous approach for supporting graph processing and handling of large data graphs directly within the SAP HANA database. Beside the comparison of WIPE against a pure relational solution for graph traversals using SQL, we also show the scalability of the AIS engine to handle very large graphs efficiently.

6.1 Setup and Methodology

All experiments are conducted on a single server machine running SUSE Linux Enterprise Server 11 (64-bit) with Intel Xeon X5650, 6 cores, 12 hardware threads running at 2.67 GHz, 32 KB L1 data cache, 32 KB L1 instruction cache, 256 KB L2 cache and 12 MB L3 cache shared and 24 GB RAM.

We generated five graph data sets that represent multi-relational, directed property graphs using the R-MAT graph generator [CZF04]. Since the R-MAT generator does not support multi-relational graphs, we enhanced the graph data generation process and labeled edges according to collected edge type distribution statistics from a set of examined real-world batch traceability data sets. We distributed the edge labels randomly across all available edges whereby we labeled edges with types a, b, and c. The selectivities for the edge types are 60 % for type a, 25 % for type b, and 15 % for type c, respectively. Table 1 lists all generated data sets as well as graph statistics that characterize the graph topology. For the remainder of the experimental analysis we will refer to the data sets by their *Data Set ID*.

Further, we use a real-world product co-purchasing graph data set that has been prepared and analyzed by Leskovec et al. [LAH07]. Figure 1 shows an example derived from this data

Table 1: Statistical information for generated graph data sets $G1$–$G5$.

Data Set ID	*# Vertices*	*# Edges*	*Avg. Vertex Out-Degree*	*Max. Vertex Out-Degree*
G1	524 306	4 989 244	22.4	453
G2	1 048 581	9 985 178	25.3	5 621
G3	2 097 122	19 979 348	27.2	9 865
G4	4 192 893	29 983 311	28.1	14 867
G5	15 814 630	39 996 191	28.3	23 546

set. The co-purchasing data set models products, users, and product categories as vertices. Ratings, relationships between product categories, and product category memberships are modeled as edges. Table 2 depicts the most important graph statistics that can be used to describe the graph topology of the data set. Since the co-purchasing graph is a multi-relational graph with highly varying subgraph topologies, we gathered the statistical information for each subgraph separately. The three subgraphs are described by the three edge type labels that exist in the data set. The subgraph *User-Product* contains all users and products as vertices and shows the relationship between these two vertex types via ratings. The subgraph *Product-Category* describes the membership of certain products to product categories. The third subgraph describes the category hierarchy of the co-purchasing graph.

Please note that we do not show the relationships between products, which are also known as co-purchasing characteristics here for the sake of simplicity. Additionally, it is worth to mention that vertices in the data sets contribute to multiple subgraphs. Because of this, the summation of number of vertices from all subgraphs is larger than the actual number of vertices in the complete graph.

We loaded the data sets into two tables, one for storing the vertices and one for storing the edges of the graph. Thereby, each vertex is represented as a record in the vertex table `VERTICES` and each edge is represented as a record in the edge table `EDGES`. Each edge record comprises a tuple of vertex identifiers specifying source and target vertex as well as an edge type label and a set of additional application-specific attributes.

Listings 3 and 4 depict a qualitative comparison between a WIPE query performing a graph traversal and the equivalent SQL query that heavily relies on chained self-joins. Please note that both queries are based on the same physical data layout (vertex and edge table). While the SQL query addresses the storage of vertices and edges explicitly via table name and schema name, a WIPE query only needs to specify a workspace identifier. The table names for vertices and edges are preconfigured in the database configuration and do not need to be specified during query execution.
Both queries perform a breadth-first traversal starting from vertex *A*, following edges with

Table 2: Statistical information for co-purchasing graph data set $A1$.

Subgraph	*# Vertices*	*# Edges*	*Avg. Vertex Out-Degree*	*Max. Vertex Out-Degree*	*Avg. Vertex In-Degree*	*Max. Vertex In-Degree*
User-Product	832 574	7 781 990	5.4	124	6.3	237
Product-Category	588 354	2 509 422	3.1	13.3	53.4	23 121
Category-Category	23 647	7 263	2.1	78	2.1	78

Listing 3: SQL statement

```
SELECT DISTINCT V.id
  FROM AIS.EDGES AS A, AIS.EDGES AS B,
       AIS.EDGES AS C, AIS.EDGES AS D,
       AIS.VERTICES AS V
 WHERE A.source = "A"
   AND D.target = V.id
   AND A.type = "a"
   AND A.target = B.source
   AND B.target = C.source
   AND C.target = D.source
```

Listing 4: WIPE statement

```
USE WORKSPACE uri:AIS;
$root = { uri:A };
$t = $root->uri:a(4,4);
RESULT uri:res FROM $t;
```

type label *a*, and finally return all vertices with a distance of exactly 4 from the start vertex. The SQL query executes an initial filter expression on the edge type label *a* to filter out all non-matching edges. Next, the start vertex is selected and the corresponding neighbor vertices are used to start the chained self-joins. For $n = 4$ traversal steps, $n - 1$ self-joins have to be performed.

In contrast, the WIPE query selects the start vertex *A* and binds the result to a temporary variable *$root*. Next, a traversal expression evaluates a breadth-first traversal from the start vertex over edges with edge type label *a* and performs 4 traversal steps. Finally, the output of the traversal expression is returned to the user. For more examples of WIPE queries, we refer to [BKL+12].

Both queries are functionally equivalent and return the same set of vertices. However, a WIPE query provides a much more intuitive and compact interface to directly interact with a graph stored in the database.

6.2 Experiments

The first experiment shows a comparison of the scalability of SQL queries performing graph traversals versus their corresponding WIPE query counterpart and is depicted in Figures 5. For the experiment in Figure 5, we varied the number of path steps between 1 and 10 and randomly chose a start vertex for each graph traversal run. We ran each query 10 times and averaged the execution time after removing the best and the worst performing query. We restricted the number of path steps to 10 since all graphs exhibit the small-world property that can be found in many real-world data graphs [Mil67]. In this experiment, we showcase how the execution time evolves when more path steps are to be performed during the graph traversal. To illustrate the behavior, we use the graph traversal over one path step as baseline and relate subsequent queries traversing over multiple path steps to this one-step graph traversal. Consequently, the execution time of queries over multiple path steps is always a multiple of the execution time of a one-step traversal.

Figure 5 shows the gained relative execution factor when comparing a one-step traversal as baseline against a multi-step traversal in subsequent queries. The relative execution factor is

plotted with logarithmic scale to the base 2. The green-colored plot with the circle markers shows the relative execution factor for the SQL query traversal with respect to the one-step traversal. The plot reflects the poor behavior for multi-step traversals of SQL queries with a number of path steps larger than 2. The blue-colored plot with square markers shows the relative execution factor of the WIPE query with respect to the one-step traversal baseline. The relative execution factor of WIPE grows much slower than the relative execution factor of the equivalent SQL statement. Thereby, a linear and slow rise is better in terms of scalability to the number of path steps to perform. The WIPE implementation shows for data set $G4$ a maximum relative execution factor of 3 and the SQL implementation shows a maximum relative execution factor of 91.

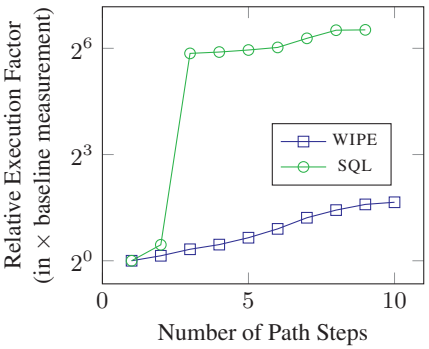


Figure 5: Scalability of SQL and WIPE for $G4$

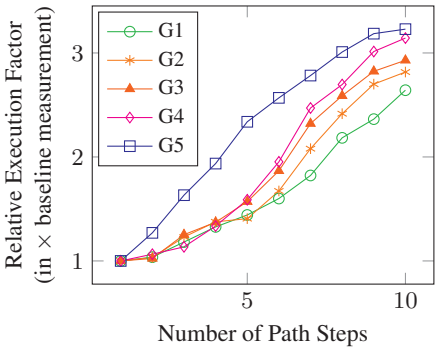


Figure 6: Scalability of WIPE

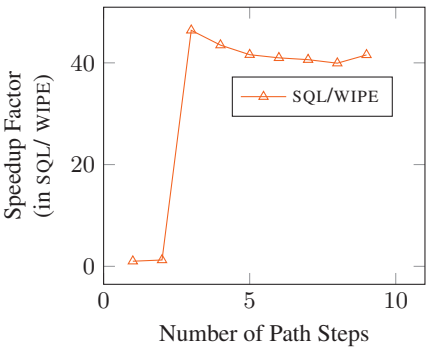


Figure 7: SQL and WIPE compared for $G4$

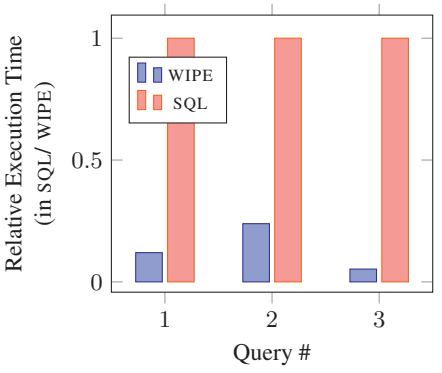


Figure 8: SQL and WIPE compared for $A1$

The next experiment illustrates the scalability of WIPE and the AIS engine with respect to growing graph data sets. Figure 6 depicts the results for given data sets $G1$–$G5$. As for the first experiment, we varied the number of path steps between 1 and 10 and chose a start vertex randomly for each graph traversal run. We used the one-step traversal as baseline and related all subsequent multi-step traversal queries to it.

Table 3: Real-world queries from the co-purchasing domain.

Query ID	*Description*
1	Select all super categories from product category “Movies."
2	Find all users which rated “Hamlet" and “Romeo and Juliet" with 5 stars.
3	Find all books that are in category “Fiction" or sub categories and have been released to DVD.

The results indicate a good scalability of WIPE and the AIS engine with respect to increasing data set sizes and varying number of path steps. For all examined data sets, we found maximum relative execution factors between 2.6 and 3.2. Thereby, we saw only marginal variations between the different data sets which are mainly caused by differences in the underlying graph topologies. Independent from the graph topology present in the examined data set, the traversal operator scales very well with respect to large graph data sets as well as increasing number of path steps to perform.

Figures 7 and 8 depict the speedup in execution time of WIPE queries compared to an equivalent SQL query. Here, we relate the execution time of the SQL query to the execution time of the equivalent WIPE query.

Figure 7 illustrates the speedup factor between SQL and WIPE for data set $G4$. We obtain a maximum speedup factor of 46 when comparing and relating the execution times of SQL queries to their equivalent WIPE query performing the same number of path steps.

For graph traversals with a path length of one or two, the gained speedup for WIPE was between 1.03 and 1.24. For one-step traversals, the SQL query can directly return the neighbors of the given start vertex without the need to perform expensive self-joins. For two-step traversals, a self-join has to be performed to retrieve vertices with path distance 2 from the root vertex. In general, for graph traversals with a larger number of path steps, a built-in operator clearly outperforms a functionally equivalent SQL query on average by a factor of 30 in our experiments.

For the SQL variant of the graph traversal, the execution time is highly dependent on the number of vertices that are being discovered at each level of the graph traversal algorithm. When a large fraction of all reachable vertices has been discovered, the speedup between SQL and WIPE will decrease slightly as can be seen for graph traversals with three or more path steps. However, WIPE still performs about 30 times as fast as the SQL implementation of the graph traversal algorithm.

In Figure 8, we compare the relative execution time for three real-world queries from the co-purchasing domain. The queries are described in Table 3. The figure shows a relative execution factor of WIPE against the equivalent SQL implementation between a factor of about 4 and 20. The most beneficial query is query 3 since it involves the most complex traversal execution and cannot be handled efficiently by the relational engine. If there are only a limited number of path steps to perform (as for query 2), the results are similar to those obtained in the other experiments. Since the execution time of the SQL query is dependent on the number of self-joins to perform (the number of path steps to traverse), the relative execution factor is lower for simple graph traversals.

7 Summary

A variety of business applications work with inherently graph-structured data and therefore profit from a data management layer providing an optimized representation of graph data structures and operators optimized for this representation. Within this paper, we outlined the SAP HANA Active Information Store project with its query and manipulation language WIPE and a property graph model as the underlying data representation. The general goal of the project on the one hand is to provide built-in graph-processing support leveraging the performance and scalability of the SAP HANA main-memory database engine. On the other hand, the project aims at providing a powerful and intuitive foundation for the development of modern business applications.

Therefore, we first motivated the project by presenting different scenarios ranging from classical graph-processing in social network analytics to the domain of supply chain management and product batch traceability. We then briefly touched the overall architecture of SAP HANA with respect to the functionality related to graph processing. The original design of SAP HANA allows to operate on multiple language stacks in parallel with local compilers exploiting a common set of low-level column-store primitives running in a scalable distributed data-flow processing environment.

In the second part of the paper, we described the graph abstraction layer using an practical example and presented results of extensive experiments. The experiments were conducted on several synthetic graph data sets to show the effects of different topologies with respect to certain query scenarios. Overall, the optimized graph-processing functionality performs significantly better than the comparable SQL representation using only relational operators. The specific support for graph-structured data sets and the matured distributed query processing engine of SAP HANA provides a superb solution for complex graph query expressions and very large graph data sets.

Acknowledgement

We would like to express our gratitude to Hannes Voigt for many inspiring discussions and for his feedback on earlier versions of this paper. We also thank our fellow Ph.D. students in Walldorf for their encouragement and support.

References

[BKL+12] Christof Bornhövd, Robert Kubis, Wolfgang Lehner, Hannes Voigt, and Horst Werner. Flexible Information Management, Exploration, and Analysis in SAP HANA. In Markus Helfert, Chiara Francalanci, and Joaquim Filipe, editors, *Proceedings of the International Conference on Data Technologies and Applications*, pages 15–28. SciTePress, 2012.

[CK04] Jeremy J. Carroll and Graham Klyne. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C recommendation, W3C, February 2004.

[CZF04] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A Recursive Model for Graph Mining. In *SIAM International Conference on Data Mining*, SDM '04, pages 442–446. Society for Industrial and Applied Mathematics, 2004.

[FCP+12] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. SAP HANA Database: Data Management for Modern Business Applications. *SIGMOD Rec.*, 40(4):45–51, January 2012.

[ISO12] ISO. *Technical product documentation – Vocabulary – Terms relating to technical drawings, product definition and related documentation (ISO 10209:2012)*. International Organization for Standardization, Geneva, Switzerland, 2012.

[Jos99] Nicolai M. Josuttis. *The C++ Standard Library: A Tutorial and Reference*. Addison-Wesley Professional, 1999.

[LAH07] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. The Dynamics of Viral Marketing. *ACM Trans. Web*, 1(1), May 2007.

[Mil67] Stanley Milgram. The Small World Problem. *Psychology Today*, 61(1):60–67, 1967.

[Obj11] Object Management Group. OMG Unified Modeling Languange (OMG UML), Infrastructure, 2011. Version 2.4.1.

[RN10] Marko A. Rodriguez and Peter Neubauer. Constructions from Dots and Lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, 2010.

[SFL+12] Vishal Sikka, Franz Färber, Wolfgang Lehner, Sang Kyun Cha, Thomas Peh, and Christof Bornhövd. Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 731–742, New York, NY, USA, 2012. ACM.

Rethinking Energy Data Management: Trends and Challenges in Today's Transforming Markets

Robert Ulbricht[2], Ulrike Fischer[1], Wolfgang Lehner[1], Hilko Donker[2]

[1]Dresden University of Technology, Database Technology Group, Germany
[2]Robotron Datenbank-Software GmbH, Dresden, Germany
first name.name@tu-dresden.de
first name.name@robotron.de

Abstract: The energy market domain is subject to a continuous transformation process, mostly driven by governmental regulations. To efficiently handle the large amounts of data and the communication processes between market participants, specialized database applications have been developed. In this paper, we present the energy data management system (EDMS) as a standard software solution, describing its core components and typical system integration aspects. However, current market topics like smart metering, energy saving, forecasting for renewable energy sources, mobile consumption and smart grids lead to new database challenges. We provide an overview of these trends and discuss their impact on existing information systems, focusing on the technical challenges of data integration, data storage, data analytics and scalability. As energy data management has to match those new requirements, promising research opportunities are offered to the database community.

1 Introduction

In the past years we could observe numerous changes on the European energy markets. Starting with the market liberalization in the late 1990s, a wide range of regulations forced energy companies to audit and completely redesign their organizational structures, business processes, and technologies. The unbundling of the former vertically integrated and not uncommonly public utility companies subdivided them into local or regional distribution system operators (DSOs), transmission system operators (TSOs) and energy producers or -suppliers. The political objective was the establishment of a competitive market environment, enforcing higher service quality, more innovative technological concepts and simultaneous cost reductions. As the new market roles' activities had to be restricted to certain operations [JP05], participants were forced to interact intensively. This also created new requirements on existing system landscapes. To efficiently handle the massive amount of data created by the new communication streams, the first generation of a specialized information system sub-type was introduced: the *Energy Data Management System* (EDMS). The EDMS' main tasks are (1) to store and process almost all kinds of master- and transaction data related to energy logistics, and (2) to manage all automated data exchange processes between the different market partners in given time frames ac-

cording to particular market rules. For example, the DSO has to send daily load curves retrieved from customers equipped with remote load profile meters to the corresponding energy supplier, the supplier is responsible for purchasing the distributed energy from a trader or broker, and finally the trader has to contract the capacities from energy producers to close his open positions. The TSO provides the infrastructure and necessary balancing energy for the trans-regional high-voltage transmission network. His associated balancing coordinator is responsible for monitoring the matching between demand and supply for all energy traders operating within the transmission grid (see Figure 1).

Figure 1: Market roles and relations after complete liberalization

Once established as costly individual implementations, pioneers and early movers soon initialized ambitious standardization efforts turning their solutions into market standard software. Because of the obtained synergies, the EDMS soon became an interesting and affordable option even for smaller, non-unbundled utility companies or big consumers like energy-intensive industries.

With market rules frequently being changed driven by diverse governmental regulation efforts (e.g. climate saving propositions), and new technologies becoming available, the EDMS is subject to a continuous and evolutionary adaptation process. So we notice the strong influence of constantly increasing capacities of renewable energy sources due to excessive funding policies (e.g., [Eur11a], [Ren12]) and industrial promotion, making conventional power plants becoming less attractive to private investors. Renewable energy is characterized by a decentralized allocation and fluctuating output, thus making it difficult to maintain stability in power networks where energy supply and demand must be balanced carefully. Amongst others, this is one security-related aspect of the energy supply system transformation (or transition) studied in [NPS+12]. To prevent collapsing grids in the near future, and to battle the negative influences of the growing economies' ever increasing energy consumption on greenhouse gas emissions, new technical concepts are introduced. This includes e.g. intelligent and networked measurement devices, improved forecasting methods, efficient storage systems or the integration of mobile consumption points.

In this paper, we will address the technical challenges for energy data management systems against the background of contemporary market policies. In detail, we make the following contributions:

- First, we will show a brief market review of existing standard products and some ongoing related research projects (Section 2).

- Second, we will give a general description of commercial EDMS, their core components, the tasks involving them within a standard architecture and the position of the EDMS in a typical system landscape (Section 3).
- Further we will provide an overview of current topics related to energy market transition and outline their impact on requirements for information system architectures (Section 4).

Finally, we conclude our observations and finish by giving some additional directions for future developments (Section 5).

2 Market Relevance

Energy data management systems are already available as standardized commercial products provided by software developing companies. Leading systems with numerous installations in different countries are *Oracle Lodestar* or the *SAP IS-U-EDM*, but due to many country-specific requirements on local markets, there are no all-dominating multinational players. Most of the market participants are still acting locally or regionally, and therefore often prefer to choose national service providers instead of adapting foreign system philosophies. Take a look at Germany for example, Europe's biggest energy market, where well-known and widely spread solutions especially among large utility companies are the *robotron*ecount* product family and *Soptim's* energy logistics solution, or the *BelVis* EDMS primarily focusing at customers of small and very small size. There are a couple of other very active but smaller players, appearing on the complete listing recently published in [Sto12]. Less frequently and shrinking in importance, EDMSs can also still be found as proprietary solutions implemented by the utility companies.

Beside commercial implementation efforts, some interesting local and supranational projects and research initiatives shall be mentioned. Each of them addresses the field of modern energy data management or related topics. In Europe currently very popular are all type of smart grid projects, almost 200 are listed on [GGFS11]. Amongst them, the most notable examples are: the *MIRABEL* project [BDD+12], trying to balance energy supply and demand using micro-requests and flex offers; the *EU DEEP*[1] project, dealing with the integration of distributed energy sources in today's electricity systems, and the *ADDRESS* [PBB+09] project led by Italy, focusing on the active participation of small and commercial consumers in a large-scale demand respond concept implementation. The *e-Highway2050*[2] is a recently approved research project on the pan-European transmission network in order to facilitate cross-border energy exchange. The Danish *EDISON*[3] project focuses on intelligent system integration to manage charges of distributed electric vehicles plugged into a grid. Also in the United States, smart grid and energy storage projects receive massive governmental support [US 12], and even Asia's emerging countries like

[1] http://www.eu-deep.com
[2] http://www.entsoe.eu/system-development/2050-electricity-highways/
[3] http://www.edison-net.dk

China and India are dealing with the implementation of smart grid technology in order to compensate their relatively week network infrastructures [HHM11].

As shown in this section, there are plenty of commercial and scientific activities involving energy data management topics. Next up we will present the EDMS as part of the underlying information technology in more detail.

3 EDMS architecture

Although the standard functionality of today's available EDMS strongly varies depending on national market rules and specific business requirements, some similar design concepts can be observed. In this section we introduce the EDMS giving a brief description of its architecture (compare Section 3.1), some supported standard processes (3.2) and common interfaces (3.3).

3.1 Core Components

In the system architecture of EDMS, we can identify seven generic core components shown in Figure 2: First, a database is used to store *Time Series* and includes an application programming interface (API), another database contains the *Master Data* objects. Then, realized either as functional modules within the database or as external applications, there are a *Task Scheduler*, a *Communication Gateway*, a *Calculation Module* and an *Administration Module*. They represent the common base for any additional specialized advancement, for example energy balancing or forecasting, modeled as *Business Logic Modules*. Therefore, they are considered as the obligatory part of an EDMS and referred as the energy data application framework. Below, we describe this framework looking closer at its elements, focusing on embedded functionality and relations within the system architecture.

Figure 2: EDMS core components

Time Series Management

Doubtless the most important requirement of a modern EDMS is the ability to handle time series data, which can be defined as sequences of numerical values in successive order collected over a period of time. Time series are common data structures for energy logistics, used to handle production or consumption data, temperatures, prices or any other related numerical information. They occur either with equidistant (=equally spaced) or non-equidistant intervals and can have additional attributes like quality status information.

Because of the large amount of time series exchanged on the market every day, all values and attributes have to be stored in a space saving and therefore compressed form but must be easy to extract, thus guaranteeing user access at any given time. This means that compression and decompression has to be done directly within the database. In Figure 3 we demonstrate an exemplary way to reduce numerical raw data combining it with separately stored context information (extracted from master data database): Every equidistant time series is stored as one single row in a table. Now time stamps are redundant, as any value can be addressed looping through the string record having the time series starting point and its period. Second, energy consumption values can carry additional status information characters extracted from the meter (e.g. T = true value, S = substitution value, E = error). Run-length encoding is another good strategy to compress them, as this type of time series is supposed to contain many true values mixed with very few disturbed ones. Like anywhere else, choosing the best compression method is a cost-optimization problem, balancing reduced hard disk space against additional CPU capacity.

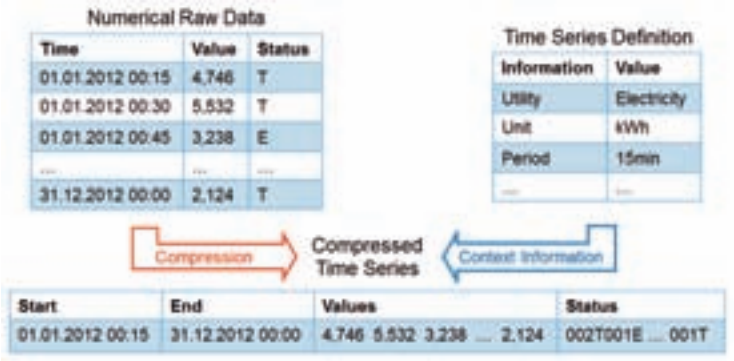


Figure 3: Structural transformation of equidistant time series

To avoid integrity problems within compressed structures, direct data manipulation cannot be allowed. Any reading or writing operation must strictly be done via an adequate application programming interface, called the time series API (Figure 2). Time zones and daylight saving time shifts have to be taken into consideration too and must be included. Further, due to the financial aspect of billing-relevant time series, all changes made on original raw data must be logged and archived. Transaction logging enables the system to automatically recover a historical situation whenever it should be requested by a market partner.

Master Data Storage

Any pure numerical data has to be related to master data objects in order to access its context information and for further processing. Master data can be extensive and modeled in many different ways. However, we observe an often repeated fundamental design concept: market partner and meter point are represented as independent entities, being temporally connected by the contract entity as shown in Figure 4. For instance, the relationship between an energy supplier and a customer (both market partners) is represented as one or more supply contracts between them, having one meter point assigned to each contract. This allows the system to handle all relationship changes (e.g. supplier change, customer moving in or out) modifying only the contract object, leaving everything else untouched. Also, contracts are not stringently required to have a real-world counterpart. Being virtual objects, they can be used to model any connection between market partners in the system.

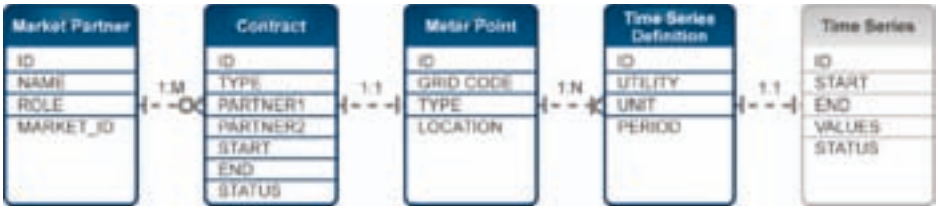


Figure 4: Relational master data objects

Load curves (time series) or numerical consumption values are assigned to the meter point object via the time series definition, which contains the additional context information shown in Figure 3. The meter point is identified by its official and unique market ID, the so-called metering- or grid code. The application must contain an audit trail for the key master data objects. The audit trail contains the prior and the new state of the data, the modifying user and date. Some of them are relevant for triggering events in order to communicate updates on master data to external market partners or neighboring systems.

Calculation Module

The calculation module offers all necessary mathematical and logical functions to manipulate existing time series or to calculate new ones based on input data (e.g. balancing sums). To offer the highest performance possible, it is directly based on the time series API. Physical units and periods of source data are automatically converted to the corresponding targets definition. Calculations can be arranged in a tree-like hierarchy according to existing master data structures. This is a common way to model different levels of aggregation. Once a hierarchical calculation is defined and activated, results are materialized and stored. Optimized maintenance strategies are implemented to force an automatic recalculation of all dependent nodes in the tree with every single update on underlying source data.

Communication Gateway

The communication gateway represents the connection between the EDMS operator and the energy market. The framework element is responsible for sending and receiving data using standardized electronic message formats like Edifact subsets or customized file for-

mats (e.g. CSV or XML structures). Therefore, in opposite to traditional database applications, automatic file format conversion is a major issue. Today's market communication is mostly done by exchanging automated emails between market partners, so protocols like SMTP, IMAP or FTP must be supported. Manual interactions by people via phone or fax are reduced to an absolute minimum, speeding up business processes through the elimination of typical human errors and restrictions. All received messages must be confirmed to the sender, indicating their syntactical and semantical correctness. Validation is done outside the system to keep incorrect files away from database transaction processes. Any transmitted data has to be treated as confidential information. That is why the module's architecture must offer a possibility to include state-of-the-art encryption mechanisms. The communication gateway must be directly connected to the master data and time series databases in order to import or export data.

Task Scheduler

As most of daily work consists of handling mass data, an EDMS is designed to perform all tasks at a highly automatized level. This is usually done by using time or event based task scheduling. Realizing time based scheduling using database core functionality is a simple and standard automatization approach. All tasks are programmed as scheduler jobs and can either be executed immediately or stored in a queue and processed step by step by assigning pre-defined priority levels. If results are not needed immediately, large report queries or complex hierarchical calculations can be scheduled in low-workload time frames, thus avoiding interferences with daily work. Job results are protocoled and result quality can be classified. Failed jobs generate new tasks, for example by mailing the error log file to an administrator. Compared to time based scheduling, tracking down possible events in order to start certain scheduler jobs is the more complex way by using common pushing or polling techniques. As for pushing, triggers must be implemented and integrity checks must be used on all relevant tables. However, triggers are hard to debug and can slow down a system's performance notably as they are active on every transaction or can cause other triggers to fire. To avoid all this polling can be used instead, but depending on the data model, the query intervals and the number of simultaneous tasks this can lead to expensive operations in relational databases.

Administration Module

Like any other standard software of comparable complexity, the EDMS needs a vast number of configuration options to offer maximum flexibility. These activities are bundled in the administration module. Parameters can be set at different levels: global, thus affecting operations throughout the whole system, like setting the servers time zone, and client- or user-specific (e.g. language settings, output file format, etc.). Additionally, local settings like user accounts and user groups are administrated here. The application has to implement a robust user security model. We distinguish between role-based restrictions on certain database functions and all or only partial-level data access restrictions with different granularities for reading and writing operations.

Business Logic Modules

Besides basic functionality, the framework includes additional elements offering support for specialized business purposes. This includes pricing, forecasting, a workflow-engine

with pre-defined business process schemes, contract management and balancing-, trading- or procurement modules. There are different points of view on what type of processes should be included in the EDMS or which should rather be handled in an external stand-alone application. These are influenced fore-mostly by three factors: (1) the market role of the system operator, (2) the evolutionary level of development and (3) the software company's product strategy. Some important functions needed by network operators are dispensable for energy retailers and vice versa. State-of-the-art systems tend to offer much more complexity than older products. For our work, we abstain from a further particularization at this point and refer to those elements as business logic modules in general.

3.2 Typical Energy Data Management Processes

Now we demonstrate how the recently introduced energy data application frame-work components work together and can involve additional functionality. Therefore we describe two typical business processes as exemplaric use cases.

Load profile processing

Load profile processing is a fundamental process which has to be handled by every network operator and is always following the same pattern (Figure 5): The DSO receives load curves from an automated meter reading system (AMRS) or from external market partners in daily intervals. The data is directly imported into the EDMS using the communication module (compare Section 3.1) and converted into time series (*raw data import*).

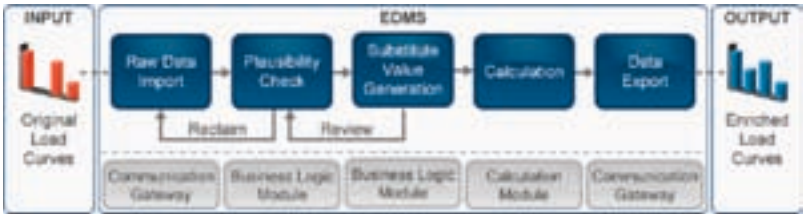


Figure 5: Load profile processing

Second, the DSO runs a *data plausibility* check. Using pre-defined comparison parameters, the quality of received load curves is verified. Any missing, incorrect or mismatching values are identified, marked for further treatment and excluded from processes like calculations or data exports. To correct these problems, *substitute values* are generated using common mechanisms like linear interpolation, single value distribution, copying historic values from comparable days or, if available, from a control measurement. These three process steps are iterative, so if all correction efforts fail the missing data must be claimed again from the sender. Once having a corrected load curve, the DSO can start with the data processing, normally consisting of calculating aggregated load curves for customers with multiple meter points, balancing sums for external suppliers operating within the DSO's network or providing billing-relevant consumption values (*calculation*).

Finally, the data export step extracts and ships the enriched, corrected or completed data using the communication module again. Because of the high amount of raw data and intra-day handling times, the whole process chain is executed almost automatically by the task scheduler, reducing manual interaction to the indispensable cases only.

Energy sales and distribution

Besides the DSO point of view, another good example for the practical use of the EDMS is the main business process conducted by any energy supplying company: Selling and distributing energy to consumers (Figure 6).

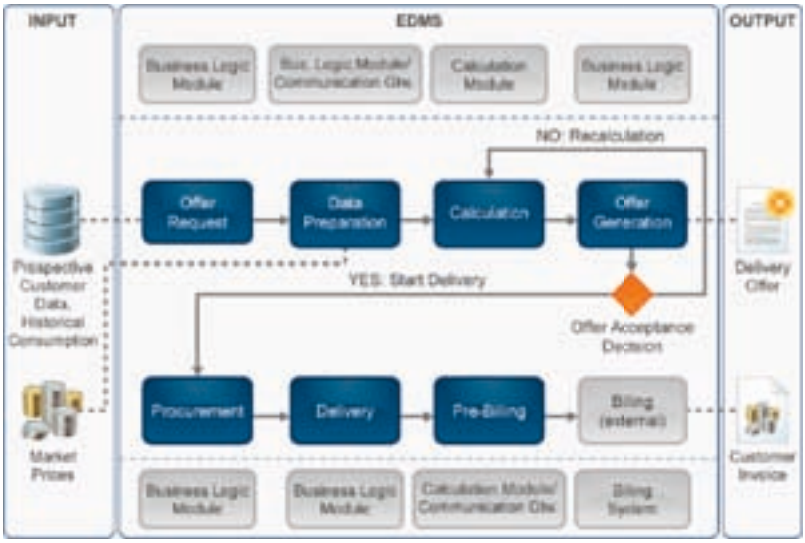


Figure 6: Standard energy sales and distribution process

The process chain starts with a prospective customer contacting the sales department requesting an offer (*offer request*). The next step is the *data preparation*. This means that the master data structures are created, the future energy consumption during the whole requested period is forecasted using historical values (if available) or based upon experience made with similar demand profiles. Furthermore, the market price information, received either from an external information broker (e.g. stock exchange services) or derived from own investigations, must be added. Now *calculation* can start by using the predicted load curve and the corresponding market price curves. Having all source data available, the process moves on to *offer generation*, where the calculated prices are approved by superiors, printed in a standardized form and handed over to the potential customer. At this point the further course depends on customer's decision: A refused offer can be reworked (e.g. including discount) or closed, thus terminating the whole process; an accepted offer initiates the *procurement* phase. A new supply contract is created and added as open position to the supplier's portfolio in order to procure the offered amount of energy from the market. After that, the *delivery* can start. The new contract is communicated to the corresponding network operator via the supplier change process. Only now the customer

starts receiving energy from his/her new supplier. At the end of the billing period (e.g. one month after first delivery), the consumption data has to be collected, pre-processed and handed over which is done in the *pre-billing* phase. Finally, *billing* can start. However, this activity has rather to be realized in a proper billing system.

As shown in this sub-section, numerous modules are involved in different energy data management processes, each module offering dedicated functionality. This underlines that an EDMS is a complex application, which requires a deep level of component integration in order to avoid unsolicited breakpoints in highly automated processes.

3.3 System Integration

In practice, we experience different approaches to integrate the EDMS into a company's system landscape. Generally, we consider the EDMS to be the central data processing unit within a utility company due to the following reasons: (1) energy data management is a centralized task involving various independent corporate units, (2) most of the data needed for daily operations and reporting is available in the system and (3) it's powerful communication module is the primary gateway to external market partners or can be well used as internal interface to neighboring systems within the corporation.

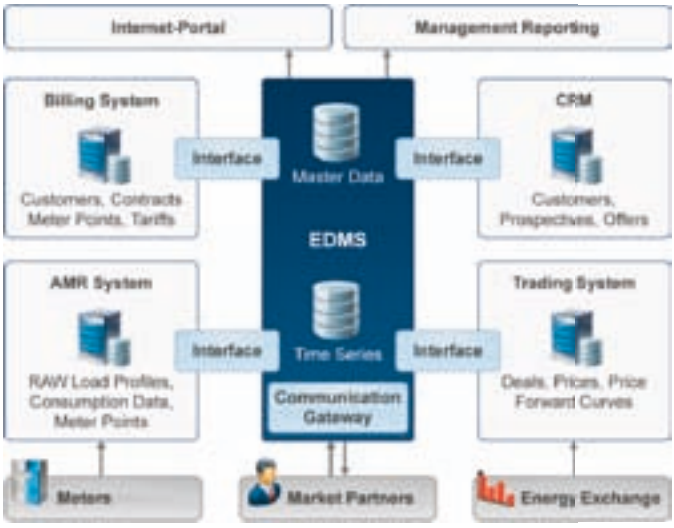


Figure 7: Example for an EDMS integration into a system landscape

In Figure 7 we demonstrate a typical system landscape in an integrated view for both DSO and energy supplier: The central positioned EDMS receives master data either from the billing system or the customer relationship management system (CRM), as they usually possess master data superiority. Meters send their load profiles to an AMRS, where raw data is accumulated, converted into time series and forwarded to the EDMS for fur-

ther processing operations. A trading system provides real-time or forward price curves needed for retail offer generation (see 3.2) - deals and portfolios are managed here too. The EDMS can be integrated into a web portal, allowing customers to access their accounted consumption values online. Furthermore, the database is used to generate all kind of management reports on relevant information.

Usual interface designs include database links, either directly between the relevant systems or using a connecting middleware, web-based services frequently being used in SOA architectures or even simple file transfer based on plain text formats or XML structures. A second way to realize data exchange between dedicated systems is to use the official standard market formats (e.g. Edifact), assuming that applications should be able to handle them accordingly. This makes implementation projects cheaper and faster, but has certain disadvantages: the information transfer is restricted to the format's syntactical capability and with every change on external file formats the interface needs maintenance. Finding the optimal technology basically depends on transfer frequency, data volume and -complexity and reasonable maintenance expenditures.

4 Challenges

New technological trends and political changes constantly create challenges and additional requirements for existing system landscapes. In this section we address some of the market's frequently discussed topics and analyze their possible impact on energy data management. As for the market challenges, smart metering (4.1), energy forecasting (4.2), energy saving (4.3), the integration of mobile consumption devices (4.4) and the smart grid technology (4.5) are the most relevant drivers. They result directly in typical challenges for data management technology and processes, especially as far as data integration, data analytics, data storage and scalability (including flanking aspects like multi-tenancy and parallelized processing) are concerned. The relationship between the database and market challenges is demonstrated in Figure 8 and will be explained more in detail in the following subsections.

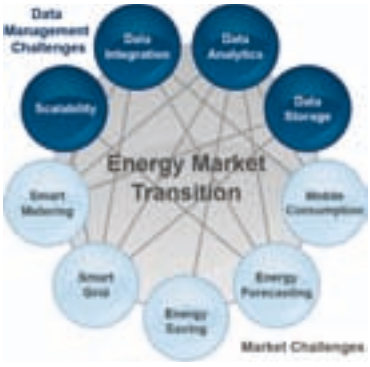


Figure 8: Relations between Market- and Data Management Challenges

4.1 Smart Metering

Many of the meters installed in households and small businesses still do not have a communication module. Load curves are not available and consumption values have to be retrieved manually in several periods, usually once or twice a year. Automated meter reading installations based on one-way communication are able to automatically transmit consumption data to a centralized database system. Making use of two-way data communication protocols enables additional features, like networked meters or remote control applications like meter status verification or on-demand data reading. This is meant by Smart Meter solutions and is widely seen as the technological key element for future smart grids (see Section 4.5).

A topic not limited to liberalized European markets, smart meters are being introduced in many of the world's developed countries. Europe, driven to a large extent by regulations, had an early start in Italy in the 2000s, soon followed by full-scale rollouts in Scandinavia and other countries [Ryb12]. North America has actually the world's highest penetration of automatic meter reading, while the Asia-Pacific region is still situated in an early stage of the adoption process with Japan and South Korea moving ahead.

Data Management Challenges

The major issue with smart metering technology is the handling of massive data amounts flooding the central data management systems on a daily basis. Instead of one single value a year, automated meters protocol load curves with a measurement rate of 5 to 60 minutes (corresponds 8760 to 105K values/year) depending on national regulations. The data is transmitted once a day or even with an intraday frequency, thus creating a challenge even for advanced *data storage* capabilities and compression algorithms.

Figure 9: Data transfer process optimization using integrated MDMS

As one attempt in order to reduce data transaction time and typical *data integration* problems, we observe the consolidation of automated meter reading systems and EDMS to *Meter Data Management Systems* (MDMS), integrating both meter reading and energy data management functionality. The goal is to shorten the transaction process chain introduced as load profile processing in Section 3.2 (compare Figure 5). This is done by eliminating interim steps caused by data transfer between independent systems, thereby at the same time sparing storage space for redundant raw data archiving as demonstrated in Figure 9: Using integrated MDMS, data can be transfered directly to the time series

database. These consolidations have two possible development directions: (1) Adding a meter reading module directly to the EDMS framework or (2) implementing EDMS components in the AMRS, each eliminating the importance of the respective opposing system.

Apparently, smart meters are cost-intensive compared to conventional metering technologies. Functions like automatic firmware updates or remote meter locking, such as a comfortable fraud prevention measures, are useful for the meter operators. However, the use of recorded data is still limited to energy delivery and billing purposes, thus not providing any advantage for private end-supply customers. That is why innovative concepts are required to make advanced meter technologies more attractive. Using advanced *data analytics* methods to support the creation of generic, consumption-specific customer profiles allows the offering of new products, for example flexible tariff contracts based on real-time energy prices. Providing value-adding features like networking smart meters to create completely integrated smart home solutions, as recommended in [GV12] and demonstrated in the form of a prototypal implementation by [JJP+10], can also make a difference.

Another aspect is growing legal discussions concerning data security in a smart meter gateway, where some important rules have to be kept in mind. Consumption data must be protected using secure software architectures and must be processed in anonymized or clustered form to prevent any possible conclusions of a single customer's consumption behavior. Furthermore, data encryption might be applied to protect the confidentiality of the customer requiring *efficient query processing* techniques on such data (e.g., [HILM02]).

4.2 Energy Forecasting

Even the smartest meter technology can only provide a recorded view on past situations, but the key to balance an energy distribution network successfully is to plan actions foresightful by predicting as many of the most influencing (correlated) parameters for operations as possible. For both demand and supply forecasts the same generalized model [Has07] can be applied (Figure 10).

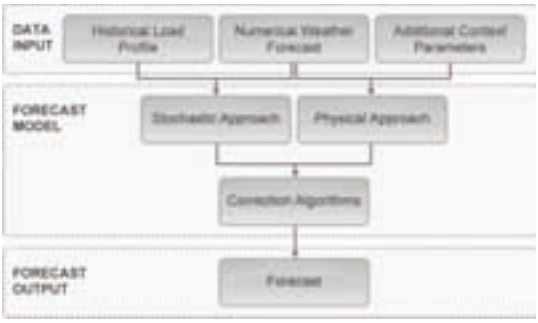


Figure 10: General energy forecast approaches

On the input side, either *historical load profiles* or *context parameters* like a supply installation's total capacity and transformation effectiveness must be provided. In case of weather-aware forecast models, the parameters must be combined with *numerical weather forecasts*, usually obtained from contracted metrological services.

After input data preparation, a forecast model must be selected and applied. *Physical Prediction* models rest upon an installations physical and technical character. Not depending on the availability of measured load curves, these approaches can quickly be implemented, making them indispensable especially for new installations, where no historical data is available. However, their practical usefulness is limited, as DSOs normally do not have access to all required technical details. In contrast, *Stochastic Prediction* models (e.g. regression, exponential smoothing) are based on time series analyses. The advantage of their employment is the direct availability of historical load profiles from the time series database and the maintenance of data assured by regular updating processes (compare Section 3.2).

Finally, different data *Correction Algorithms* like online validation of model output might be applied and the forecasted data is computed. The described forecasting elements can be combined and used in an iterative way.

Data Management Challenges

Demand and supply forecasts with high accuracy are crucial to reduce the overall energy mismatch and penalties paid for any kind of imbalances, requiring sophisticated *data analytics* approaches.

On the demand side, very good results can already be achieved based on historical data, user experience and the application of simple mathematical functions [TE08]. Domestic and industrial consumption behavior usually follows certain, often repeated patterns, and dependence on exogenous influences is limited to outside temperatures, affecting only heating or cooling installations.

In contrast, the supply side is much more challenging and becoming increasingly important. In 2010, the total share of renewables in the global electricity production had already reached 19.4%. However, as many countries still do not accomplish their final capacity targets [Ren12], this number is expected to increase quickly; mostly driven by additional wind and solar power installations. Both of them are numerous, decentralized and heavily depending on weather conditions like wind speed and -direction, global heating or cloud coverage. Especially the last circumstance adds a new dimension to energy forecasting and is the main reason why in the past years a lot of research has been conducted to improve prediction quality. Advanced approaches like the appliance of ensemble models to increase robustness and accuracy are developed.

The need for fast response times to react to new market situations as well as continuous streams of new demand and supply measurements leads to additional *data integration* and *scalability* challenges. A tight coupling and the integration of energy forecasting within the databases avoids data transfer and enables the usage of existing database optimization structures and techniques [FRL12]. The creation of complex and multiple energy models requires optimizations to handle real-time mass prediction processes. Pre-calculation and materialization of stochastic models allows for fast output generation but also requires the

development of efficient maintenance strategies as new data becomes available [DBLH11]. Finally, the granularity of the forecast models (e.g., single wind installations vs. complete regions) needs to be carefully selected as it strongly influences the quality as well as the efficiency of the output.

4.3 Energy Saving

Besides the integration of decentralized and/or renewable energy sources, the efficient use of energy is considered to be the second key component of a sustainable energy policy. All technical and organizational efforts to reduce primary energy consumption with the motivation of reducing costs or emissions can be summarized by this term. These goals are mainly achieved by adopting more efficient industrial production technologies or processes [Die07], affecting all stages of the energy chain: generation, transformation, distribution and final consumption. Most of the introduced measures focus on the public transport and building sectors, where the highest potential for savings is expected [Eur11b].

In many markets, particularly in those with low primary energy prices, customers are still unaware of existing energy saving propositions. The smart meter roll-out (compare Section 4.1) is expected to encourage consumers to manage their energy use more efficiently. As for the EDMS, we will concentrate on providing the informational base and analytical equipment needed to (1) identify any possible saving potentials, to (2) measure the impact of adopted improvements on energy consumption and (3) for permanent process monitoring. These are the key activities required in order to establish active private or corporate energy controlling and waste prevention policies.

Data Management Challenges

A typical *data analytics* use case can be found in retailing companies like supermarket chains. Due to centralized management and procurement, all retailers are supposed to have equal technical equipment and to follow the same daily operational processes. To eliminate most of the external influences, store clusters have to be defined based on context information like geographical regions and selling areas in use or average yearly revenues. Now in theory, all markets in a cluster should have identical load curves but in practice they will not, as there are always variances. Storage managers can compare their energy consumption patterns within a cluster to identify process problems and energy saving opportunities.

Consider the real-world example in Figure 11, where the values of four different load curves in one cluster are visualized, representing the best, average, worst energy consumption in the concerning cluster. Additionally, the load curve of a single store (me) is shown. Energy saving potentials are marked whenever the energy consumption of the store strongly differs from the reference curves. Data mining approaches like clustering or time series similarity, e.g. as demonstrated in the work of Misiti et al [MMOP10], help to automatically identify such saving potentials even on single appliance level.

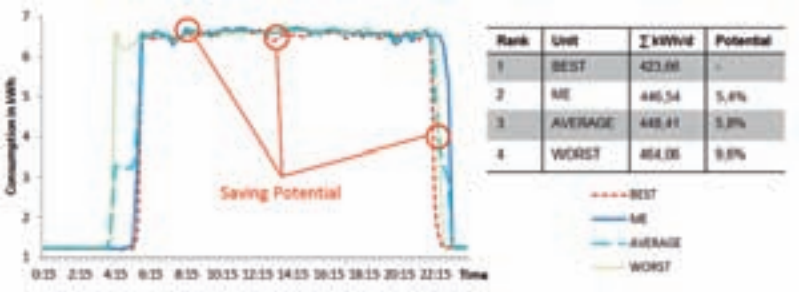


Figure 11: Exemplaric daily load curve cluster analysis

4.4 Mobile Consumption Devices

There are many research and development activities going on in the area of mobile consumption of electrical energy, also called *eMobility*. These efforts mainly rely on ecological aspects. Governments are trying to implement cleaner private and public transportation systems by using the prospective surplus production of renewable energy instead of climate-damaging, dwindling fossil fuel resources.

Data Management Challenges

Generally, information and communication technology is seen as the key enabler for eMobility, offering a multitude of basic and advanced services. The integration of these services will allow the usability for the end user without regional limitations, for example the adaption of roaming processes already known from the telecom business or recharging location services. Before any large-scale eMobility rollouts can take place, trans-regional standards have to be established for suitable infrastructures, particularly concerning regulatory frameworks, physical charging equipment and above all, the underlying information technology. The entire concept is dependent on a connected, interoperable, flexible backend application and infrastructure system that is simultaneously used for data recording, as well as a platform for the future implementation of value-added services. Case studies like [IT12] show that a robust, open architecture is essential for the success of an eMobility project, concerning both consumers and service providers. Users must be automatically identified and data from charging stations has to be collected in order to assess the driving pattern and charging behavior of each participant. With the corresponding *data analytics* methods, the information can be used by the utility company for (1) billing purposes, (2) to create demand forecasts for mobile energy (Section 4.2) and (3) to develop attractive pricing models like time-of-use tariffs or monthly flat fees.

On-Board Metering is one approach to reduce these immense infrastructure spendings. Instead of the expensive installation and maintenance of meters and communication units in dedicated public charging stations, the vehicles will be equipped with all required measuring devices [Lan12]. This enables vehicle-to-grid communication and initiates the transformation of today's static meter point installations into mobile ones, which are no longer assigned to a specific distribution network and therefore harder to handle by the DSOs who

follow conventional energy distribution processes. The *data integration* challenge here is that the contract entity can not longer be statically related to an explicit meter point, and the meter point's current location will be unknown. Although representing a new problem in the energy domain, this is an issue already being addressed by the research topic of *moving objects databases*, resulting for example in dedicated applications as such for cellphone providers or public transport information systems (e.g., [WSX+99]), wherefrom related concepts and ideas might be adopted.

4.5 Smart Grids

Finally, we turn our focus on the *Smart Grid*, one of the energy sector's latest fashionable terms, frequently used by scientists, industrial leaders and politicians. The general vision is to optimize the allocation of limited energy resources by using modern information and communication technology.

Figure 12: Combined networking elements in a smart distribution grid

A smart grid can briefly be described as a technically implemented demand response concept, achieved by connecting and monitoring conventional and renewable energy producers, consumers and flexible storage units as demonstrated in Figure 12. This is done via an intelligent interactive network, principally based on smart meters (Section 4.1) and internet technology. The idea behind this is to match energy supply and demand within the grid, by using directed, short-term interventions coordinated by a data management system which is serving as the centralized brain or steering unit. This is the point where we consider the EDMS with an integrated AMR module as a possible option to take part in.

Data Management Challenges

One challenge is to scale down its level of operations from complete TSO or DSO networks to smart grids or even further, to single smart homes forming a combined energy supply micro-system. At first appearance, this might not be an appropriate use for the complex EDMS design idea, but using *multi-tenancy* optimization to intelligent share and allocate the system resources allows the simultaneous management of many parallel or connected grids within the same database system.

Another important element of the smart grid we have not discussed yet is the energy storage unit. With the increasing share of fluctuating renewable energy sources, the resilience of distribution networks will be tested. The ability to store and allocate produced energy accordingly is an important matter to maintain network stability in peak situations and to enable possible load shifting options, but still remains difficult due to the physical characteristics of electrical power. Either classical pump storage stations or modern air compression storages are restricted to accordingly conditioned geographical regions; today's common batteries are slow-charging and low-capacity units and therefore not really able to create a significant impact on the daily load balance.

By using the smart grid communication network, all of these usually isolated micro-units can be connected in a cloud. Combined with small and flexible cogeneration units, they form a virtual power plant which can be remotely controlled by the central data management unit [PRS07]. In order to ensure the successful coordination of hundreds or thousands of such small supply and storage units, all data streams must be real-time transmitted and analyzed instantly (*data analytics*). In first place this allows an on-line validation of pre-generated load curve forecasts, followed by immediate reactions in the form of executing corresponding events like additional supply orders.

Second, the *integrity* of processed master and transaction data must be monitored at an absolutely high level to keep track of every single grid element's status. This implies plenty of work for the communication gateway, which has to manage the synchronization between the central system and all meters and data concentrators in the grid, so the modules scalability combined with parallelized data processing by the central steering and control unit is a critical factor.

5 Conclusions

In the above sections we have shown that currently available EDMS are highly specialized, standardized and powerful database applications, capable to handle mass data exchange processes and thereby satisfying most of today's energy market's requirements. However, along with the ongoing energy market's transformation, new technical challenges remain to be solved. These are basically bundled in the terms of data analytics, data and/or system integration using adequate communication technology, advanced data storage and compression techniques, and scalability optimization approaches.

For future developments, we think that additional aspects, such as data security and robustness of the chosen architectures need to be addressed. Furthermore, we consider transaction process optimization and the integration of new standard core components into the centralized EDMS framework to be the most promising implementation directions, thereby creating a close link between data and functionality. These will enable the system to handle operations more efficiently which is critical for future success, like discussed above against the background of automatic meter reading and forecasting functionality. Besides, as exemplarily described in association with mobile energy consumption, there exists the possibility of adopting existing solutions obtained while treating similar prob-

lem constellations in different market domains. After all, energy data management is a promising field for database researchers and developers, offering many opportunities for future improvements in form of interesting challenges.

Acknowledgment

Part of the work presented in this paper was funded by the European Regional Development Fund (EFRE) under co-financing by the Free State of Saxony and Robotron Datenbank-Software GmbH.

References

[BDD+12] M. Böhm, L. Dannecker, A. Doms, E. Dovgan, B. Filipic, U. Fischer, W. Lehner, T. B. Pedersen, Y. Pitarch, L. Siksnys, and T. Tusar. Data management in the MIRABEL smart grid system. In *EDBT/ICDT Workshops*, pages 95–102, 2012.

[DBLH11] L. Dannecker, M. Böhm, W. Lehner, and G. Hackenbroich. Forcasting evolving time series of energy demand and supply. In *Proceedings of the 15th international conference on Advances in databases and information systems*, ADBIS'11, pages 302–315, Berlin, Heidelberg, 2011. Springer-Verlag.

[Die07] M. Diesendorf. *Greenhouse Solutions With Sustainable Energy*. NewSouth Publishing, 2007.

[Eur11a] European Commission. Energy Roadmap 2050. http://ec.europa.eu/energy/energy-2020/roadmap/index_en.htm, 2011.

[Eur11b] European Commission. Proposal for a Directive on energy efficiency. http://ec.europa.eu/energy/efficiency/eed/eed_en.htm, 2011.

[FRL12] U. Fischer, F. Rosenthal, and W. Lehner. F2DB: The Flash-Forward Database System. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 1245 –1248, april 2012.

[GGFS11] V. Giordano, F. Gangale, G. Fulli, and M. Sánchez. Smart Grid projects in Europe. http://ses.jrc.ec.europa.eu/smart-grid-catalogue, 2011.

[GV12] T. Goette and K. Vortanz. Smart Home als Schlüsselrolle für Smart Metering. *e—m—w Zeitschrift für Energie, Markt, Wettbewerb*, 3:10–13, 2012.

[Has07] B. Hasche. Analyse von Prognosen der Windgeschwindigkeit und Windstromeinspeisung. Technical report, IER Institut für Energiewirtschaft und Rationelle Energieanwendung, Universität Stuttgart, 2007.

[HHM11] M. Hashmi, S. Hanninen, and K. Maki. Survey of smart grid concepts, architectures, and technological demonstrations worldwide. In *Innovative Smart Grid Technologies (ISGT Latin America), 2011 IEEE PES Conference on*, pages 1 –7, oct. 2011.

[HILM02] H. Haciguemues, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over Encrypted Data in Database-service-provider Model. *Proc. of the ACM SIGMOD Conference on Management of Data*, June 2002.

[IT12] C. Isernhagen and D. Tarafdar. Best Practices: e-Mobiliy- Challenges and Opportunities: A Case Study from Singapore., 2012.

[JJP+10] M. Jahn, M. Jentsch, C.R. Prause, F. Pramudianto, A. Al-Akkad, and R. Reiners. The Energy Aware Smart Home. In *Future Information Technology (FutureTech), 2010 5th International Conference on*, pages 1 –8, may 2010.

[JP05] T. Jamasb and M. Pollitt. Electricity Market Reform in the EU: Review of Progress toward Liberalization & Integration. *The Energy Journal*, 26:11–41, 2005.

[Lan12] B. Lange. Unter Spannung: Die IT hinter der Lade-Infrastruktur für E-Autos. *iX Magazin für professionelle Informationstechnik*, 7:87–93, 2012.

[MMOP10] M. Misiti, Y. Misiti, G. Oppenheim, and J.-M. Poggi. Optimized clusters for disaggregated elecricity load forecasting. *REVSTAT*, 8:105–124, 2010.

[NPS+12] J. Nitsch, T. Pregger, Y. Scholz, T. Naegler, D. Heide, D. Luca de Tena, F. Trieb, K. Nienhaus, N. Gerhardt, T. Trost, A. von Oehsen, R. Schwinn, C. Pape, H. Hahn, M. Wickert, M. Sterner, and B. Wenzel. Long-term scenarios and strategies for the deployment of renewable energies in Germany in view of European and global developments. http://www.dlr.de/tt/Portaldata/41/Resources/dokumente/institut/system/publications/leitstudie2011_kurz_en_bf.pdf, 2012.

[PBB+09] E. Peeters, R. Belhomme, C. Batlle, F. Bouffard, S. Karkkainen, D. Six, and M. Hommelberg. ADDRESS: Scenarios and architecture for Active Demand development in the smart grids of the future. In *Electricity Distribution - Part 1, 2009. CIRED 2009. 20th International Conference and Exhibition on*, pages 1 –4, june 2009.

[PRS07] D. Pudjianto, C. Ramsay, and G. Strbac. Virtual power plant and system integration of distributed energy resources. *Renewable Power Generation, IET*, 1(1):10 –16, march 2007.

[Ren12] Renewable Energy Policy Network for the 21st Century. Renewables 2011 - Global Status Report. http://www.ren21.net/REN21Activities/Publications/GlobalStatusReport/GSR2011/tabid/56142/Default.aspx, 2012.

[Ryb12] T. Ryberg. Smart Metering in Western Europe, Seventh Edition. Technical report, Berg Insight, 2012.

[Sto12] D. Stolarski. Marktüberblick Energiedatenmanagement. *e—m—w Zeitschrift für Energie, Markt, Wettbewerb*, 3:99–109, 2012.

[TE08] J. Taylor and A. Espasa. Energy forecasting. *International Journal of Forecasting*, 24(4):561–565, 2008.

[US 12] US Department of Energy. Smart Grid Investment Grant Program, Progress Report. http://www.smartgrid.gov/sites/default/files/doc/files/sgig-progressreport-final-submitted-07-16-12.pdf, july 2012.

[WSX+99] Ouri Wolfson, Prasad Sistla, Bo Xu, Jutai Zhou, and Sam Chamberlain. DOMINO: databases fOr MovINg Objects tracking. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, SIGMOD '99, pages 547–549, New York, NY, USA, 1999.

Leistungsorientierte Auswahl von Reorganisationskandidaten

Kevin Röwe[1], Fritz Walliser[1], Norbert Ritter[2]

[1]InfoDesign GmbH
Großes Feld 23
25421 Pinneberg
Kevin.Roewe@infodesigner.biz
Walliser@infodesign-msx.de

[2]Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Str. 30
22527 Hamburg
Norbert.Ritter@informatik.uni-hamburg.de

Abstract: Datenbank-Reorganisationen stellen moderne IT-Abteilungen vor massive Herausforderungen. Auf der einen Seite versprechen sie die Bereinigung von durch Degenerierung verursachten Leistungseinbußen, auf der anderen Seite verbrauchen Reorganisationsprozesse massiv an anderer Stelle dringend benötigte Ressourcen. Aktuell verwendete Ansätze zur Auswahl von Reorganisationskandidaten (d.h. von Speicherbereichen, deren Reorganisation potenziell von Nutzen ist) nehmen kaum eine Erfolgsbewertung und –messung vor, sondern beruhen größtenteils auf statischen Verfahren. Dies gibt grundsätzlich Anlass zur Überprüfung. Dieser Beitrag zeigt die Schwächen aktueller Ansätze auf und beschreibt einen in Zusammenarbeit der Universität Hamburg mit der Firma InfoDesign entwickelten Ansatz einer leistungsorientierten Auswahl von Reorganisationskandidaten. Dieser basiert auf der systematischen Erhebung von geeigneten Leistungskennzahlen und dem Prognostizieren der durch Reorganisation erreichbaren Leistungssteigerungen. Evaluationen zeigen, dass die Anzahl von Reorganisationen deutlich reduziert werden kann, in dem nur tatsächlich zu Leistungssteigerungen führende Reorganisationskandidaten ausgewählt werden.

1 Einführung

In betrieblichen Systemumgebungen kommt es im Laufe der Zeit insbesondere bedingt durch die verschiedenartigen Manipulationsmöglichkeiten der Datenbasis zu einer Verschlechterung der Leistung. In diesem Fall ist eine Reorganisation der Datenbank sinnvoll. Bei einer Reorganisation werden die Daten mit dem Ziel einer Leistungsverbesserung in einer optimalen physischen Reihenfolge neu angeordnet. In hochverfügbaren Systemlandschaften können diese Optimierungen aus Performancegründen in der Regel nicht im laufenden Betrieb während der Kernarbeitszeiten vorgenommen werden, so dass diese regelmäßig (z. B.) am Wochenende stattfinden. Durch das begrenzte Zeitfenster, das zusätzlich noch mit anderen Wartungsarbeiten, wie z. B. dem Backup, geteilt werden muss, und häufig sehr großer Datenmengen ist eine sorgfältige Auswahl der zu reorganisierenden Einheiten und der zugehörigen Reorganisationszeitpunkte geboten. Derzeit existieren Werkzeuge, die dem Administrator entsprechende Reorganisationskandidaten (d.h. von Speicherbereichen, deren Reorganisation als potenziell nützlich

erachtet wird) vorschlagen. Es bestehen jedoch erhebliche Zweifel daran, dass diese Empfehlungen „optimal“ sind, d.h. genau solche Reorganisationen vorgeschlagen werden, die auch tatsächlich einen bedeutsamen Leistungsgewinn herbeiführen; vielmehr besteht die Befürchtung, dass unnötigerweise zu viel reorganisiert wird. Die Firma InfoDesign hat im Rahmen einer Kooperation mit der Uni Hamburg aktuelle Lösungen zur Reorganisationseinplanung und -vermeidung untersucht, Schwachpunkte identifiziert und darauf basierend einen alternativen Lösungsansatz zur Auswahl der zu verwendenden Reorganisationskandidaten entwickelt. Der Lösungsansatz wurde technisch implementiert und im Praxistest evaluiert. Er wird in diesem Artikel vorgestellt.

Die einfache Grundidee dieses Ansatzes ist, dass eine Datenbank-Reorganisation dann als „erfolgreich“ betrachtet werden kann, wenn der auf der Datenbank bzw. auf dem entsprechenden Granulat stattfindende Workload nach Durchführung der Reorganisation sich in seinem Performanceverhalten beträchtlich verbessert. Das Performanceverhalten kann beispielsweise mit Kennzahlen, wie z. B. Antwortlaufzeitverhalten, I/O-/CPU-Aufwand, o.Ä. gemessen werden. Weiter sollen nur Datenbankgranulate als Kandidaten herangezogen werden, deren Leistungsverhalten sich seit der letzten Reorganisation über die Zeit hinweg verschlechtert hat. So einfach dieser Ansatz auch klingt, so wird dennoch eine solche Vorgehensweise bisher nicht zur Kandidatenauswahl verwendet. Vielmehr werden für die Auswahl häufig statische Regeln verwendet, die auf Grundlage der statistischen Informationen des Datenbankkatalogs arbeiten und aufgrund der (im Hinblick auf den Zweck der Reorganisation vorherrschenden) Unschärfe dieser Informationen häufig völlig unnötige, weil nicht zu wirklichen Leistungsverbesserungen führende, aber dennoch kostenintensive Reorganisationsvorgänge veranlassen.

2 Grundlagen

2.1 Datenbank-Reorganisation

Reorganisationen werden vor Allem mit dem Ziel der Performanceverbesserung durch Optimierung der Speicherausnutzung durchgeführt. In [SG79] werden unter anderen folgende Szenarien angeführt, in denen eine Reorganisation sinnvoll ist:

- Schemaänderungen, z.B. Hinzufügen eines neuen Attributs,
- starker Anstieg der Datenmenge,
- Partitionierung einer Datenbank in mehrere unabhängige Einheiten bzw. Entfernen einer großen Teilmenge von Daten aus einer Datenbank, zum Beispiel aufgrund einer veränderten Gesetzgebung,
- große Menge von Einfüge- und/oder Löschoperationen.

[SH94] wird in der Definition konkreter und beschreibt die Datenbank-Reorganisation als einen Prozess, der primär auf eine Verbesserung der Geschwindigkeit und Effizienz der Datenbank abzielt, da im Anschluss an eine Datenbank-Reorganisation die Daten in einer optimalen Anordnung auf dem Sekundärspeicher liegen. [WV09] nimmt eine gröbere Klassifikation der zugrunde liegenden Ursache der Notwendigkeit für eine Reorganisation vor; hierbei werden zwei Gründe der Durchführung benannt: „Verschlechterung der I/O-Performance“ und „Speicherplatz-Ausnutzung bzw. -Rückgewinnung“. Der

Schwerpunkt unserer Betrachtungen liegt auf der Überprüfung und Bewertung des Performanceverhaltens. Der Aspekt ‚Speicherplatzausnutzung und -rückgewinnung' wird nicht berücksichtigt, da die zughörigen Ursachen gut anhand von Statistiken aus dem Datenbankkatalog überprüfbar sind. [SH94] geht (weiter) davon aus, dass eine desorganisierte Anordnung von Daten zu einer signifikanten Leistungsverschlechterung des Datenbanksystems führt. Die desorganisierte Anordnung von Daten kommt dabei automatisch durch typische Datenbankoperationen im täglichen Betrieb zustande. Im Folgenden werden drei typische Betriebsprobleme nach [SH94] aufgezeigt, die in klassischen OLTP-Datenbanksystemen bei fortwährendem Betrieb entstehen.

Fragmentierung. Fragmentierung liegt vor, wenn logisch zusammengehörende Daten über einen größeren bzw. weit auseinander liegende Speicherbereiche verteilt werden. Insbesondere durch die Such- und Positionierungsoperationen der Lese- und Schreibköpfe ist mit Leistungseinbußen zu rechnen. Fragmentierungen können beispielsweise dann entstehen, wenn große Teile einer Datenbasis gelöscht wurden.

Row Chaining und Row Migration. Unter einer „chained row" versteht man einen Datensatz, der zu groß ist um in einer einzelnen Datenbankseite zu passen. Unter „row migration" versteht man ein ähnliches Problem. Der Unterschied ist, dass „row migration" insbesondere dann angewendet wird, wenn für ein Update des Datensatzes nicht ausreichend Platz in der ursprünglichen Datenbankseite zur Verfügung steht. Das Datenbanksystem lagert dann diesen Datensatz in einen anderen Bereich im Segment aus und verweist mit einem entsprechenden Pointer darauf. Ein Datensatz kann dann nicht mehr mit einer einzelnen physischen Datenbankoperation ausgelesen werden, sondern es müssen mehrere Operationen stattfinden, um das Auslesen zu ermöglichen. Bedingt dadurch kommt es zu einer signifikanten Drosselung der Geschwindigkeit [CM01].

Declustering. „Declustering" kommt vor, wenn kein Einfügeplatz für einen Datensatz in seiner logischen Reihenfolge über den Clustering-Schlüssel verfügbar ist. In diesem Fall versucht das Datenbankverwaltungssystem den Datensatz dort einzufügen, wo Platz ist. Darunter leiden insbesondere sequentielle I/O-Operationen [CM01].

Die genaue Implementierung einer Datenbank-Reorganisation bleibt in der Regel unklar, da die Umsetzungen von Datenbanksystemherstellern nicht offengelegt werden. Grundsätzlich wird zwischen zwei verschiedenen Varianten, der Offline- und der Online-Variante, unterschieden. Wie bereits angesprochen, betrachten wir vor Allem die (praxisrelevantere) Offline-Variante. Diese erfordert, dass das entsprechende zu reorganisierende Datenbankgranulat nicht im Datenbankzugriff ist. Vor allem bei Hochverfügbarkeitsanforderungen kann dies zu Problemen führen, da eine Reorganisation abhängig von der Größe eine signifikante Zeit in Anspruch nehmen kann. Die Funktionsweise der Offline-Variante wird nach [CM01] wie folgt beschrieben:

- Erstellung eines Backups der Datenbank,
- Exportieren der Daten (z.B. in eine temporäre Datei),
- Löschen der Datenbankobjekte,
- Erstellen der zuvor gelöschten Datenbankobjekte,
- Sortieren der exportierten Daten nach dem Clustering Key,
- Importieren der vorherigen Schritt sortierten Daten.

Anschließend werden alle Indizes neu aufgebaut. Nach der Reorganisation befinden sich die Daten in einer „optimalen“ physischen Anordnung und Workload-typische Phänomene wie Fragmentierung, Row Chaining und Declustering sind weitgehend ausgeschlossen, was wiederum in einem besseren Leistungsverhalten resultiert.

2.2 Der InfoMat

Die Firma InfoDesign ist unter anderem als Software-Tool-Hersteller im Bereich von IBM-DB2-Datenbank(sytem)en im Mainframe-Umfeld erfolgreich am Markt. Eine wichtige Zielsetzung ihrer Softwareprodukte ist dabei die Automation der DB2-Administration. Das Produkt „InfoMat“ unterstützt aktiv den Datenbank-Administrator in den Bereichen Datenbank-Reorganisation, Durchführung von „Runstats“, „Copies“ und „Modifies“ sowie bei „Backup“ und „Recovery“. Kunden, die den InfoMat produktiv einsetzen, verfügen typischerweise über eine IT-Landschaft, in der ein SAP-ERP-System eingesetzt und dieses auf einer DB2-Datenbank auf dem Mainframe-Betriebssystem z/OS läuft. Damit bewegen sich die Umfänge in dem Bereich einer „großen Installation“, die nicht mehr rein manuell administrierbar ist. Allein ein typisches SAP-System umfasst dabei in der Grundinstallation bereits etwa 100.000 Tabellen, wobei häufig sogar mehrere SAP-Systeme auf einem Mainframe laufen.

Das Reorganisationsmodul des InfoMat setzt hinsichtlich der Auswahl geeigneter Reorganisationskandidaten einen schwellwert-basierten Ansatz um. Dabei werden die zur Entscheidungsfindung zu betrachtenden Informationen größtenteils aus dem Statistik-Modul des Datenbankkatalogs gewonnen. Im InfoMat wurden hierzu 17 Regeln definiert (Tabelle 1), die – sollte eine dieser Bedingungen zutreffen – zur Einplanung des betroffenen Tablespace in die Reorganisation führen.

RB*	Beschreibung
RB02	Index verfügt über zu viele Extents
RB03	Tablespace verfügt über zu viele Extents
RB04	Aktive Seiten < 90% allokierte Seiten
RB05	Größter Katalog/DSNSpace unverhältnismäßig
RB06	10% Index-Seiten-Splits
RB07	LEAFDIST – Seiten zu weit auseinander
RB08	Clusterratio – Daten schlecht sortiert
RB09	Tabelle nicht in Clustered Folge

RB*	Beschreibung
RB10	20% Datenveränderung
RB11	DDL-Änderungen
RB12	GROESSE > MAXPRIQTYNP
RB13	Durch Compression eingestellt
RB14	Manuell (INFOREOL)
RB20	Reordered Row Format
RB25	Index Overallocation
RB30	Reorg mit Discard

Tabelle 1: Reorganisationsbedingungen des InfoMat / RB = Reorganisationsbedingung

Von uns durchgeführte, empirische Untersuchen zeigen, dass knapp 80% aller Reorganisationen auf die Regeln RB07 und RB10 zurück gehen. Leider besteht insbesondere an der Sinnhaftigkeit von RB10 großer Zweifel, da diese allein auf dem Entscheidungskriterium beruht, dass es im entsprechenden „Tablespace“ mehr als 20 Prozent Datenveränderungen gab. Mengenorientierte Änderungen von z.B. nicht indizierten Datenfeldern ohne größenmäßige Erweiterung derselben oder Einfügeoperationen großer Anzahl in Archiv-Tabellen würden diese Regel beispielsweise unnötigerweise auslösen.

2.3 Alternative Ansätze und Einordnung der InfoMat-Lösung

Die in Kapitel 2.2 aufgezeigten Reorganisationsbedingungen sind größtenteils nicht durch eigene Erfahrungswerte oder Forschungen der Firma InfoDesign entstanden, sondern beruhen auf Empfehlungen von SAP und IBM, die insbesondere auf die Charakteristika des SAP-Systems zurückzuführen sind. Die Anwender sind angehalten, diesen Empfehlungen zu folgen. Anwender, die den InfoMat und/oder SAP nicht einsetzen, behelfen sich in der Regel mit ähnlichen Schedulern oder haben eigene Skripte entwickelt, die den Administratoren bei der Umsetzung dieser oder ähnlicher Regeln behilflich sind. Typische Parameter für die InfoMat-unabhängige Einplanung sind dabei die Cluster Ratio, Overflow (Record Split), die Anzahl der Extents sowie Komprimierungsaspekte (Compress). Schwellwerte wurden dabei durch Erfahrungen und Beobachtungen festgelegt.

Im Bereich von DB2-LUW können Reorganisationskandidaten durch das IBM-eigene Tool REORGCHK ermittelt werden. REORGCHK arbeitet ähnlich schwellwert-basiert wie der InfoMat und verwendet hierzu standardmäßig 8 vordefinierte Regeln, die gegen den Datenbankkatalog geprüft werden. Die überprüften Parameter überlappen stark mit denen des InfoMats.

Das Thema Datenbank-Reorganisationen auf Oracle wird nach [QE10] sehr kontrovers diskutiert. QE10 führt vier Gründe für eine Reorganisation auf Oracle an, wobei die ersten 3 wiederum schwell-basiert umgesetzt werden:

- Schachtelungstiefe des Index,
- Migrated Rows,
- Archivierung und damit Löschen von Tabelleninhalten,
- Umstellung auf ein anderes Layout / Strukturänderungen.

Neben den hier beschriebenen sind uns keine grundlegend anderen Ansätze – weder in Praxis noch in der Wissenschaft – bekannt.

3 Auswahl von Reorg-Kandidaten nach Leistungsgesichtspunkten

Die zentrale Idee unseres neuen Lösungsansatzes ist es, die Entscheidung abhängig von der Performance-Entwicklung des SQL-Workloads zu machen. Hierzu bedarf es einer Protokollierung von geeigneten Leistungskennzahlen über die Zeitachse hinweg. Es wurden verschiedene Ansätze (u.a. automatisierte Durchführung eines vordefinierten SQL-Workloads unter Messung des Antwortzeitverhaltens, Nutzung des „DB2 Event-Monitors“, Auslesen von Daten auf Grundlage des „DB2 Dynamic Statement Cache“) untersucht. In Abwägung der jeweiligen Vor- und Nachteile wurde der Ansatz der Nutzung des „Dynamic Statement Cache“ (DSC) weiter verfolgt.

Der DSC ist ein geschützter Speicherbereich im DB2-System und zentraler Bestandteil des sogenannten EDM-Speichers. In dem EDM-Speicher ist auch der „Buffer Pool“ angesiedelt. Eine der Aufgaben des DSC ist es, einmal generierte Zugriffspläne von

dynamischen SQL-Anfragen zwischen zu speichern und für eine Wiederverwendung vorzuhalten. Wird eine SQL-Anfrage von einem Benutzer oder einem Programm aufgerufen, so wird zunächst einmal geprüft, ob der Ausführungsplan zu der entsprechenden SQL-Anfrage bereits zwischengespeichert wurde. Im Trefferfall kann der Zugriffsplan ohne erneute Generierung verwendet werden. Die Generierung eines Zugriffsplans kann insbesondere bei komplexen SQL-Anfragen in einer hohen CPU-Auslastung resultieren. So kann der Prozess bis zu knapp 90 Prozent der Transaktions-CPU-Kosten in Anspruch nehmen. Durch den DSC wird in der Regel eine erhebliche Performancesteigerung erreicht. Die im DSC gespeicherten dynamischen SQL-Anfragen enthalten benötigte Leistungskennzahlen zu den CPU-Kosten (Durchschnittswerte) und der Ausführungshäufigkeit der SQL-Anfragen. Die Werte liegen bereits in aggregierter Form vor. Wurde das „Prepared Statement“ in Verbindung mit Host-Variabeln verwendet, werden Literale in den SQL-Anfragen bereits durch ein spezielles Symbol (‚?‘) ersetzt, sodass hier der DSC bereits eine Reduzierung der Menge von SQL-Anfragen vornimmt und somit den „Logging Overhead“ vermindert.

Der DSC verwendet einen LRU-Algorithmus zur Verdrängung. Das heißt, die Anfrage wird verdrängt, die am längsten nicht mehr verwendet wurde. Wünschenswert wäre sicherlich ein LFU-Algorithmus, der die am seltensten benutzten Anfragen verdrängt. Da ein LFU-Algorithmus jedoch leider nicht implementiert wurde, muss ein alternativer Ansatz eingesetzt werden. In diesem Fall haben wir uns dafür entschieden, regelmäßig „Snapshots“ des DSC zu erzeugen. Durch das regelmäßige „Snapshotting“ des DSC erhält man einen guten Überblick über wichtige und repräsentative SQL-Anfragen. Es ist trotzdem nicht auszuschließen, dass im Falle eines ungünstigen Workloads eine Verzerrung stattfindet und womöglich zu einem oder mehreren Snapshot-Zeitpunkten nur nicht-repräsentative Anfragen aus dem DSC gewonnen werden können. Aus diesem Grund ist das Snapshotting-Intervall dynamisch und abhängig von dem Verdrängungsfaktor, der nach jedem Snapshot ermittelt wird. Steigt die Verdrängung, wird die Snapshot-Frequenz erhöht.

Die Inhalte des DSC residieren im EDM-Speicher. Mittels eines speziellen (SQL-) Kommandos (EXPLAIN STMTCACHE ALL) können die Daten in eine relationale Tabelle geschrieben werden. Wesentliche Tabellenfelder des DSC sind hierbei:

- eine eindeutige STMT_ID,
- der STMT_TEXT,
- die Ausführungshäufigkeit der SQL-Anfrage seit Aufnahme in den DSC (STAT_EXEC),
- die durchschnittliche Anzahl der zurück gelieferten Datensätze der SQL-Anfrage (STAT_PROW).
- die kumulierten CPU-Ausführungskosten (STAT_CPU).

Da die Entscheidung über die Notwendigkeit einer Datenbank-Reorganisation auf Werte in der Vergangenheit zurückgreifen muss, wurde eine entsprechende Historisierung vorgesehen, d.h. den aus dem DSC erhobenen Statement-Informationen werden entsprechende logische Zeitstempel hinzugefügt; hierzu folgendes Beispiel. Tabelle 2 zeigt

den Inhalt des DSC zum Zeitpunkt t, Tabelle 3 zum Zeitpunkt t+1. Tabelle 4 entspricht der transformierten Historisierungstabelle.

SQL-Anfrage	STAT_EXEC	STAT_CPU
Select * from Kunde where KdNr = ‚1011‘	1452	0.23
…		

Tabelle 2: Inhalt des DSC zum Zeitpunkt t

SQL-Anfrage	STAT_EXEC	STAT_CPU
Select * from Kunde where KdNr = ‚5311‘	1712	0.26
…		

Tabelle 3: Inhalt des DSC zum Zeitpunkt t+1

SQL-Anfrage	Zeitpunkt	STAT_EXEC	STAT_CPU
select Name from Kunde where KdNr = ‚?‘	t	1452	0.23
select Name from Kunde where KdNr = ‚?‘	t+1	1712	0.26
…			

Tabelle 4: Transformierte Historisierungstabelle

In diesem Beispiel wurde die konkrete Wertausprägung des Feldes KdNr in der WHERE-Bedingung durch ‘?’ ersetzt. Dies geschieht durch einen nachgelagerten SQL-Parsing-Prozess. Die transformierte Tabelle protokolliert zunächst einmal nur die extrahierten Werte aus dem DSC. Für eine weitere Auswertung sind die Daten noch nicht direkt verwendbar, da es sich bei den Werten STAT_CPU um Durchschnittswerte handelt – erhoben über die gesamte Laufzeit seit Aufnahme der SQL-Anfrage in den DSC. Die Entwicklung wird hierbei noch nicht korrekt aufzeigt. So ist zwar zu erkennen, dass zwischen den Zeitpunkten t und t+1 sich die durchschnittliche Zeit zur Abarbeitung der Anfrage verschlechtert hat, das genaue Ausmaß wird jedoch nicht aus der Differenz 0,03 (0,26 - 0,23) direkt ersichtlich. Die tatsächliche Verschlechterung zwischen den beiden Zeitpunkten wird durch die Überprüfung mit folgender Formel erkenntlich:

$$STAT\_EXEC(t) * STAT\_CPU(t) + ((STAT\_EXEC(t+1) - STAT\_EXEC(t)) * x = STAT\_EXEC(t+1) * STAT\_CPU(t+1)$$

In diesem Fall war die durchschnittliche Antwortzeit mit 0,43 CPU-Sekunden und einer Verschlechterung von fast 86 Prozent gegenüber dem vorherigen Zeitpunkt t deutlich verändert. Ziel der Transformation ist es, Aussagen über das Antwortlaufzeitverhalten von SQL-Anfragen entlang der Zeitachse beurteilen und analysieren zu können. Essentiell wichtig ist hier allerdings auch die Zuordnung der einzelnen SQL-Anfragen zu dem entsprechenden Datenbank-Reorganisationsgranulat. Im DB2-System findet die Reorganisation auf Tablespace-Basis statt. Das hat zur Folge, dass zu jeder protokollierten SQL-Anfrage der entsprechende Tablespace durch Abfrage des Datenbankkataloges herausgefunden werden muss. Sind diese SQL-Anfragen zugeordnet, ist der absolut gemessene Wert, z. B. für die aufgebrauchte CPU-Zeit, nicht mehr relevant. Vielmehr interessiert die relative Entwicklung (in Prozent) über die Zeit. Es ist also wünschenswert, eine Reduktion auf Tablespace-Basis durch Durchschnittswerterrechnung zu erreichen. Das sieht beispielhaft wie in Tabelle 5 dargestellt aus. Für den Tablespace TS1 fließen drei SQL-Anfragen in die Betrachtung ein. Hier kann man nun die prozentualen Veränderungen ausrechnen und die Werte normieren. Man sieht, dass sich die CPU-Zeit der SQL-Anfrage S1 zu jedem Zeitpunkt um 10 Prozent verschlechtert hat. Die

Antwortzeiten der SQL-Anfragen S2 und S3 haben sich wiederum stark verbessert; jede SQL-Anfrage pro Zeitpunkt um 50 Prozent. Bildet man nun einfach einen Durchschnittswert pro SQL-Anfrage und pro Zeitpunkt, so dominieren die Verbesserungen mit 50 Prozent der beiden (selten ausgeführten) SQL-Anfragen S2 und S3. Die aus einer ungewichteten Betrachtung der Tabelspaces gezogene Schlussfolgerung, nicht zu reorganisieren, wäre falsch. Notwendig ist eine gewichtete Berechnung, in die die Ausführungshäufigkeit der SQL-Anfragen mit eingeht. Ebenfalls ist es sinnvoll, die Kardinalität der Ergebnismenge mit der entsprechenden Gewichtung zu verknüpfen.

Tablespace	SQL-Anfrage	Anzahl ausgeführt	Durchschnittliche CPU-Zeit in Sekunden zum Zeitpunkt			
			t1	t2	t3	t4
TS1	S1	8000	0,4	0,44	0,484	0,5324
TS1	S2	2	500	250	125	62,5
TS1	S3	4	1000	500	250	125
TS2	S4	2	2	3	3,4	4
…						

Tabelle 5: Auf Tablespace reduzierte Durchschnittsberechnung

Durch die Historisierung werden CPU-Zeiten zu einzelnen SQL-Anfragen gesammelt. Diese werden anhand des Reorganisationsgranulats ‚Tablespace' logisch gruppiert.

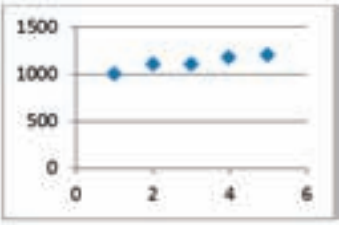


Abbildung 1:
Beispielverlauf CPU-Kosten

Abbildung 1 stellt exemplarisch den Werte-Verlauf einer Beispiel-SQL-Anfrage grafisch dar. Die Grafik zeigt eine stetige Verschlechterung der zu Beginn auf den Wert 1000 normierten CPU-Kosten pro zurück gelieferten Datensatz über die Zeit. Betrachtet man diesen Verlauf, so liegt die Vermutung nahe, dass hier eine Datenbank-Reorganisation sinnvoll und eine Absenkung der CPU-Kosten auf das Startniveau (1000) denkbar ist.

Erste Tests des Prototypen haben ergeben, dass die historisierten Leistungskennzahlen teilweise schwer maschinell auszuwerten sind, da diese von großen Streuungen und Lücken in der Messhistorie geprägt sein können. Im Folgenden werden diese Probleme kurz verdeutlicht. Grobe Streuungen könnten zum Beispiel entstehen, wenn einzelne SQL-Anfragen relativ selten ausgeführt werden. Hier fallen Spitzen in der Datenbankauslastung besonders negativ ins Gewicht, so dass die Kennzahlen dadurch stark verzerrt werden können. Die Verwendung eines Schwellwertes für die minimale Ausführungshäufigkeit konnte Verläufe dieser Art drastisch reduzieren.

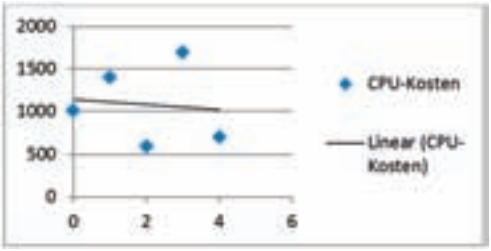


Abbildung 2: Gestreuter Beispielverlauf

Im Falle des Einsatzes der Linearen Regression [BH06] – wie in der Abbildung 2 angewandt – ist des Weiteren die Berechnung der Standardabweichung oder des Bestimmtheitsmaßes von der Trendlinie interessant. Unsere Untersuchungen haben Folgendes gezeigt: wenn die Standardabweichung oder das Bestimmtheitsmaß zu hoch ist, dann sollte die Funktionsannäherung nicht in weitere automatische Auswertungsverfahren einbezogen werden.

Abbildung 3 zeigt eine Messhistorie auf Grundlage von vier Messwerten, die in einem Zeitraum von 14 Tagen gesammelt wurden. Eine Entscheidung auf Grundlage dieser wenigen Messpunkte ist hier sehr fragwürdig. Es wurden geeignete Schwellwerte auf Basis empirischer Untersuchungen festgelegt, um diese Art von Unschärfe nicht mit in eine Entscheidung einfließen zu lassen.

Datenbank-Reorganisationen werden bei den Kunden der Firma InfoDesign in der Regel in einem festgelegten Wartungszeitfenster durchgeführt. Dieses Wartungszeitfenster liegt in der Regel am Wochenende, da zu diesem Zeitpunkt eine deutlich verminderte OLTP-Last anliegt. Die Reorganisationsentscheidung des InfoMat wird aufgrund der statischen Regeln freitags nach Ende der Kernarbeitszeit der Mitarbeiter getroffen.

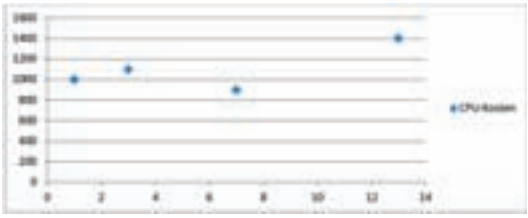


Abbildung 3: Dünn besetzter Beispielverlauf

Welche Zeiträume müssen nun aber mit einem maschinellen Verfahren bewertet werden? Ausgehend davon, dass Messdaten von Montag bis Freitag gesammelt werden, wird nach Ablauf der Kernarbeitszeit am Freitag eine Entscheidung auf Basis der über die Woche gesammelten Werte getroffen. Kommt es zu einer positiven Reorganisations-Entscheidung, kann die Bewertung in der darauf folgenden Woche wieder analog angewendet werden. Wie sieht es aber aus, wenn keine Reorganisation notwendig ist?

Prüft man die Reorganisationsbedürftigkeit des Verlaufs in Abbildung 4 nach Ablauf der ersten Woche, so kommt man zu der Entscheidung, dass keine Reorganisation notwendig ist. Klar erkennbar ist zwar, dass über die Zeit eindeutig eine Verschlechterung der Werte stattfindet, aber die Werte bewegen sich im Rahmen eines Toleranzkorridors. Zur gleichen Entscheidung kommt man, wenn man das zweite Intervall isoliert betrachtet. Prüft man allerdings den Verlauf über den gesamten Zeitraum, so ist schon eine fast 20-prozentige Verschlechterung der Leistungskennzahlen erkennbar, was sicherlich in einer positiven Reorganisations-Entscheidung resultieren sollte. Festzuhalten bleibt also, dass eine Bewertung über den Zeitraum seit dem letzten

Reorganisationszeitpunkt notwendig ist, da eine Bewertung der letzten 5 Werktage allein nicht ausreichend für eine korrekte Bewertung sein kann.

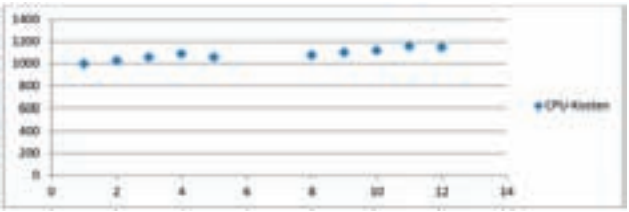


Abbildung 4: Schleichende Verschlechterung eines Beispielverlaufs

Jedoch auch die Untersuchung eines Tablespace über einen längeren Zeitraum stellt ein mathematisch-maschinelles Entscheidungsverfahren vor Probleme. In dem in Abbildung 5 dargestellten Beispiel ist eine Trendlinie eingezeichnet. Lässt man bei der Bewertung wieder einen Toleranzspielraum von knapp 10 Prozent zu (in dem keine Datenbank-Reorganisation notwendig ist), so kommt ein mathematisches Verfahren, das den gesamten Zeitraum berücksichtigt, zu der Entscheidung, dass eine Reorganisation nicht notwendig ist. Dabei ist für den Menschen die drastische Verschlechterung der letzten fünf Messpunkte klar erkennbar. Es muss also ein Algorithmus entwickelt werden, der mehrere Untersuchungszeiträume bewertet.

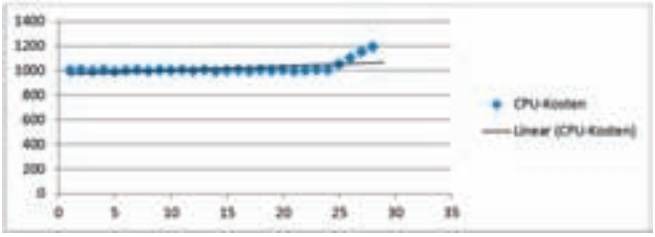


Abbildung 5: Beispiel des Problems bei einer Langzeituntersuchung

Wie im vorherigen Beispiel aufgezeigt, ist eine Untersuchung ab dem letzten Reorganisationspunkt bzw. seit Anbeginn der Messung sinnvoll. Um Schwankungen in der näheren Vergangenheit berücksichtigen zu können, sollen zusätzlich die letzten 5 Tage gesondert bewertet werden. Um ein Patt in den Reorganisationsentscheidungen zu vermeiden, ist die Gewichtung der Regeln sinnvoll, sodass hier Schwankungen aus der nahen Vergangenheit erkannt werden und maßgeblich in die Entscheidung mit eingehen.

Die Regressionsanalyse [BH06] hat sich als ein probates Mittel zur maschinellen Auswertung der historisierten Daten erwiesen, um daraus Prognosen für die weitere Entwicklung abzuleiten. Unser Prototyp hat die Möglichkeit, sowohl die logarithmische wie auch die lineare Variante anzuwenden. Das Bestimmtheitsmaß oder auch die Standardabweichung werden dazu verwendet, die gemessenen Daten daraufhin zu überprüfen, ob die errechnete Regressionsfunktion die Zusammenhänge gut wiedergibt.

Um für einen ersten Prototyp geeignete Schwellwerte für Filter- und Regressionsfunktionen definieren zu können, wurde die Software in einer Alpha-Version einer empirischen Untersuchung auf einem Testsystem unterzogen und die dabei gewonnenen

Ergebnisse wurden als Regeln im Prototyp implementiert. So müssen die folgenden Bedingungen erfüllt sein, um eine Aussage treffen zu können: es müssen mindestens 4 Messpunkte vorliegen; es müssen mindestens 70 Prozent der Messpunkte besetzt sein; jede SQL-Anfrage muss mindestens 100 Mal aufgerufen worden sein; die Standardabweichung darf bei maximal 250 liegen. Des Weiteren erfolgt eine Ausreißerfilterung. Die Schwellwerte wurden über die Zeit hinweg regelmäßig kontrolliert und an Veränderungen und neue Beobachtungen angepasst und optimiert.

Tabelle 6 zeigt zusätzlich, wie die Bewertungszeiträume der linearen und logarithmischen Regression und deren Gewichtung angesetzt wurden.

Name	Verfahren	Bewertungszeitraum	Gewichtung
Regel 1	Logarithmische Regression	Letzter Reorganisationszeitpunkt bis zum aktuellen Zeitpunkt	1
Regel 2	Logarithmische Regression	Letzte fünf Tage	1,3
Regel 3	Lineare Regression	Kompletter Zeitraum	0,5

Tabelle 6: Übersicht der implementierten Regeln

Bisher haben wir skizzenhaft aufgezeigt, wie eine Entscheidung auf SQL-Anfrage-Basis getroffen wird. Letztendlich muss jedoch anhand des Granulats Tablespace entschieden werden. Hierzu werden für jeden Tablespace alle gültigen SQL-Anfragen bewertet. Folgende Beispieltabelle (Tabelle 7) dient dazu, den Algorithmus näher zu erläutern. Zu dem Tablespace T1 wurden Messdaten über zwei SQL-Anfragen S1 und S2 gesammelt und durch die Regeln 1 bis 3 jeweils bewertet. Die Spalten „Prozentuale Entscheidung“ und „Absolute Entscheidung“ geben dabei jeweils das ungewichtete Ergebnis der Regeln an. Das gewichtete Ergebnis befindet sich in den letzten beiden Spalten „Ergebnis Prozentual“ und „Ergebnis Absolut“. Um die Masse an Daten zu reduzieren, wurden die Regel-Einzelentscheidungen auf eine SQL-Anfrage-Entscheidung reduziert, in die dann die bereinigten gewichteten Ergebnisse eingingen. Die beiden SQL-Anfragen S1 und S2 wurden unterschiedlich häufig in dem Beispiel-Untersuchungszeitraum aufgerufen. Während die SQL-Anfrage S1 5000 Mal aufgerufen wurde, wurde S2 lediglich 100 Mal aufgerufen. Wie erläutert, ist eine Gleichbehandlung der SQL-Anfragen nicht sinnvoll.

SQL-Anfrage	Anzahl Aufrufe	Regel	Regel-Gewicht	Prozentuale Leistungs-veränderung	Absolute Leistungs-veränderung	Ergebnis Prozentual	Ergebnis Absolut
S1	5000	Regel 1	1	10	27	10	27
		Regel 2	1,3	14	50	18,2	65
		Regel 3	0,5	8	21	4	10,5
		Gesamt				10,73	34,17
S2	100	Regel 1	1	-10	-2	-10	-2
		Regel 2	1,3	-5	-11	-6,5	-14,3
		Regel 3	0,5	-21	-6	-10,5	-3
		Gesamt				-9	-6,4
GESAMT						10,34	33,37

Tabelle 7: Beispiel zur Tablespace-Entscheidung

Der Unterschied ist leicht nachzuvollziehen. Die Ergebnisse der SQL-Anfragen S1 und S2 sind konträr. Anhand einer gewichteten Überführung der Einzelentscheidung in eine Gesamtentscheidung wird diese eindeutig und einsichtig. Wie beabsichtigt, wird die Gesamtbewertung in diesem Beispiel von der SQL-Anfrage S1 dominiert.

Variable	Beschreibung
$S_{1,T}..S_{N,T}$	SQL-Anfragen des Tablespaces T
$\|S_{K,T}\|$	Anzahl Aufrufe der SQL-Anfrage $S_{K,T}$ mit $1 < k < N$
$R_1..R_M$	Entscheidungsregeln
$F_{Abs}(r,s)$	Funktion zur Berechnung der Regelentscheidung „Ergebnis Absolut“. Das Ergebnis berücksichtigt bereits die Regel-Gewichtung.

Tabelle 8: Parameterübersicht

Das von uns implementierte mathematische Berechnungsverfahren sieht nun wie folgt aus, wobei die einzelnen Parameter in Tabelle 8 aufgelistet und näher erläutert sind. F_{GESAMT} berechnet dabei die Entscheidung für einen Tablespace T:

$$F_{GESAMT}(T) = \frac{\sum_{s=1}^{s=N}(\frac{1}{M}\sum_{r=1}^{r=M}(F_{Abs}(R_r, S_{s,T})) * |\, S_{s,T}\,|)}{\sum_{s=1}^{s=N}|S_{S,T}|}$$

4 Evaluation

Der Prototyp wurde bei der Firma SCHWENK Zement KG in Ulm eingesetzt. Die Firma SCHWENK Zement KG hat im Jahr 2009 mit 2.300 Mitarbeitern einen Umsatz von ca. 700 Millionen Euro erwirtschaftet. Bei der Firma SCHWENK Zement handelt es sich um einen typischen Kunden der Firma InfoDesign. Schwenk setzt mehrere SAP-Systeme weltweit ein, die auf einer DB2-Datenbank auf z/OS aufsetzen. Zur Automatisierung wichtiger administrativer Aufgaben wird der InfoMat eingesetzt.

Die Software wurde dabei testweise in dem Zeitraum von 6 Wochen eingesetzt. Dabei lag die Kernarbeitszeit zwischen 7 und 19 Uhr. Zu jeder vollen Stunde wurde jeweils ein Snapshot erstellt. Freitags beginnt der InfoMat mit der Einplanung der zu reorganisierenden Tablespaces, sodass sich die Bewertung auf die Tage von Montag bis Freitag konzentriert. Die Wochenendtage wurden in der Betrachtung vernachlässigt. Während des Untersuchungszeitraums wurden 39.729 verschiedene reduzierte SQL-Anfragen identifiziert. Bei 35.062 davon handelte es sich um SELECT-Anfragen.

Interessant für die spätere Auswertung ist vor allem, wie häufig und durchgängig diese SQL-Anfragen im DSC vorkommen. Nur noch 5.521 SQL-Anfragen erfüllten die beiden Kriterien, im Durchschnitt mindestens 100 Mal pro Werktag aufgerufen und mindestens an jedem zweiten Tag gemessen worden zu sein. Dabei konnten SQL-Anfragen zu 4.134 Tablespaces identifiziert werden. Insgesamt waren zum Analysezeitpunkt in dem Datenbanksystem 29.300 Tablespaces für Tabellen definiert, sodass damit eine Aussage über lediglich knapp 14 Prozent der Tablespaces getroffen werden konnte. Diese Zahl ist nicht negativ zu interpretieren: Prüft man die Anzahl aller Reorganisationen, die in dem Testzeitraum durchgeführt wurden, so stellt man fest, dass diese bei 2.760 liegt. Diese

2.760 Reorganisationen bestanden allerdings nur aus 1.152 unterschiedlichen Tablespaces. Da der InfoMat mehr Reorganisation als notwendig durchführt, liegt die Schlussfolgerung nahe, dass mit den 4.134 alle entscheidenden Tablespaces identifiziert werden konnten. Zum Stichtag der Auswertung wurden 527 Tablespaces durch den InfoMat zur Reorganisation eingeplant. Über 345 dieser Tablespaces konnte auch der Prototyp eine Aussage treffen. Tabelle 9 zeigt die Entscheidungen des InfoMat. Gegenübergestellt werden die Aussagen des Prototyps (Info-RM).

InfoMat - Entscheidung		Info-RM - Entscheidung			Einspar-potential	Einsparpotential inklusive TS mit unklarerer Aussage
Reorganisation aufgrund Bedingung		Reorganisation notwendig	Reorganisation nicht notwendig	Keine Aussage		
Nicht reorganisieren	3.607	436	417	2.754		
RB02	48	14	12	22	25%	71%
RB03,04,09,12,14	33	14	16	3	49 %	58 %
RB05	28	7	18	3	64%	75%
RB06	26	2	6	18	23%	92%
RB07	168	72	55	41	33%	57%
RB10	224	67	80	77	36%	70%
RB-Gesamt	527	176	169	182	32%	67%

Tabelle 9: Entscheidungstabelle nach dem ersten Messdurchgang

Es wurden zum Stichtag vom InfoMat aufgrund der Reorganisationsbedingung RB10 224 Tablespaces vorgeschlagen. Der Prototyp kommt nach Analyse der Messdaten lediglich auf 67 Tablespaces, die eine Verschlechterung der gemessenen Leistungsparameter aufzeigen. 80 Tablespaces würde der Prototyp definitiv von der Reorganisation ausschließen, da sich die Leistungskennzahlen verbessern oder zumindest konstant bleiben. Über 77 Tablespaces kann der Prototyp keine Aussage treffen, da bspw. nicht genügend Messpunkte während der Testphase gesammelt werden konnten. Die Auswertung aus dem ersten Messdurchgang basiert ausschließlich auf der Entwicklung der Regressionsfunktion. Dabei wurde geprüft, ob es sich um eine steigende oder fallende Regressionsfunktion handelt. Sinnvoll ist es, das Verfahren um einen Toleranzkorridor zu erweitern. Die Argumentation ist dabei, dass die Reorganisation ein aufwendiger Prozess ist, der sehr viel CPU- und I/O-Zeit in Anspruch nimmt, und dieser Mehraufwand in einem Verhältnis zu den zu erzielenden Einsparungen der Reorganisation stehen sollte. Minimale Verschlechterungen sollen dabei in einem gewissen Maße toleriert werden. Bei den Untersuchungen ist aufgefallen, dass die Performance sich bei sehr vielen Tablespaces nur minimal verschlechtert hat, aber eben nicht so drastisch, dass hierdurch eine Reorganisation begründbar ist. Daher wurde für einen zweiten Messdurchgang ein Schwellwert, ab dem ein Tablespace reorganisiert werden sollte, auf den CPU-Kostenwert 1 (in Sekunden) festgelegt. Dieser Wert soll dabei noch nicht als besonders sinnvoller Wert für einen Schwellwert für die Praxis verstanden werden, sondern dieser soll wirklich nur den Toleranzkorridor unter sehr konservativen Annahmen erweitern. Was sagt dieser CPU-Kostenwert aus? Betrachten wir die durchschnittlichen CPU-Kosten, die bei einer SQL-Anfrage anfallen, die exakt einen Datensatz zurückliefert. Dieser Wert

lag in unseren Messungen bei ca. 0,00395. Diese Anfrage – bei jeweiliger Planneugenerierung - könnte man ca. 253 mal ausführen, um 1 CPU-Kosteneinheit zu verbrauchen. Da die Plangenerierung sehr teuer ist und einen Großteil der CPU-Kosten bei einer SQL-Anfrage einnimmt (die bspw. nur wenige Datensätze zurückliefert) wurde geprüft, wie die durchschnittlichen CPU-Kosten bei SQL-Anfragen in der Testumgebung aussehen, die zwischen 10 und 20 Datensätze zurückliefern. Hier liegen die CPU-Kosten pro Datensatz bei ca. 0,001897. Hier könnte man die SQL-Anfrage – bei erneuter jeweiliger Planneugenerierung - immerhin 527 Mal ausführen. Der CPU-Kostenschwellwert von 1 ist also insgesamt als sehr niedrig gewählt zu werten. Unter Nutzung dieses Kostenschwellwerts ergaben sich folgende Ergebnisse eines zweiten Messdurchgangs (Tabelle 10).

InfoMat - Entscheidung		Info-RM - Entscheidung			Einspar-potential	Einsparpotential inklusive TS mit unklarerer Aussage
Reorganisation aufgrund Bedingung		Reorganisation notwendig	Reorganisation nicht notwendig	Keine Aussage		
Nicht reorganisieren	3.607	95	758	2.754		
RB02	48	7	19	22	39%	85%
RB03,04,09,12,14	33	5	25	3	76 %	85 %
RB05	28	2	23	3	82%	93%
RB06	26	0	8	18	31%	100%
RB07	168	21	106	41	63%	88%
RB10	224	20	127	77	57%	91%
RB-Gesamt	527	55	308	164	58%	89%

Tabelle 10: Entscheidungstabelle nach zweitem Messdurchgang

Die Ergebnisse des zweiten Messdurchgangs sprechen für sich. Man kommt insgesamt auf ein Einsparpotential von 58 oder sogar 89 Prozent gegenüber den vom InfoMat vorgeschlagenen Tablespaces. Die Werte erscheinen sehr positiv. Dabei darf man tatsächlich nicht vergessen, dass nicht alle Reorganisationen aus Performancegründen angestoßen werden. Bereits im zweiten Kapitel wurde darauf hingewiesen, dass Datenbank-Reorganisationen auch aus Gründen der Speicherplatz-Ausnutzung bzw. –Rückgewinnung vorgenommen werden. Tatsächlich beziehen sich auch einige der oben aufgeführten Reorganisationsbedingungen auf diese Fälle. Insbesondere aber die Reorganisationsbedingungen RB07 und RB10, die zusammen immerhin 74 Prozent der angestoßenen Reorganisationen ausmachen, sind von der Motivation durch eine Aussicht auf eine Performanceverbesserung begründet.

5 Fazit

Unter geeigneter Auswertung der CPU-Zeiten mittels Regressionsanalyse konnte festgestellt werden, dass zwischen 58 bis 89 Prozent der Reorganisationen, die durch statische Verfahren vorgeschlagen werden, aus Performance-Sicht unnötigerweise durchgeführt werden. Die Überprüfung durch den in diesem Artikel angeregten leistungsorientierten Ansatz der Auswahl der Reorganisationskandidaten bestätigt somit die Vermutung, die

sich in den IT-Abteilungen manifestiert hat. Ein weiteres Ergebnis der vorgenommenen Analyse ist, dass auch Tablespaces, die aus leistungsorientierter Sicht dringend einer Reorganisation bedürfen, nicht notwendigerweise von statischen Verfahren erkannt werden, von einem leistungsorientierten Verfahren jedoch durchaus. Somit konnte empirisch gezeigt werden, dass ein (einfach gestaltetes) leistungsorientiertes Verfahren insgesamt eine weitaus höhere Genauigkeit erzielen kann. Interessant ist in diesem Zusammenhang die Beobachtung, dass die potentiellen Kunden einer solchen Software vorrangig an einer Reorganisationsvermeidung interessiert sind.

Die Firma SCHWENK setzt die Software mittlerweile im produktiven Einsatz ein. Es hat sich dabei erwiesen, dass der in Kapitel 4 beschriebene, durch Vermeidung von unnötigen Reorganisationen erzielte Gewinn dauerhaft erzielt werden kann. Des Weiteren wurden erste Proof of Concepts bei anderen Testkunden im Bereich von OLAP-Systemen durchgeführt. Die Herausforderung lag hier insbesondere in einer Verbesserungen des SQL-Parsers, da die SQL-Anfragen bei einem OLAP-Workload komplexer und somit schwieriger zu parsen sind. Auch hier wurden vergleichbar positive Werte erreicht.

Da die vorgestellte Problematik nicht nur im z/OS-Umfeld existiert, geschieht gegenwärtig eine Portierung der Lösung auf andere Plattformen (wie Oracle und DB2 LUW).

Literatur

[BH06] Backhaus, K.: *Multivariate Analysemethoden eine anwendungsorientierte Einführung*, SpringerLink (Online service), 2006

[CM01] Mullins, C.: The DBA Corner. Database Fragmentation and Disorganization. http://www.craigsmullins.com/dbta_004.htm, Dezember 2001 [Stand: 19.10.2011]

[SG79] Sockut, G.;Goldberg, R.: Database Reorganization – Principles and Practice, in Computing Surveys, Vol. 11, No. 4, Dezember 1979

[SH94] Shallahamer, C.: *Avoiding a Database Reorganization – Understanding, detecting, and eliminating harmful database fragmentation,*
URL: http://www.allenhayden.com/cgi/getdoc.pl?file=reorg.pdf. November 1994 [Stand: 19.10.2011]

[WV09] Weaver, R.: *Database Reorganization Strategies for DB2 z/OS*
URL:http://www.mainframezone.com/it-management/database-reorganization-strategies-for-db2-z-os. Juli 2009 [Stand: 19.10.2011]

[QE10] Ahrends, J.: *Oracle Datenbank Reorganisation*
URL:
http://www.toadworld.com/Portals/0/JohannesA/artikel/Reorganisation%20Teil%201.pdf. Februar 2010 [Stand: 03.01.2013]

Making Social Media Analysis More Efficient Through Taxonomy Supported Concept Suggestion

Fabio Cardoso Coutinho, Alexander Lang, Bernhard Mitschang

IBM Deutschland Research & Development
Schoenaicher Str. 220
71032 Boeblingen
fabio.coutinho@gmail.com
alexlang@de.ibm.com
bernhard.mitschang@ipvs.uni-stuttgart.de

Abstract: Social Media sites provide consumers the ability to publicly create and shape the opinion about products, services and brands. Hence, timely understanding of content created in social media has become a priority for marketing departments, leading to the appearance of social media analysis applications. This article describes an approach to help users of IBM Cognos Consumer Insight, IBM's social media analysis offering, define and refine the analysis space more efficiently. It defines a *Concept Suggestion Component (CSC)* that suggests relevant as well as off-topic concepts within social media, and tying these concepts to taxonomies typically found in marketing around brands, products and campaigns. The CSC employs data mining techniques such as term extraction and clustering, and combines them with a sampling approach to ensure rapid and high-quality feedback. Initial evaluations presented in this article show that these goals can be accomplished for real-life data sets, simplifying the definition of the analysis space for a more comprehensive and focused analysis.

1 Introduction

According to the study in [Cor09], 500 billion impressions about products and services are annually shared among clients in social networks, while 78% of consumers trust peer recommendations. These statistics show that news of great products and services, along with their experiences, have the potential to rapidly define a product's success or failure. Therefore, the fast understanding of user experience via users' direct feedback or customer conversations about their products and services on social network web sites has become crucial to effective marketing.

Aiming to aid marketing teams in exploring social networks as marketing channels, *social media analysis* has drawn a great deal of attention recently. It can be defined as the process of measuring, analyzing, and interpreting the results of interactions and associations among people, topics and ideas through data mining techniques. IBM Cognos Consumer Insight (CCI) [IBM11] is a social media analysis application that helps companies to gain insight into consumer opinions and spot trends related to products and brands. The chal-

lenge in setting up a relevant social media *analysis space* is two-fold:

1. Homonyms, in which a monitored term has multiple meanings, causing the retrieval of unwanted posts for the analysis.
2. Missing terms, in which important terms that could be used to further refine the analysis are not present in the analysis space.

This article describes an approach to tackle above challenges through the suggestion of relevant as well as off-topic concepts to the user setting up the analysis space. This suggestion is based on term extraction from a subset of social media documents, combined with a relevancy classification of these terms according to a user-provided taxonomy.

The “on-topic" vs. “off-topic" classification of terms helps social media analysts to rapidly uncover homonyms, spot missing concepts, and rapidly refine the analysis space. The result is a significantly improved user experience and a decrease in “time to value", i.e., the time needed until end users can uncover results from social media.

To ensure both rapid response times as well as high quality of the concept suggestions, the *Concept Suggestion Component* described in this article combines several techniques such as term ranking, sampling and adaptions to well-known k-means clustering in a new way. The initial results shown in this paper seem to validate the effectiveness of our approach.

2 The Social Media Analysis Process in IBM Cognos Consumer Insight

IBM Cognos Consumer Insight (CCI) is a social media analytics application that enables marketing professionals to

- Measure the effectiveness of marketing campaigns by analyzing the volume, influence and sentiment of consumer responses across multiple social media channels.
- Anticipate consumer reaction to new products and services by examining affinity relationships and evolving topics.
- Pinpoint key influencers and social channels to engage consumer communities through both traditional and digital marketing activities.

CCI accesses a wide range of social media content, including twitter, facebook, blogs, message boards, video comments and product reviews, and provides analysis capabilities over a wide range of languages, including (but not limited to) English, German, French, Spanish and Chinese.

Figure 1 portrays one of the many available analysis in CCI based on the monitored social media content - in this case, allowing the differentiation between the content of posts from authors who own a certain product vs. those from authors who just speculate about

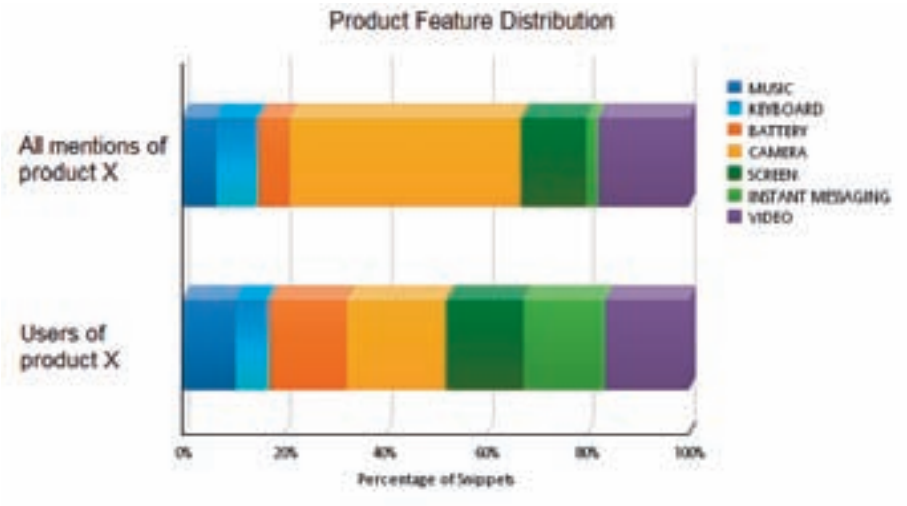


Figure 1: Using CCI to understand the importance of certain features for a particular product

it. These sort of analysis found on IBM Cognos Consumer Insight are typically used by employees in different roles within an organization:

- *Social Media Analysts* define the *analysis space* provided by CCI. They develop strategies for engagement in social media, and regularly feed back insights gained from social media monitoring into the marketing organization, to help them evolve their strategies in a timely fashion.
- *Social Media Consumers*, such as brand managers, product managers or call center staff, explore the analysis space through pre-configured dashboards in their daily business as one information channel. Another important recipient of this information are Chief Marketing Officers or the Heads of Digital Marketing.

The Social Media Analyst uses the *CCI Administration Portal* to define the analysis space in two steps:

1. A set of *content queries* that define the "raw" social media content to be pulled from several social media sources. Content queries are defined like "typical" web queries, using a keyword search syntax with operators like *and, or, not* or *proximity*.
2. A *taxonomy* of the relevant concepts that should be monitored. Top-level entries of this taxonomy are typically brands, companies grouped by their market, product lines and campaigns. Within each top-level entry, the analyst defines *concepts* such as brand and company names, products, product features, spokespeople for a certain brand (often celebrities such as sports stars), as well as marketing campaign messages. The analyst provides one or more terms or regular expressions that define

each concept, which are used by CCI's analysis platform to extract the information from social media content.

The definition is typically done iteratively:

1. The Social Media Analyst starts with a small taxonomy and initial content queries
2. The analyst runs the extraction process
3. The analyst reviews the results, refines the definitions further or adds new definitions
4. Unless the configuration is sufficient for the use case, the next iteraton starts at Step 2.

Once the definitions are both stable and comprehensive enough, the Social Media Consumers navigate within this taxonomy in the *CCI Dashboards* to answer their business questions. They only see the concepts defined in the taxonomy, and don't need to know the content queries or the taxonomy definitions being used. CCI ensures that the content queries are regularly re-executed and the taxonomy rules re-applied to make sure new information is available to the Social Media Consumers.

This approach is depicted in Figure 2, which shows the two steps described previously and the underlying components used in producing the information to be consumed by CCI users.

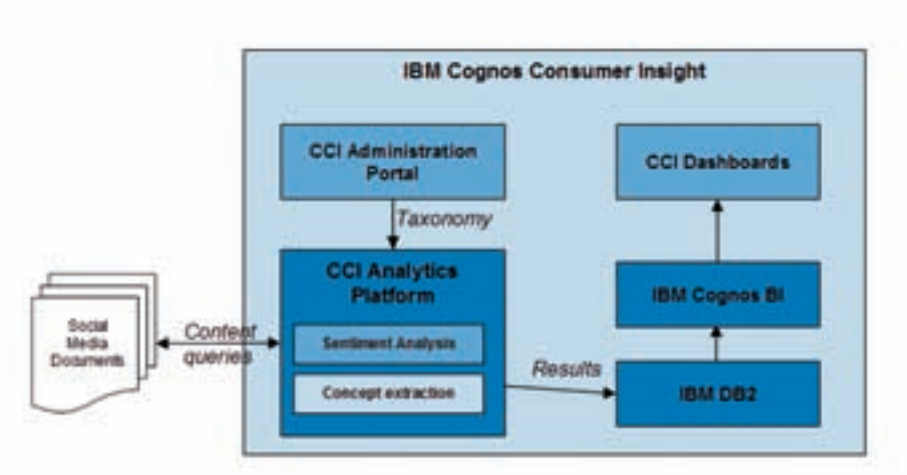


Figure 2: IBM Cognos Consumer Insight Architecture Overview

The two-step approach in defining the analysis space allows analysts to be very broad in their set of content queries (e.g., *"everything that mentions company X"*), and use the power of regular expressions within the taxonomy for very fine-grained analysis (e.g.,

identifying and extracting authors who own a certain product vs. authors who just speculate about it, or have an intention to buy, as seen in Figure 1).

Still, the configuration of social media analysis presents challenges on two fronts:

- **Ambiguous brand, product or feature names** In many cases, product names also have a meaning outside their product domain. This is typically caused by product names that are used by different companies (for example, both adidas as well as Ferrari have a product called *F50*), or product names that are a "real world" concept, for example, HTC's smartphone called *Desire*. One way to remove the ambiguity is to use content queries that contain the brand name, e.g., *adidas F50*. However, this approach can miss a lot of social media content, where users may use other terms from the domain to "implicitly" disambiguate the term - for example, a tweet like *Just played soccer with my new F50 - great!*. Hence, it's typically more accurate to disambiguate content queries through negation, such as *F50 -Ferrari*. In both cases, analysts are required to manually go through the results of the content queries to pick the right approach, as well as the right terms to further improve the content queries - a potentially time-intensive process.
- **Missing concepts in the analysis space** Not all concepts can be anticipated in advance - especially when monitoring campaigns or new developments in social media. Hence, Cognos Consumer Insight provides feedback on dynamically evolving topics through clustering of social media content. Still, enriching the taxonomy with relevant concepts as part of the definition and refinement of the analysis space makes it easier to integrate the analysis results in business processes or alerts. The challenge for the analyst is to identify the concepts that are present in the "raw" documents, but are missing from the taxonomy and its definitions. A system that automatically suggests relevant concepts can greatly cut down the time required to arrive at a comprehensive taxonomy.

The goal of the Concept Suggestion Component is to support the analyst at step 3 of the analysis space definition process by:

- Suggesting disambiguations of ambiguous content queries
- Suggesting new concepts, which will be reflected as additional content queries as well as enhancements to the taxonomy

3 The Concept Suggestion Component

Existing approaches for automatic term extraction, an obvious solution for the problem at hand, suffer from two main drawbacks. Simple and fast approaches, such as word counts shown within a word cloud, often provide too much "raw" information to be useful. Moreover, they do not provide a clear way to classify the terms according to their relevance to the user query. On the other hand, more sophisticated approaches, such as document

clustering, provide more concise topic keywords, but solve the classification problem only partially: these approaches do not allow the relation to a user-defined taxonomy, which leaves the distinction between relevant and off-topic concepts to the user. On top of that, the “time-to-feedback” for these latter approaches is very high. Considering that the configuration process typically takes several iterations, and that the alternative for automated feedback is manually sifting through results, users do accept feedback that takes longer to create than the typical time span acceptable for “interactive" user interfaces (8 seconds), but shouldn’t “block” the user for several minutes in each iteration either.

This work defines a middle-out approach to this problem that is fast enough to be acceptable, but uses techniques beyond simple counting to improve its usefulness and allow term classification. The resulting *Concept Suggestion Component (CSC)* combines good time-to-feedback with result quality, based on the following steps:

1. Downsampling of the document corpus to reduce the execution time of the analysis.
2. Extraction of the most important terms from the downsampled documents.
3. Clustering of the downsampled documents based solely on the extracted terms.
4. Suggest the Relevant and Off-Topic Concepts based on the classification of clusters into Relevant or Off-Topic according to user taxonomy. This classification is based on the distance of the clusters to a control cluster formed by the terms in the taxonomy.

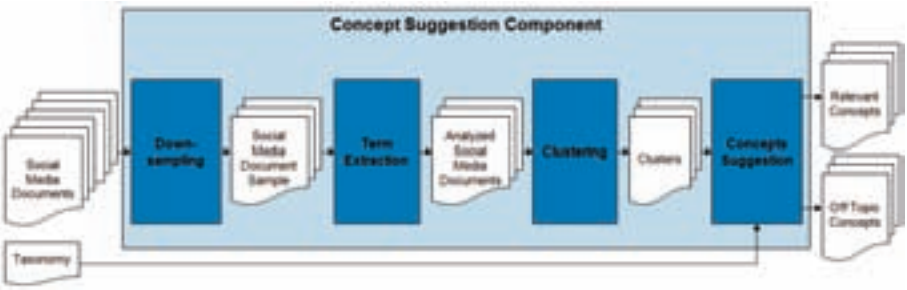


Figure 3: CSC Architecture

As seen in Figure 3, CSC is comprised of four main modules that execute sequentially to provide the desired output. Each of these modules are further detailed in the next sections.

3.1 Downsampling Module

When executed upon a very large set of documents, a clustering algorithm can take a prohibitive amount of time to finish, being then not responsive enough for user interaction.

To mitigate this problem, the first module to execute within CSC is responsible for the reduction of the document corpus via document sampling.

Sampling is a well known statistical method to analyze big populations. As defined in [JKLV99], Simple Random Sampling is commonly used in document analysis and information retrieval. This approach makes use of the fact that, if the whole document corpus is analyzed, its term structure is not embodied by any single document. Thus, if a random sample of a big enough size is taken, the characteristics of the entire corpus can be estimated from this smaller set of documents. Examples of this approach applied to information retrieval can be seen in [CCD99, Bro97, CKPT92, RGA04].

Assuming a normal distribution, the size of a sample can be determined for a given margin of error and confidence level. Figure 4 shows a statistical analysis of the inherent similarity of a document corpus that will be used for one of the case studies presented further in this article.

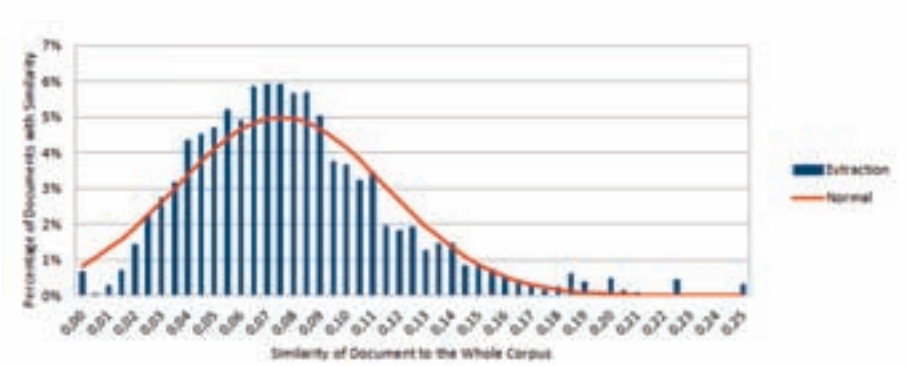


Figure 4: Document Corpus Statistical Analysis

The image shows a histogram of the similarities between the most important terms of each document when compared to the most important terms extracted from the whole of the document corpus. As it is made clear by the picture, the distribution of the similarities of the document corpus fairly resembles a normal curve. Thus, we can use the following sampling method:

For a margin of error *ME* and confidence level α, the number of samples *n* needed to estimate its mean value is determined by:

$$n = \frac{N * 0,25z^2}{(N-1) * ME^2 + (0,25z^2)} \tag{1}$$

where z is the critical standard score for $1 - \frac{\alpha}{2}$, N is the total number of documents and the margin of error is the percentual error allowed for the mean of the similarities of the document corpus when analyzing the sample instead of the whole amount of documents.

Figure 5 depicts the behavior of the chosen downsampling function with a 95% of confidence level for a 1% margin of error .

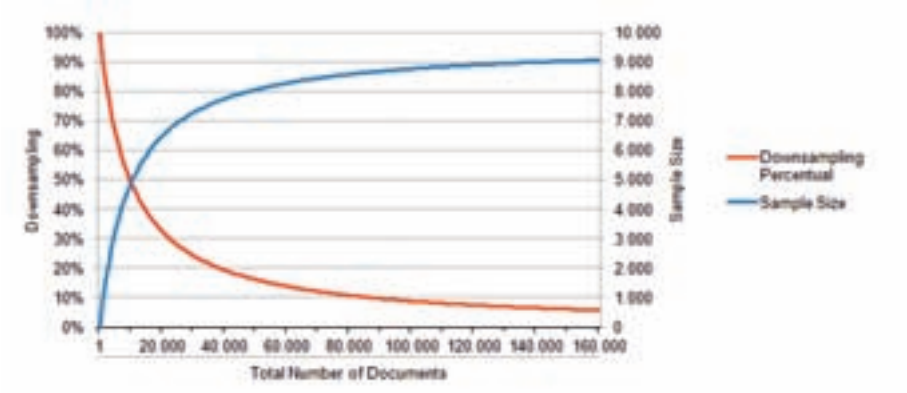


Figure 5: Downsampling Function Behavior

It is clear from the picture that the chosen sampling technique assures that only a reasonable number of documents are processed, even when a very large document corpus is analyzed - the maximum number of processed documents is smaller than 10.000 documents for an analysis with 95% confidence level with a 1% margin of error. Therefore, the use of the proposed downsampling technique allows a substantial improvement on the overall performance of the algorithm.

3.2 Term Extraction Module

In the Term Extraction Module, each document is separately analyzed and has a group of terms extracted from its contents. These extracted terms aim to provide a complete enough description of each document, so that the clustering can be performed against these words instead of the whole text.

The term extraction algorithm implemented in this work is the TextRank algorithm presented in [MT04] - an unsupervised extraction technique that is totally context independent and, therefore, each document can be analyzed independently without needing an analysis of the entire document corpus to extract its terms. The chosen algorithm has also a better performance against a big set of documents when compared to other state-of-the-art techniques.

As presented in [MT04], the generic graph-based ranking algorithm for texts consists of the following steps:

1. Identify text units that best define the task at hand, and add them as vertices in the graph.
2. Identify relations that connect such text units, and use these relations to draw edges between vertices in the graph.

3. Iterate the graph-based ranking algorithm until convergence.

4. Sort vertices based on their final score. Use the values attached to each vertex for ranking/selection decisions.

By default, the algorithm uses all the nouns found in the document analyzed as vertices of the graph. The selection of nouns provides a simple filter that typically captures brands, products and features. However, users can also select adjectives or simple noun phrases as vertices. The latter are sequences of nouns, excluding determiners or modifiers. In this case, compound terms are also treated as noun sequences. For example, *Akkulaufzeit* (German for *battery life*) is split into the two nouns *Akku* and *Laufzeit*. This split removes phenomena such as so-called *Fugenmorpheme* in German to ensure that the split yields valid nouns. For example, *Dämpfungssohle* (cushion sole) yields *Dämpfung* + *Sohle*.

As stated in the algorithm, the next step is identifying the relations that connect each of the vertices, and use them to draw the edges of the graph. TextRank uses the co-occurrence of terms to define the vertices connections - the distance between the occurrence of two terms determines if these vertices are linked or not. Following the configuration that has shown the best results in [MT04], in this implementation, the window of co-occurrence that determines a link between two vertices is of two words and the links are weighted by the amount of times that the vertices co-occur. Therefore, if a pair of words co-occur more often, the importance of this recommendation is taken into consideration in the algorithm.

Having the graph constructed, the value of each vertex can be calculated. As proposed by [MT04], the equation that determines the score of a vertex can be determined by:

$$Rank(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{V_k \in Out(V_j)} w_{jk}} Rank(V_j) \quad (2)$$

where $In(V_i)$ are the vertices that point to the vertex V_i and $Out(V_j)$ the vertices that V_j points out to. The w_{ji} is the weight of the of the link from vertex j to vertex i and the factor d is a constant set to 0,85 as suggested by [MT04].

Using this recursive formula, the score of each vertex in the graph is recalculated until convergence, and then the vertices are ranked respectively in terms of their importance to the document. Since the social media documents analyzed by CCI typically show a fairly concise nature, the use of a number greater than 10 terms to represent them is found to impact the rest of the component's performance as well as leaving their summarization excessively verbose. Therefore, in this solution, a maximum of 10 terms is used to represent each of the documents being analyzed.

Having finished the selection of the top ranked terms, the document can then be viewed as a vector represented in a maximum of 10 dimensions space as proposed by [SWY75]. The author proposes the Vector Space Model, in which a document is represented by a vector $[w_{t0}, w_{t1}, ...w_{tn}]$, where $[t_0, t_1, ...t_n]$ is a set of words and w_{ti} expresses the importance of t_i to the document. Therefore, from here on, whenever a document *D* is mentioned, it should not be interpreted as a text, but as a term vector as defined below:

$$D = [w_{t0}, w_{t1}, ...w_{t9}] \tag{3}$$

3.3 Clustering Module

Having each document mapped in a group of terms as an input from Term Extraction Module, these descriptive terms are used to separate the document corpus into a group of clusters in the Clustering Module. The clustering algorithm performed in the proposed component is a K-Means algorithm with some improvements based on the algorithm presented in [CKPT92].

As defined in [MRS08], a known issue in K-Means clustering is determining the number of clusters. However, as is the case in the study presented in [Wei06], the real output of the component is not influenced by the decision of cluster cardinality. The clusters in this solution are intermediate structures that allow the comparison of documents to the taxonomy, and therefore, the number of clusters does not influence critically on the final output of the algorithm.

Having the greatest setback of K-Means mitigated, this algorithm can be adapted to the reality of social media document clustering using the scatter gather algorithm proposed by [CKPT92] as a guide for the design of the solution to the problem at hand.

As defined in [JMF99], the typical K-Means clustering consists of the following steps:

1. Choose *k* cluster centers to coincide with *k* randomly-chosen patterns or *k* randomly defined points inside the hypervolume containing the pattern set.
2. Assign each pattern to the closest cluster center.
3. Recompute the cluster centers using the current cluster memberships.
4. If a convergence criterion is not met, go to step 2.

This section has been organized according to the previous mentioned steps. First, the selection of initial cluster centers is discussed, followed by the description of how each document is assigned to each cluster. Then methods to refine the clusters are given and, finally, the convergence of the algorithm is measured.

3.3.1 Selection of Initial Cluster Centers

Several examples on how to enhance initial cluster selection are found in document clustering literature, such as in [CKPT92, Wei06]. The main motivation behind it is due to the fact that, if one outlier document is chosen as an initial seed, no other vector is assigned to it during subsequent iterations, leading to an isolated cluster.

Seeing that, during the cluster refinement step, a noise reduction optimization is performed that avoids the clustering of outliers, these implementations are not adopted as part of the algorithm.

Therefore, the selection of initial cluster centers is made by randomly choosing *k* documents from the entire corpus to be the *k* initial centers. Hence the cluster is also represented by a vector with a maximum of 10 dimensions. This representation is used to determine the similarity between the documents and the clusters, as defined next.

3.3.2 Assigning Documents to Clusters

With the initial centers chosen, the algorithm proceeds then to assign the documents to each cluster. As determined by the typical K-Means algorithm, each document should be associated to the cluster with the closest center. However, to find the closest cluster, a distance function must be determined first.

According to [TC92], the most common distance metric used in the vector space model is the cosine similarity between the document vectors, as seen in [CKPT92, Wei06]. The cosine measure is a robust technique that measures whether two vectors are pointing towards the same direction in the space determined by the term-document matrix formed by both document's vectors. The cosine is used as an approximation of the angle between both vectors, and is used in place of the angle, since it is easier to compute. It is then defined that the similarity *Sim(x,y)* is determined by the cosine of the angle between the vectors *x* and *y*.

With the similarity formula defined, it is easy to determine the distance function between documents. [Bro97] determines that the similarity *Sim(x, y)* of two documents, can be measured by $Dist(x, y) = 1 - Sim(x, y)$. Hence, the proposed clustering algorithm uses, as distance function, the following formula:

$$Dist(x, y) = 1 - Sim(x, y) = 1 - \cos(x, y) = 1 - \frac{x \cdot y}{\parallel x \parallel \parallel y \parallel} \tag{4}$$

where $x \cdot y$ is the dot product between vectors *x* and *y*, $\parallel x \parallel$ is the norm of vector *x* and $\parallel y \parallel$ is the norm of vector *y*.

Having defined the distance property, the definition of the cluster assignment algorithm is very straightforward. Each document in the corpus has its distance to every one of the *k* cluster centers computed. The document is then assigned to the nearest one.

Since both the clusters and the documents are represented by a maximum of 10 dimensions, there is a chance that a given document is orthogonal to every cluster. Such documents are then entirely dissimilar to the entire cluster group. When this situation occurs, these documents are assigned to a special cluster - the Noise Cluster - which groups all of these dissimilar documents.

3.3.3 Cluster Refinement

With all the documents assigned to their respective clusters, the algorithm can then use the clusters' own information to refine the quality of the result. Some refinement algorithms proposed by [CKPT92] are adapted to the problem here at hand.

Cluster Center Updating [CKPT92] proposes that the most important words from the cluster's documents are a short description of the contents of the cluster, serving thus the purpose of its center. The formula that determines the term importance in a given cluster is defined as:

$$Importance(t, D) = \sum_{d_i \in D} \begin{cases} Rank(t) & \text{if } t \in d_i \\ 0 & \text{if } t \notin d_i \end{cases} \quad (5)$$

where *D* is the set of documents in the cluster and *Rank(t)* is the ranking of the term *t* in the document *d*.

Having calculated the importance of all cluster's terms, the proposed algorithm substitutes the old center for the new 10 word vector comprised of the most important words. The number of terms to describe the cluster's new center is kept to 10, so that the the cluster's description remains fairly concise and that the algorithm maintains its performance in the new iterations.

Cluster Joining In [CKPT92], the authors propose merging document clusters that are not usefully distinguished by their cluster centers. In this implementation, the clusters are compared against one another, and, if there is an overlapping of more than 50% of the cluster centers' terms, they are merged. The newly created cluster inherits all documents from both clusters and has its center updated by the same algorithm presented before.

Even though this optimization reduces the number of initial clusters, the total number of clusters is kept due to additions taken place in the Noise Reduction section of the algorithm.

Noise Reduction As explained before, the last refinement in the clustering algorithm aims to reduce the amount of documents not assigned to any cluster. In order to assure that a relevant cluster is created from the unassigned documents, some computation is necessary before creating the new clusters. Therefore, a noise reduction algorithm is proposed, as summarized in the following steps:

1. Rank the Noise Cluster's terms according to their importance.
2. Randomly choose from the Noise Cluster one document that contains the most important term and create a new cluster with it as the center.
3. Repeat step 2 until the number of clusters has once again reached *k* clusters.

4. Compare the number of documents of the smallest cluster with the number of times that the most important term on the Noise Cluster appears on the Noise Cluster's documents.

5. If this term's frequency is greater than the smallest cluster size, randomly choose from the Noise Cluster one document that contains this most important term and create a new cluster with it as the center substituting the smallest cluster.

6. Repeat steps 4 and 5 until the smallest cluster size is bigger than the frequency of the most important term remaining in the Noise Cluster.

This algorithm reduces dramatically the Noise Cluster size and, what is more important, makes sure that the remaining documents have actually no similarity with any other documents. On top of that, the use of documents containing highly important terms, makes sure that any bad initial cluster setup has a great chance of being overcome with the creation of noise-based clusters.

3.3.4 Convergence Check

As stated in [DBE99], the Davies Bouldin index takes into account both cluster dispersion and the distance between cluster means. Therefore, well separated compact clusters are preferred when converging according to this index. According to [DB79], the index "has the significance of being the system-wide average of the similarity measures of each cluster with its most similar cluster. The best choice of clusters, then, is that which minimizes this average similarity."

The proposed clustering algorithm aims then to minimize Davies-Bouldin index while it iterates through the Cluster Assignment and Clusters Refinement steps. Using such metric, the algorithm will converge to a better overall clustering result, assuring that the clusters are compact and far from each other.

3.4 Concepts Suggestion Module

The next step on the CSC information flow is the ranking of each cluster according to its relevance to the user provided taxonomy. As stated before, CSC expects a group of terms that describes the interest of the user when performing a query. After the execution of this analysis, every cluster has a ranking of how relevant it actually is to the user.

The next step in the algorithm is then determining the distance of each cluster to the taxonomy's terms. Therefore, the taxonomy is translated into a Control Cluster defined by these user-defined terms. All terms in the Control Cluster are ranked with a standard ranking of 1. Having defined this cluster as a bias, the algorithm then measures the distance of the k clusters to this control cluster. According to their results, the clusters are then classified in three categories, as presented below:

1. If $Dist(C_i, ControlCluster) = 1$, then the clusters have no terms in common. Therefore, C_i is not relevant.
2. If $\mu + \sigma < Dist(C_i, ControlCluster) < 1$, then the cluster is an outlier and, thus, considered ambiguous.
3. If $Dist(C_i, ControlCluster) \leq \mu + \sigma$, then $Cluster_i$ is considered relevant.

In these inequations, the μ stands for the mean of all non orthogonal clusters' distance to the Control Cluster, whereas σ stands for their standard deviation.

With the classified clusters as an input, the concept suggestion algorithm aims to extract, from each of these cluster groups, words that serve as a good identifier to their content. Loosely based on the *tf*idf* metric for term extraction that favors terms that distinguish certain individual documents from the remainder of the collection, the importance of a term in a cluster group is given by defining *tf* as the term's importance to the group, and *idf* as a term that varies inversely with its importance to the remainder of the groups. Thus, the term-weighting formula for a given group is defined as:

$$w_t(t, d, D) = Importance(t, d) * (\frac{Frequency(t, d)}{Frequency(t, D - d)})^n \quad (6)$$

In the above definition, *d* stands for the documents of a group, *D* the entire document corpus and *n* determines the relevance of the exclusivity factor on the term weighing.

This weighing procedure allows the ranking of the terms and present to the user the concepts that best define each one of the categorized groups. Thus, as an output of this module and of the component itself, two lists of concepts are provided:

- List of Relevant Concepts, extracted from Relevant Clusters.
- List of Off-Topic Concepts, extracted from Off-Topic Clusters.

4 Implementation and Evaluation

4.1 CCI-Integration

As stated before, the main goal of the proposed solution is to enhance CCI's usability, making it an easier task to the user. An overview of the component's integration with CCI is presented in Figure 6.

Figure 6 shows CSC as an add-on to CCI's architecture. This component receives two inputs - the Taxonomy and the Social Media Documents - and, as output, provides a list of concepts to the user via the Administration Portal. These concepts are already classified as relevant or off-topic, so that the user can then, in a timely manner, refine his set-up. The refinement can be either made by introducing into the set-up relevant concepts previously left out, or by excluding off-topic terms not relevant to his interests.

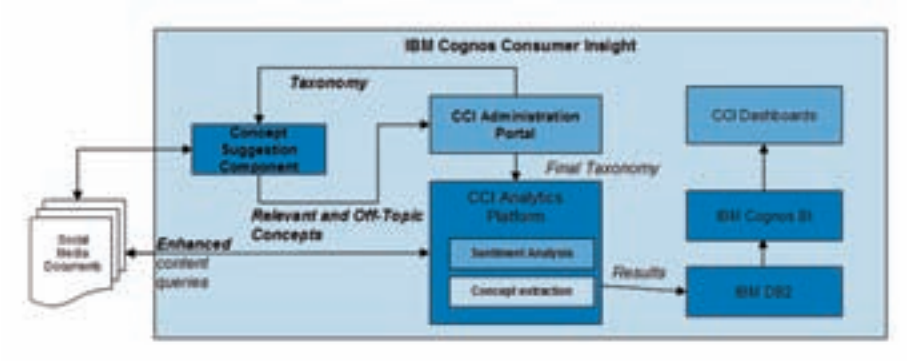


Figure 6: Solution Architecture

As seen in Figure 6, the software receives the Taxonomy directly from CCI. These are the terms that are used to determine whether the suggested concepts are relevant or not to the user query. The second input the component receives are the Social Media Documents. These are the results from the user's query in CCI's Administration Portal. These documents are analyzed by the algorithm proposed in the previous section and the output is extracted.

This output is then, as mentioned beforehand, provided to CCI's Administration Portal. If CSC manages to suggest these concepts rapidly enough, the end user is able to refine the set-up parameters in a much faster manner, thus improving the overall usability of CCI's analysis set-up.

This service provided by the component is made available to CCI as a REST web-service with JSON message format. The chosen architecture provides a non intrusive development, that keeps the current flow of user operations.

4.2 Evaluation

To perform the component's evaluation two case studies are proposed in which the ambiguity of monitored concepts are known to be a good example for the use of CSC. After analyzing the results of the case studies, CSC's performance is briefly analyzed.

4.2.1 Case Study - Adidas F50 Football Boots

The first case study is the monitoring of the *f50* concept, which is the name of a known brand of Adidas' football boots, as well as a known sports car brand produced by Ferrari. The social media content used in this case study is drawn from blogs, discussion forums, microblogs (e.g., Twitter), video comments and product reviews. It focuses on English content posted between January 1st and March 1st, 2012. The taxonomy used by the

Concept Suggestion Component contains two top-level entries:

1. *Shoe Brands*, which includes a single concept, *adidas*
2. *Football Shoes*, which includes a single concept, *F50*, as a product name.

Figure 7 presents the Relevant Concepts for the performed query. As expected, all terms suggested are somehow football related and can be used to further refine the set-up of CCI.

Figure 7: Relevant Concepts for *f50*

Figure 8 presents the Off-Topic Concepts for the performed query. The most important off-topic concept is *ferrari*. Other important terms retrieved by the algorithm - such as *currency* and *vs* - appear due to posts related to Fujitsu's F50 currency dispensers.

As the taxonomy is very simple, some terms that were identified as off-topic could be relevant to the user - such as *cristiano* and *ronaldo*, the name of a football player sponsored by Nike. In this case, the user is free to enhance the taxonomy with an additional top-level entry *Brand Spokespeople*, and add *Christiano Ronaldo*.

Figure 8: Off-Topic Concepts for *f50*

In order to improve the initial definitions, the user can select *ferrari* and *camcorder* from the off-topic terms, and add them as so-called *exclude terms* to the *F50* concept. The CSC will automatically update the CCI content query from *F50* to *f50 -ferrari -fujitsu*. The user can also create new concepts from on-topic terms such as *micoach*, or enhance the definition of *F50* with the product name variant *adizero*.

4.3 Case Study - Nike and Adidas Football Boots

In order to illustrate the behavior of CSC when analyzing more than one concept in a query, the previous example is extended to monitor not only the *f50* boot, but also Nike's *Mercurial* football shoe. The query is changed to monitor both terms. The taxonomy is enhanced with the following:

1. The concept *Nike* is added to *Shoe Brands*
2. The concept *Mercurial* is aded to *Football Shoes*

and, in the taxonomy, the term *nike* is added as a brand. This query's execution provides the output presented in figures 9 and 10.

Figure 9: Relevant Concepts for *f50* and *mercurial*

Figure 10: Off-Topic Concepts for *f50* and *mercurial*

As expected, the Relevant Concepts are still all football-related terms like *vapor* and *superfly* which refer to brands of nike football boots. However, some new terms appear when compared to the previous analysis. The same behavior is found on the Off-Topic Concepts. Some of the most important terms from the previous query remain in the extraction, while new terms are added that are related to the term *mercurial*.

In this example, it is clear that the addition of the term *mercurial* to the query introduced some documents that are not relevant to the user. This is due to the fact that *mercurial* is thorougly used as an adjective to define personality of people, which brings various groups of documents relating to various subjects such as football players behavior and political and cultural discussions. The off-topic concept suggestion allows nevertheless the identification of terms such as *control* and *repository*, which relate to a cross-platform, distributed revision control tool for software developers that is also named mercurial.

4.4 Performance Evaluation

In this section, the performance of CSC is evaluated. The key focus here is "time to insight" for end users, as this determines whether the approach can really provide interactive and high-value feedback.

To evaluate the overall performance, the two queries proposed in the previous section are analyzed. For each of these queries, two timespans are proposed - the 2 first months of

2012, and the 12 months of 2011. The performance of the algorithm is presented in Table 1. The results are presented with the entire document corpus and also for a downsampling of the corpus with a 95% of confidence level for a 1% margin of error.

	CCI-Concepts	
	f50	*f50* and *mercurial*
2 months, entire corpus	5.083 documents analyzed in 52 seconds	18.962 documents analyzed in 120 seconds
12 months, entire corpus	27.851 documents analyzed in 284 seconds	69.265 documents analyzed in 852 seconds
2 months, downsampling	3.325 documents analyzed in 37 seconds	6.376 documents analyzed in 66 seconds
12 months, downsampling	7.142 documents analyzed in 81 seconds	8.436 documents analyzed in 78 seconds

Table 1: Concept Suggestion Component's Performance Evaluation

As seen on Table 1, when executed against the entire document corpus for a whole year, the algorithm's execution time can reach up to almost 15 minutes. Such a long time is prohibitive for interactive use. However, the use of the downsampling technique enables its usage due to its great enhancement in overall performance.

As seen on the analysis, the downsampling enables CSC to execute up to 10x faster, allowing a fast enough output for the user, even when a very demanding query is performed. Seeing that, for the analyzed set-up, the maximum number of processed documents is smaller than 10.000 documents, the downsampling should allow in every situation a responsive enough execution.

Figures 11 and 12 compare the extracted concepts when the entire document corpus is analyzed and when they are downsampled. The pictures portray the most downsampled scenario - the 12 months extraction of both *f50* and *mercurial* CCI-Concepts.

Figure 11: Comparison of Relevant Concepts Extraction With Downsampling

The figures show that, in fact, the huge gain in performance does not substantially impact the quality of the extraction. As expected, the most important terms for both the Relevant and Off-Topic Concepts are found even when the downsampling is performed. In the

Figure 12: Comparison of Off-Topic Concepts Extraction With Downsampling

presented example, for both the Relevant and Off-Topic Concepts, the 10 most important extracted terms were the same when performing downsampling - 100% precision when comparing both approaches. When all extracted terms are considered, the downsampled execution achieved a precision of 77% and 93% for the Relevant and Off-Topic Concepts respectively.

5 Conclusions and Future Research

This work proposes an approach that provides Social Media Analysts with relevant and off-topic concepts within their analysis domain to effectively support them in configuring Social Media Analysis dashboards for their stakeholders.

The core of this approach is the Concept Suggestion Component (CSC). It is comprised of four main modules. First, the documents are randomly sampled to enable the algorithm's execution in a responsive manner. Then, a term extraction module extracts, from each analyzed document, the terms that best define its content. Next, a clustering module uses these extracted terms to group the documents according to their content, forming well defined clusters. Finally, the concept suggestion module compares each cluster to the user taxonomy and suggests Relevant and Off-Topic concepts. The implementation of CSC has been evaluated in several case studies, which have shown that CSC improves the Social Media Analysis workflow of IBM Cognos Consumer Insight. As shown by these examples, its use leads to a better user experience on the definition and refinement of the analysis space, resulting in an analysis that is both more focused as well as more comprehensive.

The results presented in this article show that, in fact, our approach to term extraction and classification is faster than traditional document clustering. On top of that, the modularization of the solution allows the application of parts of the proposed algorithm to other analysis scenarios that can benefit from term extraction and document clustering, such as comparing the vocabulary and themes used by different social media authors about the same topic or product.

References

[Bro97] A. Z. Broder. On The Resemblance And Containment of Documents. In *Compression and Complexity of Sequences*, pages 21–29, Salerno, Italy, June 1997. IEEE Computer Society Press.

[CCD99] Jamie Callan, Margaret Connell, and Aiqun Du. Automatic Discovery of Language Models for Text Databases. In *In Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 479–490. ACM Press, 1999.

[CKPT92] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J.W. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1992.

[Cor09] Sean Corcoran. The Broad Reach Of Social Technologies. Technical report, Forrester Research, 2009.

[DB79] D.L. Davies and D.W. Bouldin. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1:224–227, 1979.

[DBE99] A. Demiriz, K. Bennett, and M.J. Embrechts. Semi-Supervised Clustering Using Genetic Algorithms. In *In Artificial Neural Networks in Engineering (ANNIE-99*, pages 809–814, 1999.

[IBM11] IBM. IBM Cognos Consumer Insight. Published online at http://www.ibm.com/software/analytics/cognos/analytic-applications/consumer-insight, 2011.

[JKLV99] Fan Jiang, Ravi Kannan, Michael L. Littman, and Santosh Vempala. Efficient Singular Value Decomposition via Improved Document Sampling. Technical report, DEPT. OF COMPUTER SCIENCE, DUKE UNIVERSITY, 1999.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Comput. Surv.*, 31, 1999.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[MT04] R. Mihalcea and P. Tarau. TextRank: Bringing Order into Texts. *Proceedings of EMNLP*, 2004.

[RGA04] J. Ruoming, A. Goswami, and G. Agrawal. Fast and Exact Out-of-core and Distributed K-Means Clustering. *Knowledge and Information Systems*, 10, 2004.

[SWY75] G. Salton, A. Wong, and C.S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 1975.

[TC92] H. R. Turtle and W. B. Croft. A Comparison of Text Retrieval Models. *Comput. J.*, 35(3):279–290, 1992.

[Wei06] D. Weiss. *Descriptive Clustering as a Method for Exploring Text Collections*. PhD thesis, Poznań University of Technology, 2006.

Datensicherheit in mandantenfähigen Cloud Umgebungen

Tim Waizenegger[1], Oliver Schiller[1], Cataldo Mega[2]

[1]Universität Stuttgart, Institut für Parallele und Verteilte Systeme
Universitätsstr. 38, 70569 Stuttgart
{Tim.Waizenegger,Oliver.Schiller}@ipvs.uni-stuttgart.de

[2]IBM Software Group
Schönaicherstr. 220, 71032 Böblingen
Cataldo_Mega@de.ibm.com

Abstract: Cloud Computing wird aktuell hauptsächlich für wissenschaftliches Rechnen und endkundenorientierte Dienste verwendet, da die Kostenersparnis hier ein besonders wichtiger Faktor ist. Die Betreiber von Cloud Plattformen sind jedoch immer stärker daran interessiert Cloud Dienste auch im Enterprise Segment anzubieten, um hier gleichermaßen von Kostenvorteilen zu profitieren.

Die Kundenresonanz aus diesem Segment lässt jedoch zu wünschen übrig. Die Gründe dafür sind Bedenken bezüglich Datensicherheit und -vertraulichkeit in mandantenfähigen Systemen. Um diesem Problem zu begegnen, haben wir die Herausforderungen bei der Absicherung von mandantenfähigen Cloud Diensten untersucht, und den Umgang mit vertraulichem Schlüsselmaterial und Anmeldedaten als Schwachstelle identifiziert.

Dieser Beitrag zeigt eine konzeptionelle Lösung zur zentralen Ablage und Zugriffsverwaltung sensibler Daten, sowie deren prototypische Implementierung innerhalb der IBM Cloud Lösung *SmartCloud Content Management*.

1 Einleitung

Cloud Computing zeichnet sich als neues, zukunftsträchtiges Vertriebsmodell für IT ab. Dies wird durch die zunehmende Anzahl existierender Angebote auch im und für das Geschäftsumfeld bestätigt; Gartner sagt voraus, dass 2013 mehr als 80% aller neuen, kommerziellen Anwendungen im Geschäftsumfeld auf Cloud Plattformen betrieben werden [Ga11]. Darüber hinaus wird vorausgesagt, dass bis 2016 40% der Unternehmenskunden eine unabhängige Sicherheitsuntersuchung als Voraussetzung für den Einsatz einer Cloud-Lösung einführen werden. Die zentrale Herausforderung besteht darin, eine ausreichende Sicherheit zu gewährleisten [XF11].

Das erfolgreiche Begegnen dieser Herausforderung wird insbesondere durch eine weitere, für den Erfolg von Cloud Computing essentielle Eigenschaft schwierig: die gemeinsame Nutzung von physischen Ressourcen durch mehrere Kunden [NI11].

Im Geschäftsumfeld entspricht ein Kunde einem Unternehmen, d.h. einer organisatorisch

abgeschlossenen Einheit. In diesem Fall wird der Kunde auch als Mandant bezeichnet. Die Konsolidierung mehrerer Mandanten auf eine physische Ressource wird dementsprechend als *Mandantenfähigkeit* bezeichnet [Wa10]. Mandantenfähigkeit verbessert die Skaleneffekte und verringert die Betriebskosten je Mandant. Dies erhöht zum einem die Gewinnspanne des Betreibers und zum anderen erlaubt es, den Dienst zu attraktiveren Konditionen - verglichen zu konventionellen Lösungen - anzubieten. Mit der Nutzung von Ressourcen durch mehrere Kunden entstehen jedoch zugleich neue Risiken, die zusätzliche Maßnahmen seitens der Cloud Betreiber erfordern, um Datensicherheit zu gewährleisten.

Ein typischer Cloud Dienst ist als Komposition von Programmen aufgebaut welche durch interne Kommunikation einen Mehrwertdienst bilden. Aufgrund dieser Heterogenität ist die Absicherung solcher Systeme alleine durch Einschränkung des Zugriffs schwierig. Falls der physische Zugang zu einem System oder der logische Zugang zu den Daten für Angreifer nicht verhindert werden kann, so bleibt nur die Verschlüsselung der Daten, um unerlaubten Zugriff auszuschließen.

Durch Datenverschlüsselung findet eine Verschiebung des Risikos von den zuvor sensiblen Daten auf das Schlüsselmaterial statt. Dadurch wird eine Reduktion der Menge an kritischen Daten um mehrere Größenordnungen erreicht, mit der die sichere Aufbewahrung dieser Daten erst möglich ist.

Der Einsatz von Verschlüsselung und die hierfür notwendigen Schlüssel machen den Einsatz eines Schlüsselspeichers unerlässlich. Ein solches System muss dafür Sorge tragen, dass ein unerlaubter Zugriff sowie ein Verlust von Schlüsselmaterial verhindert werden. Die verschlüsselten Daten können nur dann als sicher angesehen werden, wenn der Zugriff auf Schlüssel effektiv kontrolliert wird [BEE+10].

Ein wichtiger Aspekt von Cloud Computing ist die Loslösung der Dienste und Daten von physischen Maschinen. Wird daher Datenverschlüsselung in einer Cloudanwendung eingesetzt, so ist der verwendete Schlüssel an die Daten und nicht an die physische Maschine gebunden. Andere physische Maschinen in der Cloud, die zu einem späteren Zeitpunkt auf diesen Daten operieren sollen, benötigen ebenso Zugang zu dem Schlüssel. Damit die verschlüsselten Daten effektiv geschützt sind, muss verhindert werden, dass ein Angreifer, der Zugriff auf die Daten erlangt, auch den Schlüssel erhält. Es ist daher nicht möglich, Daten und Schlüssel gemeinsam abzulegen; ein separater Mechanismus zur Ablage und Verteilung der Schlüssel ist notwendig.

Ein Schlüsselspeicher, der in Cloudanwendungen eingesetzt werden kann, muss Kriterien erfüllen, die über das hinausgehen, was in konventionellen Installationen erforderlich ist. Dynamische Skalierung und Hochverfügbarkeit auf unzuverlässiger Hardware sind Eckpfeiler des Cloud Computings, die wesentlich zum Erfolg des Konzepts beitragen. Ein Schlüsselspeicher muss diese Aspekte daher unterstützen. Kapitel 2.1 zeigt, dass diese Anforderungen in bestehenden Lösungen nur unzureichend erfüllt sind.

Das hier vorgestellte Konzept schließt die Lücke zwischen Schlüsselverwaltung und Cloud Computing und ermöglicht damit den effektiven Einsatz von Verschlüsselung in Cloud Umgebungen.

In Kapitel 2 werden die Basistechnologien aus Cloud Computing und Verschlüsselung eingeführt, sowie ein Überblick über *IBM SmartCloud Content Management* gegeben. SCCM

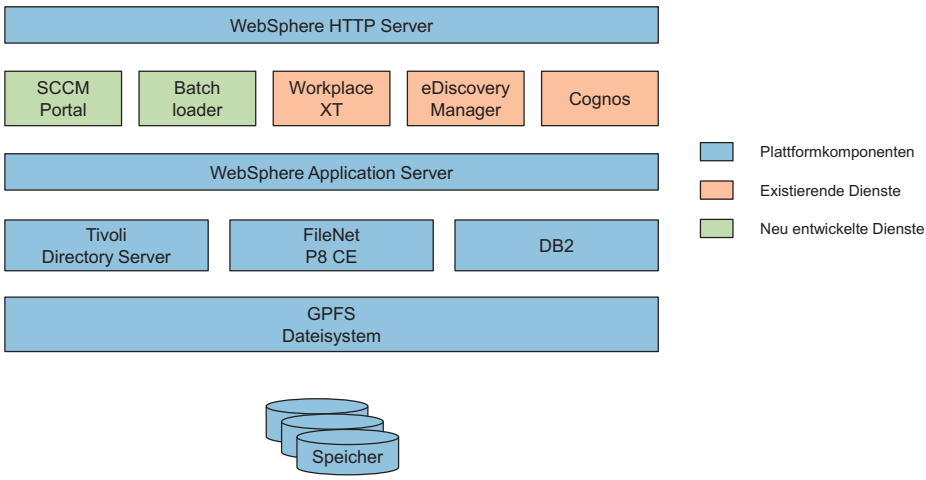


Abbildung 1: Komponenten der SmartCloud Content Management Plattform

ist die Enterprise Cloud-Lösung, anhand derer unsere Untersuchungen stattfanden. Kapitel 3 behandelt im Detail das Konzept und die Architektur unseres Schlüsselspeichers und zeigt dessen Integration in SCCM. In Kapitel 4 bewerten wir unser Konzept anhand der Voraussetzungen des Cloud Umfelds und geben Hinweise auf künftige Arbeiten in Kapitel 5.

2 Hintergrund – IBM SmartCloud Content Management

Das Ziel von *IBM SmartCloud Content Management (SCCM)* ist es, dem Kunden einen Dienst zur Verwaltung sensibler Unternehmensdaten anzubieten, der einen geringen Investitionsaufwand erfordert und günstige Betriebskosten bietet [IB12a]. Diese Randbedingungen legen eine cloudbasierte mandantenfähige Lösung nah, da durch das Wegfallen dedizierter Hardware beim Kunden die Anfangsinvestition niedrig bleibt, während durch Mandantenfähigkeit die Betriebskosten des Anbieters reduziert werden.

SCCM nutzt bestehende IBM Produkte, um die nötige Basisfunktionalität bereitzustellen, sowie neu entwickelte Komponenten wie Mechanismen zum Import von Daten und eine einheitliche Konfigurations- und Verwaltungsoberfläche. Abbildung 1 zeigt den Aufbau von SCCM und gibt einen Überblick über die Komponenten.

Ein solch komponentenbasierter Aufbau ist typisch für Enterprise Cloud Lösungen, da

die Wiederverwendung von etablierten Komponenten im Umfeld von Enterprise-Software, aus Kosten- und Zuverlässigkeitsgründen, der Neuimplementierung vorgezogen wird. Die einzelnen Komponenten eines solchen Systems benötigen jeweils Zugang zu gemeinsamem Schlüsselmaterial, wenn Verschlüsselung eingesetzt wird. Eine weitere Quelle sicherheitsrelevanter Informationen ergibt sich aus der internen Kommunikation der Komponenten, insbesondere der Middleware. Im Fall von SCCM sind dies die DB2 Datenbank und die FileNet P8 Content Engine, deren Dienste von sämtlichen darüber liegenden Komponenten benutzt werden [MKW+09]. Um diese Kommunikation abzusichern, ist die Angabe von Anmeldedaten bei Benutzung der Dienste unerlässlich. Im Zuge der Implementierung einer zentralen Ablage für kryptografisches Schlüsselmaterial wird daher auch die Ablage dieser Anmeldedaten als eine Aufgabe des Schlüsselspeichers angesehen. Der bisherige Ansatz zur Speicherung dieser Daten bestand in der Ablage in Konfigurationsdateien und Konfigurationsdatenbanken, meist im Klartext oder in trivial verschleierter Form [IB12b].

Es besteht daher die Notwendigkeit für ein System, das den Komponenten einer Cloudanwendung sicheren Zugriff auf zentral gespeichertes Schlüsselmaterial ermöglicht. Bei den relevanten Komponenten in SCCM handelt es sich um Anwendungen im Kontext des WebSphere Application Server. Wir erfordern daher ein System, welches sich flexibel an solche Anwendungen anbinden lässt.

2.1 Bestehende Lösungen zur Schlüsselverwaltung

Vor der Entwicklung des Schlüsselspeichers haben wir untersucht, ob eine Neuentwicklung notwendig, oder eine verfügbare Lösung geeignet ist.

Bestehende Verschlüsselungsprodukte setzen stets eine proprietäre Schlüsselverwaltung ein, die nicht von unterschiedlichen Komponenten in einem heterogenen Cloud Umfeld benutzt werden kann. Mit *KMIP*[1] finden aktuell Bemühungen statt, einen solchen Standard zu schaffen. KMIP definiert ein Protokoll zur Kommunikation zwischen Client und Server, welches den gemeinsamen Einsatz von Produkten unterschiedlicher Hersteller ermöglichen soll. Serverseitig unterstützen die im Folgenden vorgestellten Produkte KMIP, jedoch hat kein Hersteller einen KMIP Client im Angebot, der sich flexibel an eigene Anwendungen anbinden lässt.

Die kommerziellen Verschlüsselungsprodukte von *SafeNet*[2] und *Vormetric*[3] bieten eine zentrale Schlüsselverwaltung und die Möglichkeit unterschiedliche Clients anzubinden. Beide Produkte unterstützen jedoch nur eine definierte Menge von Clientanwendungen und können nicht mit selbst entwickelten Anwendungen genutzt werden. Ein Konzept für Mandantenfähigkeit ist ebenfalls nicht vorhanden. Von den Herstellern wird lediglich die Abbildung von Mandanten auf andere strukturelle Objekte in der Schlüsselverwaltung empfohlen.

[1] `https://www.oasis-open.org/committees/kmip`
[2] `http://www.safenet-inc.com`
[3] `http://www.vormetric.com`

Lösungen zum Speichern und Erzeugen von kryptografischem Material sind mit dem Java Key Store und der PKCS12 Implementierung bereits in der Programmiersprache Java enthalten [Or04]. In unserem Prototypen werden diese zum Erzeugen, Signieren und Überprüfen der Zertifikate benutzt, nicht aber, um das Schlüsselmaterial der Clients zu speichern. Der Grund dafür ist die Architektur dieser Key Stores. Sie sind dafür ausgelegt, eine überschaubare Menge von kryptografischem Material zugehörig zu *einer* Entität zu speichern. Weder Anforderungen für Mandantenfähigkeit sind erfüllt, noch solche bezüglich Skalierbarkeit.

Im Gegensatz dazu ist unser Konzept für den Einsatz in Cloud Umgebungen vorgesehen und unterstützt massive Mandantenfähigkeit sowie horizontale Skalierung, wie in Kapitel 4 deutlich wird.

3 Der Schlüsselspeicher für SCCM

Im Folgenden wird das von uns entwickelte Konzept eines Schlüsselspeichers für Cloudanwendungen vorgestellt und anschließend die prototypische Implementierung in SCCM beschrieben. Die zentralen Entwurfsaspekte unseres Konzepts ergeben sich aus den Anforderungen des Cloud Computings. Dies umfasst die funktionalen Aspekte Skalierbarkeit und Hochverfügbarkeit, sowie einen hohen Sicherheitsstandard, der durch die Aufteilung von Zuständigkeiten erreicht wird. Die Zuständigkeiten werden unter drei Parteien aufgeteilt: Einem *Client*, der Zugriff auf Schlüssel beantragt, einem *Server*, der über den Zugriff entscheidet, und einem *Kundenadministrator*, der die Berechtigungen der Clients definiert und verwaltet. Im Folgenden wird mit *Client* eine funktionale Komponente unseres Konzepts bezeichnet, ein Kunde als organisatorische Einheit wird stets *Kunde* genannt.

3.1 Systemarchitektur

Unser Schlüsselspeicher ist als Client-Server Lösung entwickelt. Die Serverkomponente leistet das zentrale Speichern der Schlüssel und die Zugriffsverwaltung, während die Clientkomponente in die Cloudanwendungen integriert ist. Sie implementiert das Kommunikationsprotokoll und bietet der Anwendung über eine Programmierschnittstelle Zugriff auf die Dienste des Servers.

Eine Client-Server-Architektur ist notwendig, da im Cloud-Umfeld verschiedene Instanzen der Anwendung auf denselben Daten operieren und damit Zugriff auf gemeinsames Schlüsselmaterial erfordern. Darüber hinaus werden durch diese Trennung von Zuständigkeiten die sensiblen Schlüssel auf den Server verschoben, was es ermöglicht, zentral über jeden Datenzugriff zu entscheiden.

Als Ergebnis können sämtliche auf dem Client gespeicherten Daten als unkritisch angesehen werden, da ein Angreifer lediglich verschlüsselte Daten vorfindet, jedoch keinen Zugang zu dem Schlüssel hat.

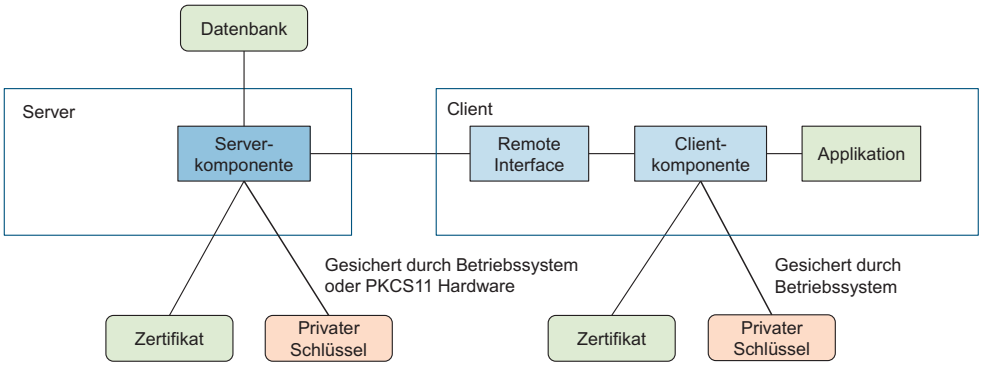


Abbildung 2: Client-Server Architektur des Schlüsselspeichers

Abbildung 2 zeigt die Architektur des Systems. Die Serverkomponente ist an eine relationale Datenbank angebunden, welche das Schlüsselmaterial und die Zugriffsberechtigungen speichert.

Die Cloudanwendung greift über die Programmierschnittstelle der Clientkomponente auf den Server zu, um Schlüsselmaterial zu lesen oder zu schreiben. Für die Anwendung erfolgt dieser Zugriff transparent, da die Clientkomponente Netzwerkkommunikation und Authentifizierung übernimmt.

Falls eine Cloudanwendung konsequent Verschlüsselung mit einem zentralen Schlüsselspeicher einsetzt, so wird dieser zu einer kritischen Komponente, ohne die die gesamte Anwendung nicht funktionieren kann. Daher war bei der Entwicklung unseres Konzepts die Ausfallsicherheit und Redundanz der Serverkomponente ein wichtiges Kriterium. Um auf bestehende, etablierte Technologien zurückgreifen zu können, ist die Serverkomponente zustandslos ausgelegt. Dies wird umgesetzt mit einer anfragebasierten Authentifizierung. Damit muss der Server keine Sitzungen verwalten und kann jede Anfrage unabhängig von den vorhergehenden bearbeiten. Die Anfragen können daher beliebig auf ein verteiltes Cluster von Anwendungsservern verteilt werden. Zur Ablage des Schlüsselmaterials wird eine relationale Datenbank benutzt, welche durch Replikation oder Partitionierung ebenfalls verteilt ausgeführt werden kann.

In großen Cloud-Umgebungen mit hohem Lastaufkommen erlauben diese Designaspekte neben der Ausfallsicherheit, durch horizontale Skalierung die Leistung des Systems auf-

recht zu erhalten.

Der Schlüsselspeicher wurde für die Anwendung in SCCM - einer Enterprise-Lösung - entwickelt. Mandantenfähigkeit bedeutet in diesem Kontext, dass sich mehrere Kunden zwar die physische Maschine teilen, die darüber liegende Software aber zu genau einem Kunden gehört. Der Server ist daher in der Lage, Clients zu bedienen, die zu verschiedenen Kunden gehören. Die Clientkomponente ist nicht mandantenfähig ausgelegt, sondern an einen Kunden gebunden. Sie weist die Zugehörigkeit zu einem Kunden während der Authentifizierung dem Server gegenüber aus, sodass ein Zugriff auf fremdes Schlüsselmaterial ausgeschlossen ist. Das Kommunikationsprotokoll basiert auf gegenseitiger Authentifizierung mit asymmetrischer Verschlüsselung.

Sowohl die Server- als auch die Clientkomponente benötigen ein Zertifikat und den dazugehörigen privaten Schlüssel. Der private Schlüssel ist das Gegenstück zu dem öffentlichen Schlüssel in dem Zertifikat. Die Rolle der Zertifikate und deren Erstellung wird in Kapitel 3.2.1 behandelt. Der private Schlüssel stellt das letzte Glied in einer Kette von Risikoverschiebungen dar, weshalb seine Sicherheit von kritischer Bedeutung ist. Der private Schlüssel auf Seite des Servers wird benutzt, um den Inhalt der Datenbank zu verschlüsseln. In unserer Lösung ist er daher in einem Hardware Key-Store abgelegt, aus dem ihn nur der legitime Serverprozess auslesen kann [Gu09]. Der private Schlüssel des Clients wird nicht zur Verschlüsselung von Daten verwendet, sondern dient dem Client dazu sich gegenüber dem Server zu authentifizieren. Er wird mit Betriebssystemmitteln wie SELinux vor unerlaubtem Zugriff geschützt [KAAS11].

Um die Sicherheit der Daten zu gewährleisten, darf der Schlüsselspeicher nur Zugriff auf Schlüssel gewähren, falls die Anfrage von einer Clientkomponente stammt, die die nötige Berechtigung aufweist. Im Folgenden wird daher beschrieben, wie der Server das Schlüsselmaterial ablegt und mit einem neuartigen Autorisierungsschema die Clientberechtigungen zuordnet.

3.2 Autorisierungsschema – Zugriffsabstraktion über Gruppen

Das Autorisierungsschema beschreibt die Datenstrukturen und Prozesse, durch welche die Berechtigungen der Clients den Objekten in dem Schlüsselspeicher zugeordnet werden.

Das Entity-Relationship Modell in Abbildung 3 zeigt die Datenobjekte, die das Autorisierungsschema ausmachen. Es kann anhand der enthaltenen Beziehungen direkt in ein relationales Datenmodell für den Server umgesetzt werden. Die drei wesentlichen Objekte in dem Schema sind der Client, die Gruppe und der Schlüssel.

Ein *Client* im Modell des Autorisierungsschemas ist eine Instanz der im vorigen Abschnitt beschriebenen Clientkomponente, die auf mindestens einem Cloud-Rechner aktiv ist. Der *Schlüssel* ist das Datenobjekt, welches das sensible Schlüsselmaterial enthält und über die *Gruppe* mit den Clients verknüpft ist, denen Zugriff gewährt werden soll. Die Kardinalitäten der Objektbeziehungen in Abbildung 3 verdeutlichen die Funktion der Gruppen als zentralen Aspekt des Autorisierungsschemas. Jeder Schlüssel ist genau einer Gruppe zugeordnet, ein Client wiederum kann Mitglied beliebig vieler Gruppen sein. Ebenso kann

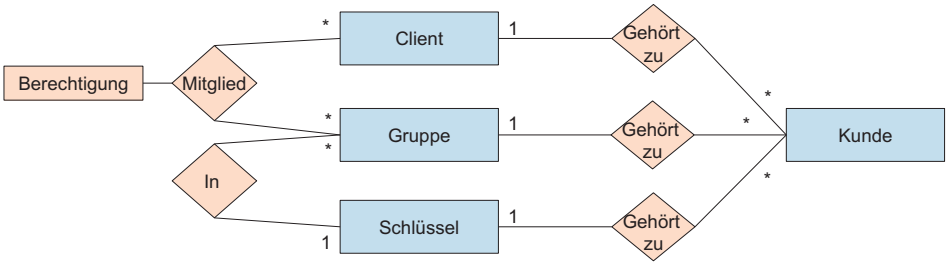


Abbildung 3: Zugriffskontrolle über Gruppen

jede Gruppe beliebig viele Mitglieder haben.

Die Beziehung, welche die Gruppenmitgliedschaft ausdrückt, hat ein Attribut welches die Art der Berechtigung kennzeichnet, die der Client auf Schlüssel dieser Gruppe hat. In unserem Konzept sind drei Mitgliedschaftstypen umgesetzt: *schreibend*, *lesend* und *hinzufügend*. Der erste Typ erlaubt vollständigen Zugriff, der das Hinzufügen, Löschen und Lesen von Schlüsseln ermöglicht. Die letzteren Typen schränken diesen Zugriff entsprechend ein. Der Typ *hinzufügend* kann dabei wie ein Nachttresor verstanden werden, in den lediglich neues Schlüsselmaterial eingefügt, jedoch kein bestehendes manipuliert oder gelesen werden kann.

Mit diesem Autorisierungsschema ist es dem Server möglich, über Zugriffe zu entscheiden, ohne Wissen über den Client vorzuhalten. Der Server speichert lediglich die Zuordnung der Schlüssel zu Gruppen, entsprechend obigem Schema. Um über den Zugriff eines Clients zu entscheiden, benötigt der Server dessen Gruppenmitgliedschaft. Im Folgenden wird die Methode beschrieben, durch welche ein Client dem Server gegenüber die Berechtigungen ausweisen kann, welche der Kundenadministrator ihm zugewiesen hat.

3.2.1 Zertifikatbasierte Autorisierung der Clients

Die Verwendung von Zertifikaten zur *Authentifizierung* hat eine lange Tradition und gilt anderen Konzepten wie der passwortbasierten Authentifizierung, aufgrund der Länge des geheimen Datums, als überlegen [Am94].

Um die Interaktion mit der Serverkomponente zu minimieren und die Aufteilung von Zuständigkeiten umzusetzen, wurde im Rahmen dieses Beitrags die Verwendung von Zertifikaten auf die *Autorisierung* ausgeweitet. Dieser Ansatz macht Gebrauch von erweiterten Attributen in Zertifikaten, mit denen es möglich ist, abgesehen von den Basisinformationen zur Identifikation, zusätzliche Daten in dem signierten Teil des Zertifikats abzulegen [RSA83].

Wir verwenden diese Datenfelder für das Speichern von Zugriffsberechtigungen, wobei die Signatur des Zertifikats gewährleistet, dass jede Manipulation entdeckt wird. Dabei sind die anfangs erwähnten drei Parteien beteiligt. Das Zertifikat wird von einem *Kunde-*

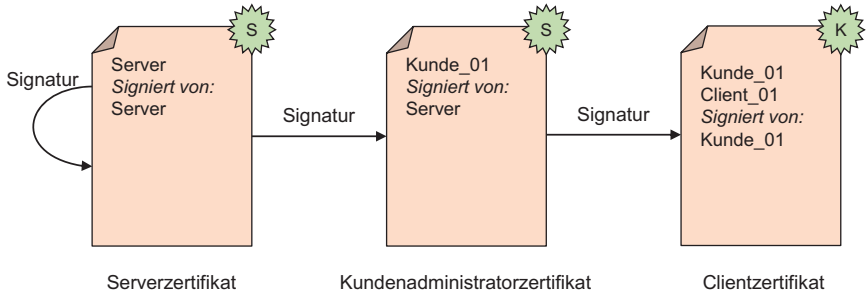


Abbildung 4: Beziehungen der Zertifikate

nadministrator überprüft und signiert und dem *Client* übergeben, der das Zertifikat benutzt, um seine Berechtigungen gegenüber einem *Server* auszuweisen. Das Vertrauensverhältnis zwischen Kundenadministrator und Server wird auf den Client erweitert, da der Server einem Zertifikat vertraut indem er die Legitimität der Signatur überprüft, womit er sich von der Echtheit des Zertifikats überzeugt hat [An01].

3.3 Aufteilung der Zuständigkeiten durch Zertifikatshierarchie

Im Folgenden wird beschrieben, wie das Autorisierungsschema aus dem vorigen Kapitel mittels zertifikatbasierter Autorisierung umgesetzt wird, und in welcher Beziehung die beteiligten Zertifikate stehen.

Um die Zuordnung von Gruppenberechtigungen auf Clients mit Zertifikaten umzusetzen, wird für jeden Client ein Zertifikat ausgestellt. Dieses wird stets einem Client, also einer Instanz der Clientkomponente, zugeordnet und enthält die Gruppenmitgliedschaften und Berechtigungen dieses Clients.

Die nötige Hierarchie von Zertifikaten ist dargestellt in Abbildung 4. Sie beginnt mit dem Server, welchem bei der Installation des Systems ein selbst-signiertes Zertifikat zugewiesen wird. Dieses Serverzertifikat wird verwendet, um das Kundenadministratorzertifikat bei der Einführung eines neuen Kunden zu signieren. In diesem Prozess behält der Server eine Kopie des ausgestellten Kundenzertifikats, mit dessen Hilfe er die Gültigkeit späterer Clientzertifikate überprüft [CA61]. Der Kundenadministrator ist nun in der Lage die Cloudanwendung zu provisionieren und Clientzertifikate mit Gruppenzugehörigkeit auszustellen.

Mit dieser Aufteilung der Zuständigkeiten ist der Herausgeber der Zertifikate - also der Kundenadministrator - dafür verantwortlich, nur legitimen Clients Zugriff zu gewähren.

3.4 SCCM Integration

Zur Evaluation unseres Konzepts wurde ein Prototyp in Java implementiert, dessen Clientkomponente als Dienst von den Komponenten der Cloudanwendung SmartCloud Content Management benutzt wird. Als Integrationsebene wurden die Anwendungen des WebSphere Application Server gewählt, da diese verteilt auf unterschiedlichen Maschinen laufen und verschiedene Aufgaben erfüllen, was eine Zuordnung von disjunkten Zugriffsrechten ermöglicht. Jede dieser Applikationen aus Abbildung 1 wird daher als Client behandelt und erhält ein eigenes Zertifikat mit den Zugangsberechtigungen, die für diese Applikation erforderlich sind.

Wie in Kapitel 1 beschrieben, soll der zentrale Schlüsselspeicher sowohl kryptografisches Schlüsselmaterial, als auch Anmeldedaten zu internen Systemen absichern. eine Clientkomponente für unseren Schlüsselspeicher wird daher an die Anwendungen *SCCM Portal*, *Batch loader*, *WorkplaceXT* und *eDiscovery Manager* angebunden. Für jede dieser Anwendungen wird ein Clientzertifikat ausgestellt, das die nötigen Zugriffsberechtigungen widerspiegelt.

Die in Kapitel 3.2 eingeführte parametrisierte Gruppenmitgliedschaft macht es nun möglich, den Batch loader auf einfügenden Zugriff zu beschränken, wenn er neue Schlüssel für geladene Daten speichert. Die Komponente eDiscovery Manager dient dem Finden und Abrufen von Daten und erhält daher lesenden Zugriff, während WorkplaceXT schreibenden Zugriff erfordert. Das Cognos Berichterstellungssystem ist in die Benutzeroberfläche des SCCM Portals integriert und benötigt an dieser Stelle nur lesenden Zugriff auf die DB2 Datenbank, weshalb hier ein Clientzertifikat vergeben wird, das lediglich Zugang zu den Anmeldedaten für einen eingeschränkten DB2 Benutzer ermöglicht.

4 Diskussion und Evaluation

Um die Eignung des hier vorgestellten Konzepts in einem Cloud Umfeld zu beurteilen, werden im Folgenden die Skalierbarkeit, Hochverfügbarkeit und Sicherheit anhand von Aspekten der Systemarchitektur gezeigt.

Hochverfügbarkeit wird mit horizontaler Skalierung erreicht. Beim Ausfall von Servern stehen weitere zur Verfügung, auf welche die Anfragen umgeleitet werden. Derselbe Mechanismus erlaubt es, bei großer Last die Anfragen parallel auf verteilten Servern auszuführen und damit Engpässe zu vermeiden.

Zwei wichtige Eigenschaften der Architektur ermöglichen eine horizontale Skalierung der Serverkomponente über ein Cluster von Anwendungs- und Datenbankservern. Dies sind die Zustandslosigkeit der Serverkomponente, sowie die Datenhaltung in einer relationalen Datenbank. Damit ist es möglich, horizontale Skalierung mit Funktionen der Middlewarekomponenten umzusetzen anstatt eine anwendungsspezifische Lösung zu erfordern.

Auf Seite des Anwendungsservers kann die Serverkomponente, ebenfalls wegen der Zustandslosigkeit, in beliebig vielen Instanzen ausgeführt werden. Ein Load Balancer ver-

teilt die Anfragen der Clients unter diesen. Unsere Implementierung setzt dieses Konzept durch den Einsatz eines WebSphere Clusters um. Dieses besteht aus einer Anzahl von Servern, die auf separaten physischen Maschinen laufen und jeweils eine Instanz der Schlüsselspeicher Serverkomponente ausführen. Der WebSphere Cluster wird verwaltet von einem zentralen Deployment Manager. Dieser koordiniert das Hinzufügen und Entfernen von Servern innerhalb des Clusters und dient als Load Balancer welcher Clientanfragen auf die Clustermitglieder verteilt[4]. Um die Verfügbarkeit des Systems auch bei Ausfall des Deployment Managers zu gewährleisten, wird dieser in einer active-passive Konfiguration betrieben. Eine zweite passive Instanz des Deployment Managers läuft dabei auf einer separaten physischen Maschine und synchronisiert ihre Konfiguration mit dem aktiven Deployment Manager. Sollte dieser nicht mehr erreichbar sein, wird die passive Instanz aktiv und nimmt die Anfragen entgegen.

Im Gegensatz zu dem Anwendungsserver sind die Instanzen der relationalen Datenbank nicht unabhängig voneinander, da sie einen konsistenten Datenbestand repräsentieren müssen. Es ist daher eine Synchronisation der Instanzen notwendig. Die Synchronisation der Datenbankinstanzen kann mit unterschiedlichen Verfahren umgesetzt werden.

In unserem Konzept wurde die Skalierung der Datenbank durch Replikation und Partitionierung gelöst[5]. Besonders in Szenarien, die nur wenige schreibende Zugriffe auf den Schlüsselspeicher erfordern, ist Replikation gut geeignet, da sich eine konsistente Kopie des Datenbestandes auf jedem Server befindet. Die Anfragen der Anwendungsserver können daher beliebig auf die Datenbankserver verteilt werden, was Engpässe vermeidet.

In Szenarien, die viele Schreibzugriffe auf verteilte Datenbankserver erfordern, ist vollständige Replikation jedoch schlecht geeignet, da zur Erhaltung der Konsistenz eine teure Synchronisation der Datenbankserver erforderlich ist. Eine bessere Strategie für solche Szenarien ist Datenbankpartitionierung, da sie keine Synchronisation der verteilten Datenbanken erfordert.

Wir verwenden daher einen heterogenen Ansatz aus Partitionierung und Replikation, um die Vorteile beider Mechanismen auszunutzen. Wir verwenden horizontale Partitionierung auf dem Schlüsselmaterial, während die Partitionen auf mehrere Server repliziert werden. Mit derzeitigen Systemen ist nur statische Partitionierung möglich, so dass zum Zeitpunkt der Systemprovisionierung entschieden werden muss, anhand welcher Kriterien die Aufteilung erfolgt. Im Kontext von Enterprise-Anwendungen ist es sinnvoll nach Mandanten zu partitionieren, da so jedem Kunden dedizierte Ressourcen zugesichert werden können. Die Datenpartition eines Kunden wird durch Replikation auf mehrere Server verteilt, um den jeweiligen Anforderungen des Kunden nachzukommen.

Der Grund für die Entwicklung des zentralen Schlüsselspeichers sind Defizite in bestehenden Lösungen. Sie sind nicht mandantenfähig und lassen sich nicht an eigene Anwendungen anbinden. Diese Anforderungen wurden in unserem Konzept berücksichtigt. Mit der Implementierung unseres Prototyps basierend auf einer relationalen Datenbank und einer zustandslosen Serverkomponente werden die, in Cloud Umgebungen wichtigen, Anforde-

[4] http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/welcclusters.html

[5] http://pic.dhe.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.swg.im.iis.db.repl.sqlrepl.doc/topics/iiyrscncsqlreplovu.html

rungen an Skalierung und Hochverfügbarkeit auf etablierte Systeme delegiert.

Die Einführung der zertifikatbasierten Autorisierung ermöglicht es in Cloud Umgebungen Vertrauen beim Kunden zu schaffen, indem dedizierte Kundenadministratoren bestimmt werden, die alleinig über die Zugriffsberechtigungen auf das Schlüsselmaterial entscheiden.

Sicherheit vor Betrug durch die teilnehmenden Parteien ist in diesem System implizit durch die Integrität der Zertifikate gegeben. Ein Kundenadministrator hat legitime Kontrolle über alle Berechtigungen unterhalb seiner Stufe, er kann aber nicht durch Manipulation die Daten anderer Kunden kompromittieren. Der Server kann die Zertifikate, die ein Kundenadministrator ausstellt, eindeutig diesem Kunden zuordnen, selbst wenn der Administrator die Identitätsinformationen des Zertifikats manipuliert. Dasselbe gilt für die Clientzertifikate. Durch die Signatur sind sie vor Manipulation durch jemanden anderen als den Kundenadministrator geschützt.

Unser Konzept der zentralen Schlüsselverwaltung mit verteilten Zuständigkeiten ermöglicht es daher, in einer Cloud Umgebung einen höheren Sicherheitsstandard zu erreichen. Das Risiko wird verschoben von den Rechenknoten in der Cloud auf eine kleine Anzahl von Servern. Durch die neue Rolle des Kundenadministrators wird darüber hinaus - im Cloud Umfeld vermisstes - Vertrauen an den Kunden zurückgegeben und eine attraktive Cloud Plattform geschaffen.

5 Ausblick

Der Kern dieses Beitrags ist die Trennung von Zuständigkeiten zwischen dem Client, welcher kryptografisches Schlüsselmaterial erzeugt und benutzt, und dem Server der es verwaltet. Damit ist die situationsabhängige Autorisierung von Clients möglich, womit die Grundlage geschaffen wurde, mit einem Überwachungssystem kompromittierte Clients zu erkennen, und deren Zugriff zu sperren.

In zukünftigen Arbeiten muss daher untersucht werden, mit welchen Mechanismen eine solche Erkennung möglich ist, und wie sie vor Manipulation geschützt werden kann.

Viele Rechenzentren verfügen über Sicherheitssysteme, die mit Sensoren den unerlaubten Zugang zu Räumen und Maschinen erkennen. Leicht können diese Informationen heran gezogen werden, um kompromittierte Maschinen zu identifizieren.

Literatur

[Am94] Edward Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall, 1994.

[An01] Ross Anderson. *Security Engineering*. John Wiley & Sons, Chichester, 2001.

[BEE+10] Jean Bacon, David Evans, David M. Eyers, Matteo Migliavacca, Peter Pietzuch und Brian Shand. Enforcing end-to-end application security in the cloud (big ideas paper).

In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, Middleware '10, Seiten 293–312, Berlin, Heidelberg, 2010. Springer-Verlag.

[CA61] Steve Lloyd Carlisle Adams. *Understanding Pki: Concepts, Standards, and Deployment Considerations*. Pearson, 1961.

[Ga11] Gartner. Gartner Reveals Top Predictions for IT Organizations and Users for 2012 and Beyond. 2011.

[Gu09] PKCS 11 Base Functionality v2.30: Cryptoki, July 2009.

[IB12a] IBM. Cloud-based content management service. Bericht, 2012.

[IB12b] IBM. IBM WebSphere Application Server V8.5 information center - Java 2 Connector authentication data entry settings, 2012.

[KAAS11] K.A. Khan, M. Amin, A.K. Afridi und W. Shehzad. SELinux in and out. In *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*, Seiten 339 –343, may 2011.

[MKW+09] Cataldo Mega, Kathleen Krebs, Frank Wagner, Norbert Ritter und Bernhard Mitschang. *Content-Management-Systeme der nächsten Generation*, Seiten 539–567. Wissens- und Informationsmanagement; Strategien, Organisation und Prozesse. Gabler Verlag, Wiesbaden, January 2009.

[NI11] National Institute of Standards und Technology. The NIST Definition of Cloud Computing. *Special Publication 800-145*, 2011.

[Or04] Oracle. *Java PKCS11 Reference Guide*, 2004.

[RSA83] R. L. Rivest, A. Shamir und L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 26:96–99, January 1983.

[Wa10] Phil Wainewright. Defining the true meaning of cloud. *ZDNet*, 2010.

[XF11] IBM X-Force. IBM X-Force® 2011 Mid-year Trend and Risk Report. Bericht, IBM Security Solutions, 2011.

Demonstrating Near Real-Time Analytics with IBM DB2 Analytics Accelerator

Daniel Martin, Iliyana Ivanova, Raphael Mueller,
Luis Eduardo Velez Montoya, Klaus Maruschka

IBM DB2 Analytics Accelerator Development
IBM Deutschland Research & Development GmbH
Schoenaicherstrasse 220
71032 Boeblingen, Germany
danmartin@de.ibm.com, iivanova@de.ibm.com, raphaelm@de.ibm.com
velez@de.ibm.com, marusch@de.ibm.com

Abstract: Version 3 of the IBM[1] DB2 Analytics Accelerator (IDAA) takes a major step towards the vision of a universal relational DBMS that transparently processes both, OLTP and analytical-type queries in a single system. Based on heuristics in DB2 for z/OS, the DB2 optimizer decides if a query should be executed by "mainline" DB2 or if it is beneficial to forward it to the attached IBM DB2 Analytics Optimizer that operates on copies of the DB2 tables. The new "incremental update" functionality keeps these copy tables in sync by employing replication technology that monitors the DB2 transaction log and asynchronously applies the changes in micro-batches to IDAA. This enables near real-time analytics over online data, effectively marrying traditionally separated OLTP and data warehouse environments. With IDAA, reports can access data that is constantly refreshed in contrast to traditional warehouses that are updated on a daily or even weekly basis. Without any changes to the applications and without the need to introduce cross-system ETL flows, an existing OLTP environment can be used for reporting purposes as well. In this demo, we present a near real-time reporting application modeled on an industry benchmark (TPC-DS), but with a constantly changing set of tables with over 800 million rows that is running on DB2 for z/OS. In a browser-based user interface, demo attendants can influence the rate of changes to the tables and observe how the reporting queries are capturing new data as it is being modified by a separately running OLTP workload generator.

1 Introduction

IBM DB2 Analytics Accelerator (IDAA) version 3[2] is an evolution of IBM Smart Analytics Optimizer (ISAO) version 1[SBS+11] which was using the BLINK in-memory query engine[RSQ+08, BBC+12]. Since version 2, the query engine and hardware used as the

[1] IBM, DB2, and z/OS are trademarks of International Business Machines Corporation in USA and/or other countries. Other company, product or service names may be trademarks, or service marks of others. All trademarks are copyright of their respective owners.

[2] http://www-01.ibm.com/software/data/db2/zos/analytics-accelerator/

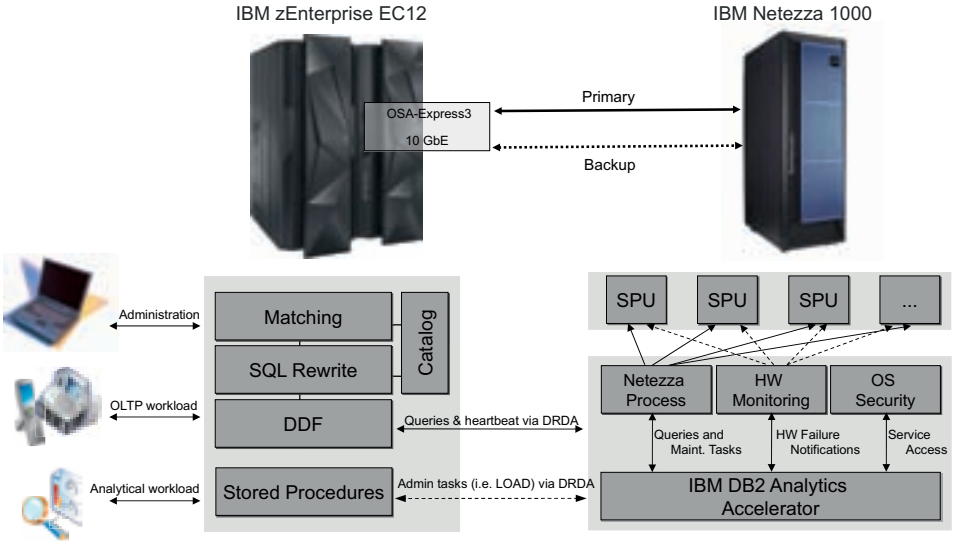


Figure 1: IDAA components and interaction with System z

basis for IDAA is Netezza 1000 (also known as "TwinFin")[3]. Figure 1 gives an overview of the involved components: IDAA is an appliance add-on to DB2 for z/OS running on an IBM zEnterprise 196 or EC12 mainframe, it comes in form of a purposely-built software and hardware combination that is attached to the mainframe via redundant 10GB fiber-channel connections to allow DB2 for z/OS to transparently offload scan-intensive queries. The acceleration factor compared to DB2 for z/OS standalone for such kind of queries can be up to 2000, because IDAA processes table scans on all of its disks in parallel, leveraging FPGAs to apply decompression, projection and restriction operations. Pushing these operations down to the level of the disk dramatically decreases the overall size of data required to be shipped over the internal network fabric and between the main parallel processing units (called SPU in Figure 1) that process joins, aggregations or complex expressions.

It is very important to note that nothing has to be changed on existing applications using DB2 for z/OS in order to get these benefits: they are connecting to DB2 for z/OS and are using the DB2 SQL syntax. In fact, there is no indication to an application if a query was processed by "mainline" DB2 or by IDAA. Based on how many fetch operations are estimated for a given query and how well existing indices match the predicates of that query, DB2 decides whether to process the query by itself or if it should be offloaded to IDAA. Deep integration into existing DB2 components ensures that only minimal training is required to operate IDAA with DB2 for z/OS as compared to DB2 for z/OS standalone. DB2 for z/OS is the owner of the data; data maintenance, backup and monitoring procedures do not change. Because IDAA operates on copies of the tables in DB2 for z/OS, whenever

[3] http://www-01.ibm.com/software/data/netezza/1000/

there are changes to these tables, the changes need also to be reflected in IDAA. Currently, this is done by running DB2 UNLOAD utilities to either refresh an entire table in IDAA or by refreshing the changed partitions of that table.

The vision of IDAA is to allow reporting over ”online” data, providing a solution that combines the features of traditionally separated OLTP and reporting systems that are connected by ETL jobs. For such scenarios, the granularity of the aforementioned refresh mechanisms is too coarse: if the total size of the changes is very low but the changes itself are spread over multiple partitions (or the tables itself are not partitioned), re-loading the majority of partitions or an entire table causes a lot of unnecessary work. Also, the practical minimum latency that can be achieved by running the refresh procedures is in the range of hours, it is impractical to run the UNLOAD-based refresh procedures every hour or even less than that. IDAA version 3 introduces a feature called ”Incremental Update” that offers a mechanism to refresh the copy tables in IDAA by asynchronously monitoring the DB2 transaction log for changes. The changes are staged in memory and are transferred over the network to IDAA once their associated unit of work has been committed. For efficiency reasons, IDAA applies the changes it receives in ”micro batches” that typically contain about 60 seconds of data from committed transactions sent from the log capture process on DB2 for z/OS. Multiple changes to a row within a micro batch are consolidated to a single change during the 60 seconds collection phase.

2 Demo Scenario

The demo is running on an IBM zEnterprise 196 mainframe running DB2 for z/OS which is connected to IDAA on a Netezza 1000-3 hardware. The IDAA hardware being used is a quarter-rack system that internally uses 24 1TB disks, 3 IBM HS-22 blades with 2 Intel Westmere 2.4Ghz 4-core CPUs and 3 FPGA daughter-boards on the Intel-blades with 2 4-core Xilinx FPGAs each. Communications in this FPGA-x86 cluster are coordinated using two IBM 3650 M3 hosts with 2 Intel Nehalem 2.4 Ghz CPUs each. The demo workload is based on TPC-DS[4] with the fact table (STORE_SALES) at a size of about 800.000.000 rows. Figure 2 depicts the demo scenario: a web interface allows users to run TPC-DS-based reporting queries and view the results as the queries are being processed. A second option allows users to start an OLTP workload generator that modifies the data by running high-frequent insert transactions on the tables. While the OLTP workload is running, users can again submit the reporting queries and observe that the reports are constantly changing as they are picking up the new data from the workload generator. The demo also provides status information about the total number changes that have been captured, the total number of changes that have been applied and an estimate of the current observable latency of the report, i.e. how far back in seconds is the current snapshot of the data on IDAA compared the state of the tables on DB2.

We will also provide a second demo running on this system that focuses on the capability of the DB2 for z/OS optimizer heuristic to distinguish between an analytical query that would

[4] http://www.tpc.org/tpcds/default.asp

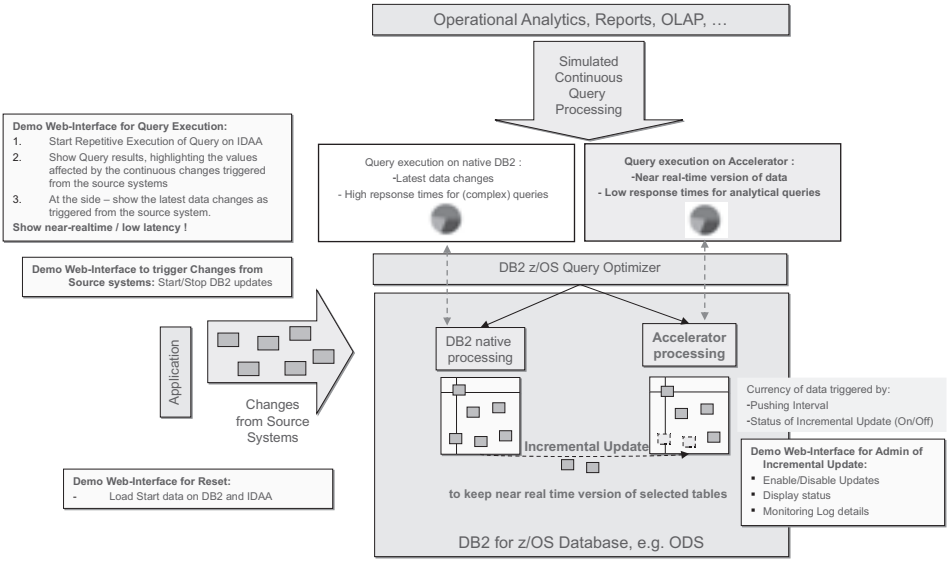


Figure 2: IDAA "Incremental Update" Demo Scenario

benefit when being offloaded to IDAA and transactional queries that run best on DB2 for z/OS. The workload generator in this demo simulates a real world business analytics workload with a mix of transactional and complex analytical queries (coming e.g. from IBM Cognos BI[5]) that are triggered by a number of concurrent users. From the moment IDAA is enabled on the system, demo attendants will notice a drastic decrease in response time for the complex analytical queries and a major increase in throughput of the overall system.

References

[BBC+12] R. Barber, P. Bendel, M. Czech, O. Draese, F. Ho, N. Hrle, S. Idreos, M.S. Kim, O. Koeth, and J.G. Lee. Business Analytics in (a) Blink. *IEEE Data Engineering Bulletin*, 35(1):9–14, 2012.

[RSQ+08] V. Raman, G. Swart, L. Qiao, F. Reiss, V. Dialani, D. Kossmann, I. Narang, and R. Sidle. Constant-time query processing. In *IEEE 24th International Conference on Data Engineering (ICDE)*, pages 60–69, 2008.

[SBS+11] K. Stolze, F. Beier, K.U. Sattler, S. Sprenger, C.C. Grolimund, and M. Czech. Architecture of a Highly Scalable Data Warehouse Appliance Integrated to Mainframe Database Systems. *Database Systems for Business, Technology, and the Web (BTW)*, 2011.

[5] http://www-01.ibm.com/software/analytics/cognos/

PythiaSearch - Interaktives, Multimodales Multimedia-Retrieval

David Zellhöfer, Thomas Böttcher, Maria Bertram, Christoph Schmidt,
Claudius Tillmann, Markus Uhlig, Marcel Zierenberg, Ingo Schmitt

Brandenburgische Technische Universität
Walther-Pauer-Str. 1, 03046 Cottbus
david.zellhoefer|tboettcher@tu-cottbus.de

Abstract: *PythiaSearch* ist ein interaktives Multimedia-Retrieval-System. Es vereint verschiedene Suchstrategien, diverse Visualisierungen und erlaubt eine Personalisierung der Retrieval-Ergebnisse mittels eines Präferenz-basierten Relevance Feedbacks. Das System nutzt die probabilistische Anfragesprache *CQQL* und erlaubt eine multimodale Anfragedefinition basierend auf Bildern, Texten oder Metadaten.

1 Motivation

Multimodale Retrievalsysteme (MIRS) sind häufig nur beschränkt anpassbar. Meistens können nur Gewichtungen von fest vorgegebenen Features verschoben werden, um die Anfrage an das Informationsbedürfnis des Nutzers anzupassen. *PythiaSearch* stellt einen adaptiveren Ansatz dar, welcher auf der probabilistischen, logikbasierten Anfragesprache CQQL [Sch08] basiert. Die Stärke des Systems liegt dabei vor allem in der Kombinationen von ähnlichkeitsbasierten und booleschen Anfragebedingungen, deren Gewichtung mittels Präferenzen angepasst werden kann. Hierdurch wird insbesondere Experten ein personalisierbares Werkzeug geboten, welches auf die volle Mächtigkeit einer logikbasierten Anfragesprache zurückgreifen kann.

2 Schnittstelle und Interaktion

Die graphischen Schnittstelle (GUI) unterstützt verschiedene Suchstrategien (gerichtet und explorativ), die während typischen Suchprozessen zu beobachten sind [RMMH00]. Dabei wird durchgängig auf die Anfragesprache CQQL zurückgegriffen, welche das kognitive Retrievalmodell der Polyrepräsentation [Ing96] umsetzt. Hierdurch wird es möglich, die GUI und die Anfrageverarbeitung ohne konzeptionelle Brüche umzusetzen [Zel12b]. In dieser Arbeit soll die graphische Oberfläche vorgestellt und unterschiedlichen Suchstrategien, Ergebnisvisualisierungen sowie dazugehörige Personalisierungsmöglichkeiten in Erweiterung von [ZBB+12] erläutert werden. Gemäß der Prinzipien der nutzerzentrierten Softwareentwicklung wurden die Anforderungen an die Software in Kooperation mit po-

tentiellen Nutzer der Medien- und Marktforschungsbranche (z.B. Bertelsmann, Deutsche Telekom oder TNS Infratest) im Rahmen von drei Workshops in 2011 und 2012 erhoben. Die Leistungsfähigkeit konnte in einer Nutzungsstudien [Zel12a] gezeigt werden.

Aufbau der grafischen Benutzeroberfäche Der vorgestellte Prototyp ist für die gängigen Betriebsysteme Mac OS X, Windows sowie Linux verfügbar und ermöglicht dem Nutzer die direkte Interaktion mit den visualisierten Dokumenten (z.B. Bildern, PDFs, etc.). Im Folgenden sollen zunächst die Grundelemente der GUI (siehe Abbildung 1) beschrieben werden.

Abbildung 1: Aufbau der GUI (Mac OS X)

1. Das Eingabe-Feld dient zur Vergabe von Suchwörtern. Diese können z.B. mittels boolescher Operatoren verbunden werden.

2. Das multimediale Eingabefenster ermöglicht es dem Anwender ein oder mehrere QBE-Dokumente (Query By Example; z.B. ein Bild oder PDF) zu wählen.

3. Die Steuerung für die Suche kann genutzt werden, um eine neue Suche zu starten sowie die Anzahl der angezeigten Dokumente zu konfigurieren.

4. Der Suchverlauf erlaubt es dem Anwender, bereits durchgeführte Suchen, Lernschritte etc. wieder aufzurufen bzw. wieder rückgängig zu machen.

5. In der Ergebnissicht werden alle relevanten Dokumente dargestellt. Diese Sicht erlaubt dem Nutzer eine direkte Interaktion mit den visualisierten Dokumenten. Hierbei können diese verschoben als auch gestapelt um z. B. für die entstandene Gruppe Annotationen zu vergeben.

6. Mithilfe dieses Auswahlmenüs können verschiedene Visualisierungen der Ergebnissicht (5) eingestellt werden. In Abbildung 1 (zentrales Fenster) ist die Matrix-Ansicht dargestellt, welche die Elemente nach absteigender Relevanz sortiert. Weitere Details zu den Visualisierungen finden sich in Abschnitt 2.

7. Für eine Personalisierung der Suchergebnisse können Dokumente aus der Visualisierung in das Präferenz-Fenster (mithilfe von Drag & Drop oder dem Kontextmenü) gezogen werden. Hierbei wird eine Halbordnung definiert, welche die Relevanz der Objekte in Bezug auf die Anfrage beschreibt. Das QBE-Dokument befindet sich hierbei im Zentrum des Fensters und die Relevanz der Dokumente nimmt mit steigender Entfernung zum Zentrum ab [Zel12b].

8. Die facettierte Suche erlaubt es, einen Filter auf die bisherigen Suchergebnisse zu setzen. Hierbei spiegelt eine Facette eine boolesche Bedingung wieder, welche direkt in eine gewichtete CQQL-Anfrage transformiert wird. In der Ergebnisliste werden bei Aktivierung einer Facette nur Dokumente gezeigt, welche die definierte Bedingung erfüllen (z. B. die Abwesenheit von Personen auf einem Foto).

9. In dieser Ansicht werden repräsentative Bilder der durchsuchten Datenbank angezeigt, um diese explorativ erschließen zu können. Bei Auswahl eines der Bilder werden ähnliche Bilder in der jeweils ausgewählten Visualisierung angezeigt, um Browsing zu ermöglichen.

Unterstützte Suchstrategien *PythiaSearch* unterstützt gängige Suchstrategien. Hierbei wird eine *gerichtete Suche*, bei der der Nutzer bereits sein Informationswunsch kennt, als auch eine *explorative Suche* ermöglicht. Beide Suchstrategien sind kombinierbar, wobei der Wechsel der Suchstrategie jederzeit aus einer beliebigen Ansicht vorgenommen werden kann. Für die *gerichtete Suche* stehen dem Nutzer zwei Eingabefelder zur Verfügung. Mit dem Texteingabefeld können einfache Keyword-basierte Suchanfragen definiert werden. Über das multimediale Eingabefenster können sowohl Bilder als auch PDF-Dokumente zur Anfragedefinition genutzt werden. Beide Eingabe können kombiniert werden, so dass ein Informationswunsch auf multimodaler Ebene definiert werden kann. Enthält ein PDF-Dokument neben Text auch Bilder so wird die gesamte Struktur zur Ähnlichkeitsberechnung herangezogen (sowohl Bilder als auch Text in Abhängigkeit der Struktur des Dokuments). Für jede vorhandene Dokumentenrepräsentation auf jeder Ebene eines jeden Anfragedokuments wird eine Ähnlichkeitsberechnung durchgeführt. Über einen speziellen Operator (z.B. auf CQQL basierend) werden die Einzelähnlichkeiten miteinander aggregiert.
Für Anwender, die ihren Informationswunsch nicht explizit definieren können wurde das *explorative Browsing* integriert. Es ermöglicht einen Überblick über einzelne Dokumente innerhalb der verwendeten Datenbank zu erhalten (siehe Abbildung 1 (9)). Die Datenbank wird dabei mit einem Cluster-basierten Ansatz aufgearbeitet, so dass dem Anwender zunächst nur ein Element einer Klasse präsentiert wird. Über die einzelnen Visualisierungsformen können dann die Objekte innerhalb einer Klasse visualisiert werden. Die während der Exploration gefundenen Dokumente können im Anschluss beispielsweise als QBE-Dokument genutzt werden, um eine gerichtete Suche zu starten.

Ergebnisvisualisierung

Ausgehend von den verschiedenen Suchstrategien können aktuell drei unterschiedliche Visualisierungen gewählt werden, um die Ergebnisdokumente zu präsentieren.

Matrix In der Standardansicht werden Dokumente durch die berechneten Ähnlichkeitswerte sortiert und in einem Raster angezeigt. Diese Ansicht ist ideal, um das Ranking einer Suchanfrage zu betrachten und wird durch eine gerichtete Suchanfrage generiert.

SOM Die SOM-Ansicht ist eine selbst-organisierende Karte [Koh95] in der alle Objekte durch die gewählte Eigenschaft (wie z. B. Farbe, Textur oder einer CQQL-Anfrage) automatisch sortiert werden. Die SOM ermöglicht es, einen Überblick über die verschiedenen

Ausprägungen aller Medienelemente zu erhalten, hierbei ist es möglich sich auf eine Ausprägung zu fixieren (z.B. eine konkrete Farbe) und in diesen Bereich hinein zu navigieren. Dokumente, die sich innerhalb der SOM nahe beieinander liegen besitzen eine ähnliche Charakteristik in der gewählten Eigenschaft.

Cluster Die Ergebnisse einer Suche können durch das Clustern weiter verarbeitet werden, um neue Eigenschaften aufzuzeigen oder ähnliche Elemente zu gruppieren. Im Gegensatz zur SOM erfolgt die Trennung hier hart, d.h. Die Ergebnisse gehören zu genau einem Cluster. Ein Cluster beinhaltet Dokumente, die bezüglich einer CQQL-Formel möglichst ähnlich zueinander sind, während Elemente unterschiedlicher Cluster möglichst unähnlich von einander sind. Die gewählte CQQL-Formel kann ein Merkmal aber auch eine beliebige (logische) Kombination von Merkmalen (z. B. GPS-Koordinaten und Farbe) verwenden, wodurch die definierte Eigenschaft für das Clustern sehr flexibel ist und manuell auf die Bedürfnisse des Nutzers angepasst werden kann.

3 Demonstration

Innerhalb der Demonstration soll der typische Ablauf eines Retrieval-Prozesses gezeigt werden. Hierbei wird die Extraktion von Features, die Auswahl eines Anfragedokumentes und die verschiedenen Visualisierungen (vgl. Abschnitt 2) vorgestellt. Desweiteren wird die Verfeinerung eines Informationswunsches durch die Vergabe von Präferenzen durchgeführt und somit eine Personalisierung der Ergebnisse erzielt. Im weiteren Verlauf wird durch eine kombinierte Anfrageformulierung eine multimodale Suche demonstriert.

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 03FO3072 gefördert.

Literatur

[Ing96] Peter Ingwersen. Cognitive perspectives of information retrieval interaction: elements of a cognitive IR theory. *Journal of Documentation*, 52:3–50, 1996.

[Koh95] Teuvo Kohonen. *Self-organizing maps*, Jgg. 30 of *Springer series in information sciences*. Springer, Berlin, 1995.

[RMMH00] Harald Reiterer, Gabriela Mußler, M. Thomas Mann und Siegfried Handschuh. INSYDER - an information assistant for business intelligence. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '00, Seiten 112–119. ACM, 2000.

[Sch08] Ingo Schmitt. QQL: A DB&IR Query Language. *The VLDB Journal*, 17(1):39–56, 2008.

[ZBB+12] David Zellhöfer, Maria Bertram, Thomas Böttcher, Christoph Schmidt, Claudius Tillmann und Ingo Schmitt. PythiaSearch – A Multiple Search Strategy-supportive Multimedia Retrieval System. In *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*, ICMR '12, Seite to appear. ACM, 2012.

[Zel12a] David Zellhöfer. On the Usability of PythiaSearch. Bericht 9, Brandenburg University of Technology, Cottbus, 2012.

[Zel12b] David Zellhöfer. A permeable expert search strategy approach to multimodal retrieval. In *Proceedings of the 4th Information Interaction in Context Symposium*, IIIX '12, Seiten 62–71, New York, NY, USA, 2012. ACM.

ScyPer: A Hybrid OLTP&OLAP Distributed Main Memory Database System for Scalable Real-Time Analytics

Tobias Mühlbauer*, Wolf Rödiger, Angelika Reiser, Alfons Kemper, Thomas Neumann

Technische Universität München · Boltzmannstraße 3 · D-85748 Garching
{muehlbau, roediger, reiser, kemper, neumann}@in.tum.de

Abstract: ScyPer is an abbreviation for *Sc*aled-out H*yPer*, a version of the HyPer main memory hybrid OLTP&OLAP database system that horizontally scales out on shared-nothing commodity hardware. Our demo shows that ScyPer a) achieves a near-linear scale-out of OLAP query throughput with the number of active nodes, b) sustains a constant OLTP throughput, c) is resilient to node failures, and d) offers real-time analytical capabilities through market-leading query response times and periodically forked TX-consistent virtual memory snapshots with sub-second lifetime durations.

1 Motivation

Database systems face two distinct workloads: online transactional processing (OLTP) and online analytical processing (OLAP). These two workloads are nowadays mostly processed in separate systems, a transactional one and a data warehouse for OLAP, which is periodically updated by a so-called extract-transform-load (ETL) phase. However, ETL interferes with mission-critical OLTP performance and is thus often carried out once every night which inevitably leads to a problem of *data staleness*. Industry leaders such as Hasso Plattner of SAP argue that this data staleness is inappropriate for *real-time business analytics* [PZ11]. New hybrid OLTP&OLAP main memory database systems such as SAP's HANA or HyPer [KN11] achieve best-of-breed transaction processing throughput and OLAP query response times in one system in parallel on the same database state.

Back-of-the-envelope calculations show that these main memory database systems can, on the one hand, handle the OLTP workload and transactional data volumes even for the largest commercial enterprises. E.g., for Amazon we estimate a yearly volume of 54GB for the order lines, the dominating repository in such a sales application [KN11]. Today, servers with 1TB of main memory are available for less than $40,000 and database systems such as HyPer can process more than 100,000 transactions per second on such machines — enough to process a human generated OLTP workload even during busy hours. Furthermore, limited main memory is not a restriction as data can be divided into hot and cold data where the latter can be compacted and swapped to disk [FKN12]. Thus, we conjecture that for the vast majority of use cases OLTP can be managed by a single database server.

Analytical queries on the other hand can be quite complex and computationally expensive. Thus, to maintain performance under high OLAP load, the database needs to be scaled

*Tobias Mühlbauer is a recipient of the Google Europe Fellowship in Structured Data Analysis, and this research is supported in part by this Google Fellowship.

out. A scale-out also addresses the need for high availability in the sense that the database can fail-over to an active replica on the fly. In this work we demonstrate ScyPer, a version of the HyPer database system that horizontally scales out on commodity hardware.

2 The HyPer Main Memory Database System

HyPer [KN11] is a hybrid OLTP&OLAP relational main memory database system in which OLAP queries are processed on arbitrarily recent virtual memory (VM) snapshots of the transactional database. This isolation prevents long-running OLAP queries from interfering with mission-critical OLTP processing for which the ACID properties are guaranteed. Unlike traditional database systems, HyPer achieves market-leading performance (compared to state-of-the-art main memory OLTP or OLAP database systems) for both, OLTP and OLAP workloads, operating simultaneously on the same database. HyPer's performance is, among others, due to the following key characteristics: (i) HyPer relies on in-memory data management which eliminates the ballast caused by DBMS-controlled page structures and buffer management. (ii) OLAP processing is separated from mission-critical OLTP processing by forking transaction (TX)-consistent VM snapshots using the POSIX system call `fork()`. The hardware- and operating system-supported "copy on update" mechanism then preserves the consistency of the snapshot by copying pages only if a page is modified. No concurrency control mechanism — other than a short synchronization between the fork and OLTP processing — is needed to separate the two workload classes. (iii) Transactions and queries are specified in SQL or in a scripting language and are efficiently compiled into LLVM code [Neu11]. (iv) As in VoltDB, parallel transactions are separated via lock-free admission control. Thus, non-conflicting transactions can be simultaneously processed in multiple threads (working on different partitions (Ptn)) without the need of concurrency control. (v) HyPer relies on logical logging where, in essence, the invocation parameters of transaction procedures are logged via a high-speed network.

3 ScyPer Architecture

A ScyPer cluster consists of one primary and several secondary nodes where each node runs a ScyPer instance. The architecture of the system is shown in Figure 1.

The primary node is the entry point of the system for transactions as well as analytical queries. The OLTP workload is processed in an OLTP process and the logical redo log is multicasted to all secondary nodes using pragmatic general multicast (PGM). The PGM protocol is scalable and provides the reliable delivery of packets in guaranteed ordering from a single sender to multiple receivers. The redo log, in essence, contains the invocation parameters of transaction procedures together with log sequence numbers. The primary node is not restricted to but usually uses a row-store data layout which is a suitable choice for OLTP processing and keeps indexes that support efficient transaction processing. It can, but does not necessarily have to have TX-consistent snapshots on which it can process OLAP queries or write TX-consistent backups out to a storage node. A coordinator process on the primary node receives incoming OLAP queries and load balances these queries among the secondary nodes.

Secondary nodes receive the multicasted logical redo log from the primary node and rerun each of the transactions. As a large portion of a usual OLTP workload is read-only (i.e.,

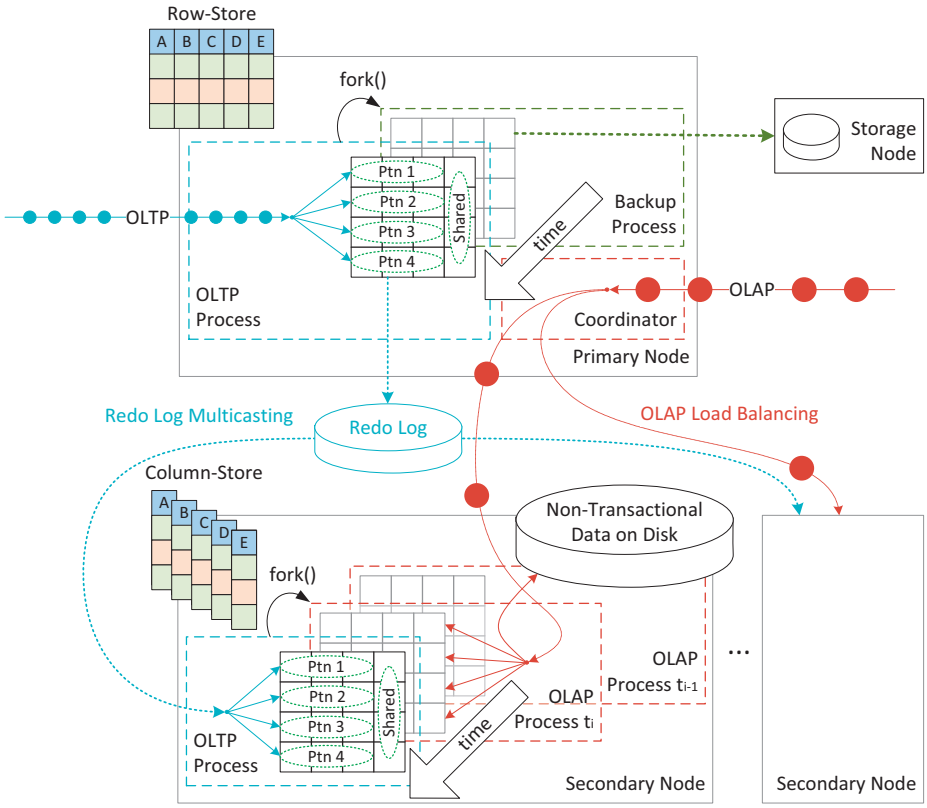


Figure 1: ScyPer architecture

no redo is necessary), secondary nodes usually face less OLTP work than primary nodes. These additional resources are used to process incoming OLAP queries or create backups on forked TX-consistent snapshots. Furthermore, indexes for efficient analytical query processing can be created. Secondary nodes can either store data in a row-, column-, or hybrid row- and column-store data format. Additionally, these nodes can have non-transactional data on disk which can be queried by OLAP queries.

Clients are not restricted to stored transactions and may add new transaction definitions. Internally, such a request is converted to a system-internal transaction which treats the transaction definition as an input of a system-internal *compile* transaction. Similarly, cross-node consistent snapshots can be created where the snapshots have the common logical time of the log sequence number of the system-internal transaction. On such snapshots, queries can be parallelized over several nodes.

All nodes multicast heartbeats such that node failures can be detected. Secondary nodes can fail arbitrarily as we assume that clients re-send OLAP query requests after a timeout. Alternatively OLAP requests can be replicated in the system so processing can be resumed if a secondary node fails. In case of a primary node failure, the secondary nodes elect a new primary using a PAXOS-based protocol. The latest acknowledged log sequence number when failing over is determined by majority consensus.

Figure 2: ScyPer CLI Client

Figure 3: ScyPer Web Client

4 Demonstration Setup

To demonstrate the capabilities of ScyPer, we run (i) a benchmark demo and (ii) an interactive demo where we simulate the sales order processing system (order entry, payment, delivery) of a merchandising company. The schema and a mixed OLTP and OLAP workload for this system are based on the CH-benCHmark [C$^+$11], a "merge" of the standard TPC-C (OLTP) and TPC-H (OLAP) benchmarks. This scenario resembles the core business of a commercial merchandiser like Amazon.

The demos run on a ScyPer cluster composed of a varying number of commodity machines (Intel Core i7-3770 CPU, 32 GB dual-channel DDR3-1600 DRAM, Linux 3.5 64bit) and located at TUM. The primary node of the cluster solely processes the OLTP workload; secondary nodes process the logical redo log and periodically fork TX-consistent snapshots with a varying lifetime for OLAP processing. During both demos we simulate node failures to demonstrate ScyPer's high availability capabilities.

(i) In the *benchmark demo* we run the CH-benCHmark with a varying number of nodes. Performance metrics such as the linearly-scalable OLAP and the constant OLTP throughput as well as query response times are visualized on a dashboard.

(ii) In the *interactive demo* we only run the OLTP workload of the CH-benCHmark while participants can simultaneously enter analytical SQL-92 queries via a command line (see Figure 2) or a web (see Figure 3) interface that presents results "at the keystroke".

References

[C$^+$11] R. Cole et al. The mixed workload CH-benCHmark. In *DBTest*, pages 8:1–8:6, 2011.

[FKN12] F. Funke, A. Kemper, and T. Neumann. Compacting Transactional Data in Hybrid OLTP&OLAP Databases. *PVLDB*, 5(11):1424–1435, 2012.

[KN11] A. Kemper and T. Neumann. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *ICDE*, pages 195–206, 2011.

[Neu11] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9):539–550, 2011.

[PZ11] H. Plattner and A. Zeier. *In-Memory Data Management: An Inflection Point for Enterprise Applications*. Springer, 2011.

FlexY: Flexible, datengetriebene Prozessmodelle mit *YAWL*

Sebastian Schick, Holger Meyer, Andreas Heuer

Universität Rostock, Institut für Informatik
Lehrstuhl für Datenbank- und Informationssysteme
{schick, hme, heuer} @informatik.uni-rostock.de

Abstract: Wir präsentieren mit *FlexY* ein Workflow-Management-System (WFMS) zur flexiblen, datengetriebenen Umsetzung von Prozessen mit *YAWL*. Durch den Einsatz des Systems wird die datenabhängige Steuerung und Kontrolle von Arbeitsabläufen in Informationssystemen zur Laufzeit ermöglicht. Mit einem flexiblen Prozessmodell wird die gemeinsame Modellierung von Kontroll- und Datenflussabhängigkeiten erreicht. Änderungsoperationen die den Zustand der Dokumente und deren Struktur beschreiben, können so zu bestimmten Zeitpunkten im Prozessverlauf abgefragt und direkt an die Änderung der Prozessinstanz geknüpft werden.

1 Motivation

Moderne Informationssysteme werden heute z. B. für die Publikation von digitalen, wissenschaftlichen Dokumenten eingesetzt. Zunehmend komplexere Teilaufgaben im Publikationsprozess erfordern den Einsatz von Workflow-Management-Systemen (WFMS), die den Benutzer bei der Abarbeitung der anfallenden Aufgaben unterstützen. Der Prozessverlauf ist dabei maßgeblich abhängig von den Dokumenten und dem Prozesszustand. So können innerhalb der Prozessmodelle Bereiche identifiziert werden, die die Bearbeitung von Dokumentfragmenten beschreiben. Ein Beispiel stellt die Extraktion von medienspezifischen Merkmalen als wichtiger Bestandteil der inhaltsbasierten Suche dar. Jeder Dokumenttyp (Text, Bild und Video) wird durch entsprechende Prozessschritte bearbeitet. Ändern Dokumente häufig Struktur und Inhalt, kommen Systeme, die unflexible Prozessmodelle verwenden, schnell an ihre Grenzen. Es müssen komplexe, redundante Prozessmodelle modelliert werden, um die Verarbeitung der unterschiedlichen Dokumentvarianten zu unterstützen. Die Fehleranfälligkeit und Probleme bei der Wartung solcher Prozessmodelle erhöhen sich damit in Abhängigkeit der zu verarbeitenden Dokumentmodelle.

In dieser Arbeit wird ein WFMS vorgestellt, das die datenabhängige Steuerung und Kontrolle von Arbeitsabläufen in Informationssystemen zur Laufzeit ermöglicht [Sch12]. Der Ansatz beschreibt die Beziehungen zwischen Prozessmodellen und Dokumenten, um Prozessinstanzen zur Laufzeit flexibel auf veränderte Dokumentstrukturen und -inhalte anzupassen. Er ermöglicht die Modellierung der Kontrollfluss-Perspektive zusammen mit Dokumenten, deren Strukturen und Inhalte. Hierfür wird die Integration unterschiedlicher, externer Datenquellen durch einheitliche Schnittstellen und Modellierungskonzepte umgesetzt. So können Datenquellen im Prozessmodell beschrieben und den Aktivitäten zur

Laufzeit bereitgestellt werden. Weiterhin wird eine Unterteilung in anwendungsabhängige Prozessbestandteile (den Basisprozess) und dokumentabhängige Prozessbestandteile vorgenommen. Die Anpassungen der dokumentabhängigen Prozessbestandteile werden in Abhängigkeit von Dokumentänderungen in der externen Datenquellen umgesetzt.

Verfahren für die Integration externer Datenquellen können allgemein in zwei Kategorien unterteilt werden. Ansätze wie PHILharmonicFlows [KR11] versuchen die strikte Trennung zwischen Datenfluss und Kontrollfluss aufzuheben, indem komplexe Abhängigkeiten zwischen Datenobjekten und Aktivitäten beschrieben werden. Eine bessere Anbindung externer Datenquellen wird durch Arbeiten wie SIMPL [RRS+11] erreicht. In dem vorgestellten System wird der Ansatz aus [SMH11a] für die einheitliche Integration externer Datenquellen umgesetzt. Um eine flexible, datengetriebene Anpassung von Prozessmodellen zu erreichen eignen sich die Konzepte der Unterspezifikation und Modifikation besonders. Im vorgestellten System wird das Konzept des Late Modeling aus [SMH11b] genutzt. Es basiert auf [SSO01], wo Sadiq et al. Platzhalter einführen, um mittels Late Modeling Prozessinstanzen anzupassen. In [RRKD05] werden Änderungsoperationen eingeführt, die die Möglichkeit bieten Bereiche in Prozessinstanzen zu verändern.

2 Architektur

Eine flexible, datengetriebene Abarbeitung der Prozessmodelle wird erreicht, indem eine Trennung von anwendungsspezifischen und dokumentspezifischen Prozessbestandteilen vorgenommen wird. Observer-Aktivitäten (s. Abb. 1 Aktivität t_2^O) ermitteln den Dokum-

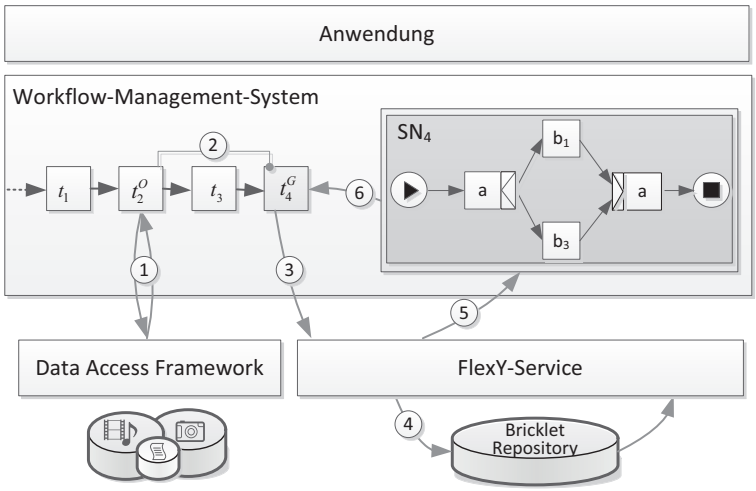


Abbildung 1: Modell für flexible, datengetriebene Prozesse

entzustand an beliebigen Stellen im Prozessmodell. Ein Observer nutzt das DAF (s. Abb. 1: Schritt 1), um den aktuellen Zustand der verwendeten externen Dokumente zu bestimmen.

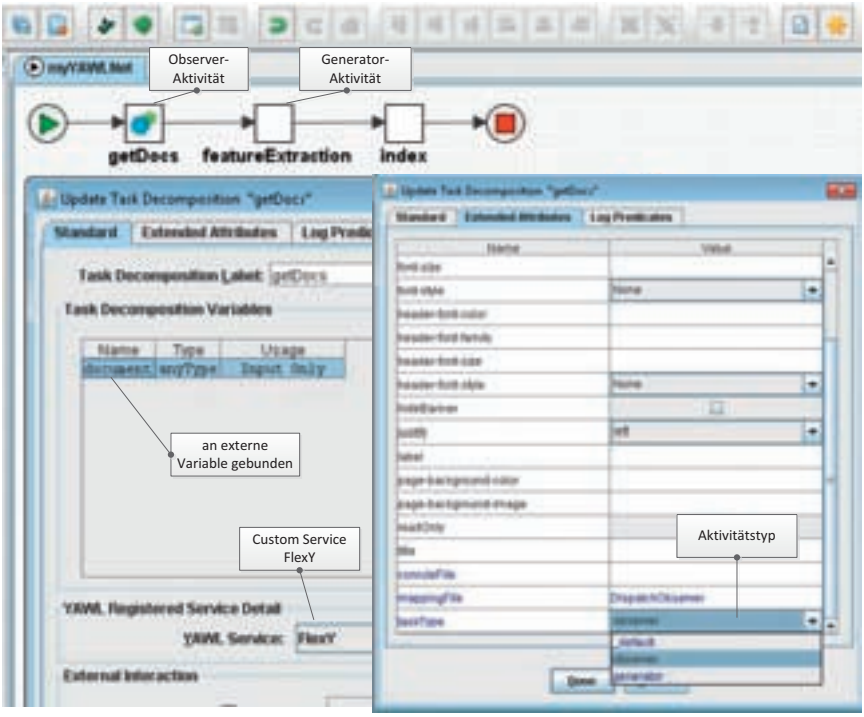


Abbildung 2: Screenshots von *FlexY*

Oberserver ermöglichen es so, neben dem Zustand auch Zustandsübergänge der Dokumente zu ermitteln. Die anwendungsabhängigen Prozessbestandteile werden durch einen Basisprozess modelliert (s. Abb. 1: Aktivitäten $t_1 \ldots t_4$). In den Basisprozess können dokumentabhängige Prozessbestandteile, die durch Bricklets[1] beschrieben sind, eingefügt werden. Bricklets dürfen im Basisprozess nur in dafür vorgesehene Bereiche eingefügt werden, weil der Basisprozess nicht verändert werden darf. Diese Bereiche werden durch Generator-Aktivitäten (s. Abb. 1 Aktivität t_4^G) beschrieben. Ein Generator generiert zur Laufzeit ein Subnetz aus den Bricklets und führt es aus. Welche Bricklets für die Generierung genutzt werden, wird im Vorfeld durch verschiedene Observer bestimmt (s. Abb. 1: Schritt 2). Wird ein Generator aktiviert, ruft er den *FlexY*-Service auf (s. Abb. 1: Schritt 3). Dort wird aus der Menge der aktivierten Bricklets (s. Abb. 1: Schritt 4) ein Prozessmodell generiert (s. Abb. 1: Schritt 5). Dieses Prozessmodell wird dann umgehend vom Generator ausgeführt (s. Abb. 1: Schritt 6).

Für die Umsetzung des Systems wird das WFMS *YAWL* [vdAtH05] erweitert, sodass Prozessinstanzen nach dem Prinzip des Late Modeling angepasst werden können. Die Erweiterung von *YAWL* wird durch zwei neue Services erreicht (s. Abbildung 1). Das Data Access Framework (DAF) wird genutzt, um externe Datenquellen einheitlich zu integrieren. Über eine Schnittstelle für Aktivitäten können damit verschiedenen Datenquellen in der

[1] Bricklets sind Prozessbausteine, die durch ein vollständiges *YAWL*-Netz beschrieben sind.

Kontrollflussperspektive bereitgestellt werden. Mit dem *FlexY*-Service kann der Zustand von Dokumenten aus externen Datenquellen ermittelt werden, um in Abhängigkeit davon die Prozessinstanzen anzupassen. Die Übertragung auf BPMN 2.0 wird aktuell untersucht.

3 Demonstration

Der Prototyp ist an das WFMS *YAWL* gekoppelt, dass für die Instanziierung und Ausführung der Prozessmodelle genutzt wird. Es ist möglich, *FlexY*-Prozessmodelle zu modellieren und im Anschluss mit dem Prototyp auszuführen. Während der Ausführung der Prozessinstanzen wird gezeigt, welche Auswirkung unterschiedliche Dokumente auf die Abarbeitung der Prozessinstanzen haben. Zur Verdeutlichung werden dem Besucher des Demonstrationsstandes z. B. die generierten Subprozesse gezeigt. Der Besucher wird außerdem die Möglichkeit haben, die dynamische Komposition und Ausführung der Prozessmodelle durch Änderung von Parametern und Dokumenten zu beeinflussen. Abbildung 2 zeigt verschiedene Perspektiven der Beispiel-Anwendung, die für die Modellierung der Prozessmodelle verwendet werden.

Literatur

[KR11] Vera Künzle und Manfred Reichert. PHILharmonicFlows: towards a framework for object-aware process management. *Journal of Software Maintenance*, 23(4):205–244, 2011.

[RRKD05] Manfred Reichert, Stefanie Rinderle, Ulrich Kreher und Peter Dadam. Adaptive Process Management with ADEPT2. In *ICDE*, Seiten 1113–1114. IEEE, 2005.

[RRS+11] Peter Reimann, Michael Reiter, Holger Schwarz, Dimka Karastoyanova und Frank Leymann. SIMPL - A Framework for Accessing External Data in Simulation Workflows. In *BTW*, Jgg. 180 of *LNI*, Seiten 534–553. GI, 2011.

[Sch12] Sebastian Schick. *Flexible, datengetriebene Workflows für den Publikationsprozess in digitalen Bibliotheken*. Dissertation, Universität Rostock, Juli 2012.

[SMH11a] Sebastian Schick, Holger Meyer und Andreas Heuer. Enhancing Workflow Data Interaction Patterns by a Transaction Model. In *Proceedings II of the 15th East-European Conference on Advances in Databases and Information Systems, ADBIS 2011, Vienna, Austria*, Jgg. 789, Seiten 33–44. CEUR, September 2011.

[SMH11b] Sebastian Schick, Holger Meyer und Andreas Heuer. Flexible Publication Workflows Using Dynamic Dispatch. In *ICADL*, Jgg. 7008 of *Lecture Notes in Computer Science*, Seiten 257–266. Springer, 2011.

[SSO01] Shazia Sadiq, Wasim Sadiq und Maria Orlowska. Pockets of Flexibility in Workflow Specification. In *Conceptual Modeling – ER 2001*, Jgg. 2224 of *Lecture Notes in Computer Science*, Seiten 513–526. Springer Berlin / Heidelberg, 2001.

[vdAtH05] Wil M. P. van der Aalst und Arthur H. M. ter Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

Applying Stratosphere for Big Data Analytics

Marcus Leich*[1], Jochen Adamek*[2], Moritz Schubotz*[3],
Arvid Heise†[4], Astrid Rheinländer‡[5], Volker Markl*[6]
*Technische Universität Berlin, Germany †Hasso-Plattner Institute Potsdam, Germany
‡Humboldt-Universität zu Berlin, Germany
[1,2,3,6]{marcus.leich,j.adamek,schubotz,volker.markl}@tu-berlin.de
[4]arvid.heise@hpi.uni-potsdam.de, [5]rheinlae@informatik.hu-berlin.de

Abstract: Analyzing big data sets as they occur in modern business and science applications requires query languages that allow for the specification of complex data processing tasks. Moreover, these ideally declarative query specifications have to be optimized, parallelized and scheduled for processing on massively parallel data processing platforms. This paper demonstrates the application of Stratosphere to different kinds of Big Data Analytics tasks. Using examples from different application domains, we show how to formulate analytical tasks as Meteor queries and execute them with Stratosphere. These examples include data cleansing and information extraction tasks, and a correlation analysis of microblogging and stock trade volume data that we describe in detail in this paper.

1 Introduction

Analytics in numerous economic, political, and scientific sectors are focussing more and more on gaining new knowledge from web scale data. Tasks like social media analysis or market monitoring require complex processing chains that need to handle structured and unstructured data. While systems such as Hadoop are capable of handling such workloads, they are comparatively difficult to develop for. Script languages like Pig Latin and JAQL have been proposed to ease development for Hadoop and facilitate ad hoc queries. This paper demonstrates the application of Stratosphere to complex analytics tasks. We present queries for three different use cases ranging from business analytics to information extraction and information integration using Stratosphere's query language Meteor. The paper is structured in the following way: Section 2 provides an overview of Stratospheres key components. Section 3 showcases how Stratosphere can be used correlate tweets with stock trade volume data. The description includes the query, it's execution, and a customized user interface. Additionally, we outline two further tasks that involve biomedical information extraction and integration of open government data. Section 4 concludes this paper.

2 Overview of Stratosphere

This section briefly describes the key components of Stratosphere relevant for the demonstration. For a more detailed description, please refer to [WK09, BEH+10, HRL+12].

Stratosphere[BEH+10] is a massively parallel data processing system. It consists of a declarative query language **Meteor**[HRL+12], the **Pact** programming model, and **Nephele** [WK09], the execution engine. Users express their queries using the Meteor language. Here, high level operators, such as filter, are applied to (semi-)structured data sets. Meteor operators consist of Pact programs, directed acyclic graphs of second-order functions. The Pact programming model is a generalization of the MapReduce concept. In addition to the conventional *map* and *reduce*, Pact provides three supplemental, second order functions which allow for efficient implementation of cross products, equi-joins, and groupings from two sources. Pact programs are optimized and compiled into data flow graphs, which are processed in parallel by the Nephele execution engine. Therefore Stratosphere is capable of transforming complex user queries into optimized parallel execution graphs.

3 Demonstrations

3.1 Correlation of Tweets and Stock Trade Volume

Objective: The program is inspired by the work of Ruiz et al. [RHC+12] and computes the correlation of microblog posts (tweets) and stock trade volume. While the socioeconomic implications of this relationship are certainly interesting, this paper covers only the implementation of such an analysis.

The Meteor script: The first part of the program (Figure 1 top) specifies the sources for the tweet and stock volume data (line 3-4). Line 6 filters the relevant tweets. The following two blocks of code group the filtered tweets and trade volume data by week and aggregate the values in each group. The last portion of the script joins the tweet counts with the stock volume data, computes the correlation, and stores the result.

UI and Execution: Since the Meteor program is executed on a cluster, a web interface is provided to trigger the computation from remote machines. The UI (Figure 1) provides an input field for the query. After submission of the Meteor program, the server checks the syntax, builds the Sopremo operator graph, and compiles it into a Pact program. Stratosphere optimizes this Pact program, and executes it on Nephele. Figure 1 shows the optimized Pact program (bottom left) and the Nephele graph during execution (bottom right). After execution, result files, up to a certain size, can be inspected in the web interface. Suitable data types, like time series, can be visualized directly in the browser.

3.2 Further Applications

Finding relationships between drugs and genes is a fundamental task in pharmacogenetics, where differing drug responses due to genetic variations are studied. We present a query which extracts relationships between genes and drugs from the biomedical literature using text mining methods. First, the query analyzes the syntactical structure of the given texts and identifies occurrences of gene and drug names. Finally, a relation extraction component inspects all gene/drug name pairs, occurring in the same sentence, to detect relationships between genes and drugs.

We additionally demonstrate the integration of Open Government Data and other freely available data sets with Stratosphere. Specifically, we combine the publicly available spending from the US government to legal entities with information from Freebase about companies and their employees as well as persons and their relationships to find potential cases of nepotisms and other suspicious money flows. The results may be used by data journalists to start in-depth investigations on the involved entities.

4 Conclusion

In this paper we have demonstrated the application of Stratosphere to a correlation analysis of microblogging and stock trade volume data using Meteor.

Acknowledgements: This work is funded by the German Research Foundation under grant "FOR 1036: Stratosphere – Information Management on the Cloud" and by the European Commission under FP7 Project No. 296448 – "Data Supply Chains for Pools, Services and Analytics in Economics and Finance."

References

[BEH+10] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. Nephele/PACTs: A Programming Model and Execution Framework for Web-Scale Analytical Processing. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 119–130, New York, NY, USA, 2010. ACM.

[HRL+12] Arvid Heise, Astrid Rheinländer, Marcus Leich, Ulf Leser, and Felix Naumann. Meteor/Sopremo: An Extensible Query Language and Operator Model. In *Proceedings of the International Workshop on End-to-end Management of Big Data (BigData) in conjunction with VLDB 2012*, Istanbul, Turkey, 0 2012.

[RHC+12] Eduardo J Ruiz, Vagelis Hristidis, Carlos Castillo, Aristides Gionis, and Alejandro Jaimes. Correlating financial time series with micro-blogging activity. In *WSDM '12: Proceedings of the fifth ACM international conference on Web search and data mining*. ACM Request Permissions, February 2012.

[WK09] Daniel Warneke and Odej Kao. Nephele: efficient parallel data processing in the cloud. In *Proceedings of the 2nd Workshop on Many-Task Computing on Grids and Supercomputers*, MTAGS '09, pages 8:1–8:10, New York, NY, USA, 2009. ACM.

Figure 1: **Top:** Meteor's Query Submission Interface. **Bottom left:** Optimized PACT plan. **Bottom right:** Nephele execution graph.

Gesture-Based Navigation in Graph Databases – The Kevin Bacon Game –

Felix Beier Stephan Baumann Heiko Betz Stefan Hagedorn
Ilmenau University of Technology, Germany
first.last@tu-ilmenau.de

Timo Wagner
Objectivity, Inc.
twagner@objectivity.com

Abstract: Motion sensing devices like Microsoft's Kinect offer an alternative to traditional computer input devices like keyboards and mouses. Graph databases can naturally make use of gesture control as traversing graphs can easily be described by swiping or pointing gestures. In our demo we traverse the Internet Movie Database (IMDB) using a Kinect interface with the control logic in our data stream engine AnduIN. The gesture detection is done based on AnduIN's complex event processing functionality.

1 Introduction

Research has shown that more than 60% of the interpersonal information exchange are passed non-verbal. New hardware achievements like gesture detecting devices as Microsoft's Kinect [SFC+11] moved gestures as a new input alternative into the focus of research and industry. Also graph databases recently attracted interest again, not only driven by social networks, but also by applications in the fields of linked and big data. In contrary to SQL systems, they offer natural ways for user interaction, e.g., traversing nodes and edges can be intuitively expressed with hand movements. In our demonstration we show how gesture based interaction and graph databases can be combined. We address the following issues:

- Exploiting online techniques for complex event processing (CEP) implemented in our data stream engine AnduIN [KKH+11] for gesture recognition in sensor data which is provided by a motion sensing input device as the Kinect camera.
- Using this solution to build a gesture controlled interface for a graph database which enables the user to navigate through the data with natural movements.

We have previously used gesture detection to navigate an OLAP cube [HSS+12]. There we used gestures to execute for example rotate, drill down or role up operations. A link to a video of this demo is provided in the references.

The rest of this paper is structured according to our software architecture as shown in Figure 1. First, we provide a short overview of the Kinect and its drivers. Next, we introduce the InfiniteGraph graph database, followed by an introduction of the main gestures used for interaction. The paper closes with a description of the demonstration.

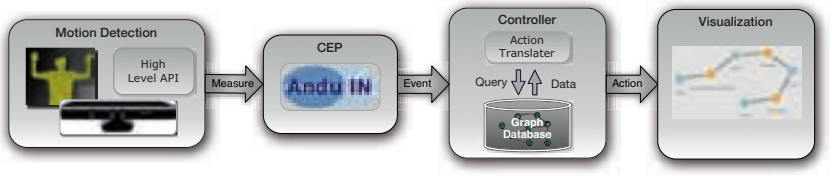


Figure 1: Software architecture. Motion sensors deliver a continuous stream of body positions. AnduIN's complex event processing (CEP) engine is used for detecting gestures in this sequence of poses. The gesture descriptions are passed to the graph DB wrapper which translates them into database queries for updating the controller's state and to visualize query results.

2 Gesture Detection using the Kinect System

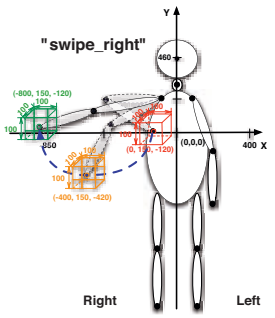
The Kinect camera comprises three types of sensors: a real-time infrared depth camera, a simple color camera, and multiple microphones. The cameras deliver 640x480 pixel images with a centimeter resolution at a rate of 30 frames per second. The sensors work at distances from 0.8m to 3.5m.

For communication with the Kinect, we used the OpenNI framework [Ope]. It implements several middleware components and provides interfaces for sensors as well as for applications. Components are registered in the form of *modules* and can be used for accessing the sensor values directly, or at a higher abstraction level as provided by NITE [NIT]. NITE implements modules for full body analysis and realizes the tracking of a human in form of a skeleton which consists of 15 characteristic joints like head, left/right hand, etc. The 3D coordinates of each skeleton joint are calculated every ten milliseconds using the 2D color image in conjunction with the depth image. For example, the coordinates of the left foot can be accessed with the following statements:

```
XnSkeletonJointPosition joint;
g_UserGenerator.GetSkeletonCap().GetSkeletonJointPosition(
     playerID, XN_SKEL_LEFT_FOOT, joint);
```

3 InfiniteGraph as a graph database

Our demonstration is based on InfiniteGraph [IG], developed by Objectivity Inc., as one representative of (commercially available) graph database systems. InfiniteGraph is a schema based graph database, that defines its schema using Java objects. Database management and access is provided by the class `GraphFactory`. Base classes for nodes and edges are provided, i.e. `BaseVertex` and `BaseEdge`, from which a user can derive application-specific subclasses. InfiniteGraph offers a transactional concept to insert, delete, or modify nodes or edges. A given graph can be traversed or searched. For this, methods to retrieve connected edges or neighboring nodes and such are provided in the base classes. Searching is based on a `Guide` that defines the order in which nodes are traversed. InfiniteGraph supports indexing according to class properties. In our demonstration we will focus on the traversal of a given graph. Although, we implemented our demonstration for InfiniteGraph, the approach is generally transferrable to similar systems like neo4j, where similar interfaces are available.




```
SELECT "swipe_right"
MATCHING (
  kinect(
    abs(0 + left_hand_x - torso_x) < 100 and
    abs(-150 + left_hand_y - torso_y) < 100 and
    abs(120 + left_hand_z - torso_z) < 100
  ) ->
  kinect(
    abs(400 + left_hand_x - torso_x) < 100 and
    abs(-150 + left_hand_y - torso_y) < 100 and
    abs(420 + left_hand_z - torso_z) < 150
  ) within 1 seconds select first consume all
) ->
kinect(
  abs(800 + left_hand_x - torso_x) < 100 and
  abs(-150 + left_hand_y - torso_y) < 100 and
  abs(-120 + left_hand_z - torso_z) < 100
) within 1 seconds select first consume all;
```

Figure 2: "Swipe_Right" gesture for expanding a selected edge

4 The Controller and Database Interaction

After identifying gestures based on events by the AnduIN system, it is the controller's task to derive queries for the graph database based on the actions and the current state and trigger adequate queries for the graph database.

For this demonstration we have implemented a basic set of gestures/queries on the InfiniteGraph database focusing on graph navigation. Each detected gesture triggers a new query and results are displayed on the screen. The gestures are

- select edge/node - the user can select one of the currently visualized nodes or edges
- about edge/node - the user can request additional information about a node or edge, i.e. get the properties
- undo/redo - during a graph traversal, undo and redo of moves are enabled
- shortest path - the user can query the database for the shortest path between two present nodes
- solve - triggers a shortest path query between a start and end node

For example, a selected *Person* node of "Johnny Depp" is shown in Figure 3. The visualized graph can be expanded further by selecting an outgoing edge and expanding it with a "Swipe_Right" gesture. The gesture and its corresponding AnduIN CEP translation is illustrated in Figure 2. Once, the gesture is detected by AnduIN, the new node is requested from the InfiniteGraph database with *getNode(ID)*, where ID corresponds to the outgoing selected edge (comp. "The Ginger Man" in Figure 3), and is derived by the controller from the preceding select edge gesture.

5 Demonstration

For the demo session, we plan to show our prototype system in action. We will bring a notebook equipped with a Kinect camera running our InfiniteGraph interface for IMDB. Everyone can try to navigate the database by gestures playing the Kevin Bacon [Kev] game. The game is simple, one starts out with a node representing the actor Kevin Bacon and traverses the graph through gesture navigation to find the shortest path to another given actor (Figure 3). The length of the path is then the Kevin Bacon number of the other actor.

For the database we use a simple data model. There are two node types, `Person` and `Project`, and one edge type `workedOn`. `Person` contains name and gender of an actor, `Project` contains name and release year of a movie. `workedOn` contains the name of the role an actor played in a movie.

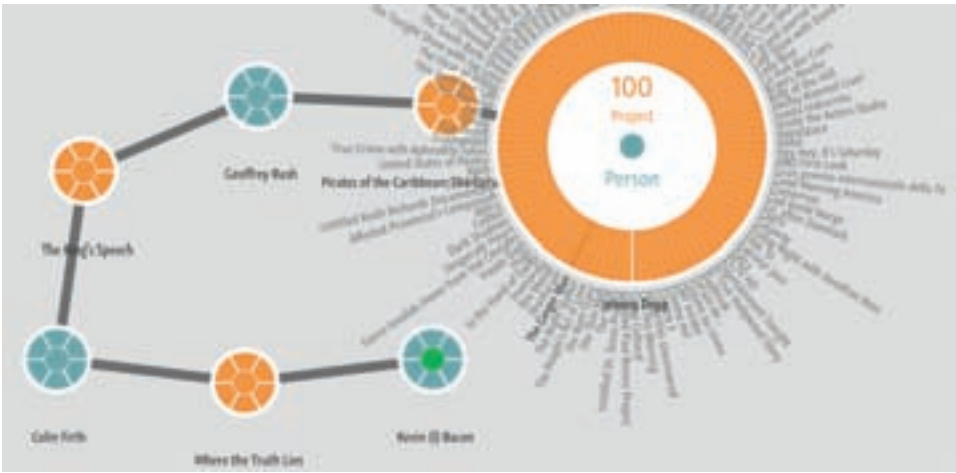


Figure 3: Screenshot of the demonstration showing a path between Kevin Bacon and Johnny Depp.

For a user playing the game, this means he or she has to select a node, query its properties, check for its outgoing edges and decide which next node, i.e. movie or actor, to choose. In addition, we allow a glance into the internals of the overall system. Raw events generated by the Kinect camera are passed to our stream engine AnduIN, which performs the gesture detection. Due to the usage of AnduIN's CEP functionality, necessary gestures can be exchanged on the fly. To show this effect, we prepare different gestures for a single interaction with the database. Additionally, the usage of the simple SQL like declarative user interface offers an easy way to define new gestures. After a short briefing, participants should be able to define there own gestures and apply them instead of the original ones. This way it will be possible for users to customize UI controls for traversing the graph.

References

[HSS+12] Steffen Hirte, Eugen Schubert, Andreas Seifert, Stephan Baumann, Daniel Klan, and Kai-Uwe Sattler. Data3 - A Kinect Interface for OLAP using Complex Event Processing. In *ICDE*, April 2012. Video: http://youtu.be/DROXI0_wDRM.

[IG] InfiniteGraph. http://objectivity.com/INFINITEGRAPH.

[Kev] The Kevin Bacon Game. `http://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon`.

[KKH+11] D. Klan, M. Karnstedt, K. Hose, L. Ribe-Baumann, and K. Sattler. Stream Engines Meet Wireless Sensor Networks: Cost-Based Planning and Processing of Complex Queries in AnduIN, Distributed and Parallel Databases. *Distributed and Parallel Databases*, 29(1):151–183, January 2011.

[NIT] NITE Middleware. `http://www.primesense.com/?p=515`.

[Ope] OpenNi. `http://www.openni.org/images/stories/pdf/OpenNI_UserGuide_v3.pdf`.

[SFC+11] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. *Computer Vision and Pattern Recognition*, June 2011.

YAGO2s: Modular High-Quality Information Extraction with an Application to Flight Planning

Fabian M. Suchanek, Johannes Hoffart, Erdal Kuzey, Edwin Lewis-Kelham
Max Planck Institute for Informatics, Germany

Abstract: In this paper, we present YAGO2s, the new edition of the YAGO ontology [SKW07, HSBW12]. The software architecture has been refactored from scratch, yielding a design that modularizes both code and data. This modularization enables us to add in new data sources more easily, while still maintaining the high accuracy and coherence of the ontology. Thus, we believe that YAGO2s occupies a sweetspot between a centralized design and a completely distributed design.

In this demo, we present an application of this design to the task of planning a flight. Our proposed system finds flights between all airports close to the departure city to all airports close to the destination city.

1 Knowledge Base Construction

In recent years, many projects have successfully created large-scale knowledge bases (KBs) in an automated fashion. The KBs contain millions of entities (such as rivers, universities, people, and movies), and millions of facts about them (such as who acted in which movie, which river is located in which country, etc.). There are several strategies to build such KBs. One strategy is to accumulate and reconcile as much data as possible. Projects such as TextRunner [BCS+07], and NELL [CBK+10] follow this strategy, as did YAGO [SKW07, HSBW12]. The advantage of this method is that the knowledge is largely coherent, because it is curated by a single provider. The drawback is that the project becomes more and more difficult to maintain as more resources are integrated, and as more people work on it. An alternative approach is to furnish only small pieces of data, and to interlink these. This is the approach favored by the Semantic Web community as *Linked Open Data*. The advantage of this strategy is that individual datasets are curated by their respective owners, thus modularizing the data and distributing the work. The drawback is that it is challenging to establish links between the data sets. Thus, the quality of the cross-ontology data is often not perfect [HHM+10, DSSM10]. The DBpedia ontology [ABK+07] is pursuing another approach: It relies on information extraction, but outsources some of the manual tasks to a community. This has the advantage of distributing the work. At the same time, it can lead to slight incoherences [HSBW12], and nothing is known about the accuracy of the data in DBpedia.

The YAGO project started in 2007 by combining WordNet [Fel98] and Wikipedia to a coherent general-purpose KB. Thus, YAGO was in the camp of the centralized KBs. This allowed it to enforce accuracy and coherence on its data. Every entity in YAGO and every relationship is unique. Manual evaluations proved [SKW07, HSBW12] that the probability that a given statement in YAGO is correct stands at 95%. This focus on precision is the main characteristics of YAGO in the realm of automatically constructed KBs.

In recent years, more and more people have joined the YAGO team, and more and more resources have been integrated into the KB. This yielded YAGO2 [HSBW12]. Today, YAGO is driven by a core team of 5 people, with around a dozen more people working on directly related projects. With more people joining, and now a small research group dedicated to ontology development, the centralized mode was no longer sustainable. However, a distributed mode in the spirit of the Semantic Web or a community-based approach makes it harder to achieve the data quality or the coherence of YAGO2. Therefore, we have opted for a middle course, coined YAGO2s ("YAGO 2 star"). This new framework is based on the data sources and the extraction mechanisms of YAGO2 [HSBW12]. However, the entire software architecture for the new YAGO2s has been reengineered from scratch.

2 The YAGO2s Architecture

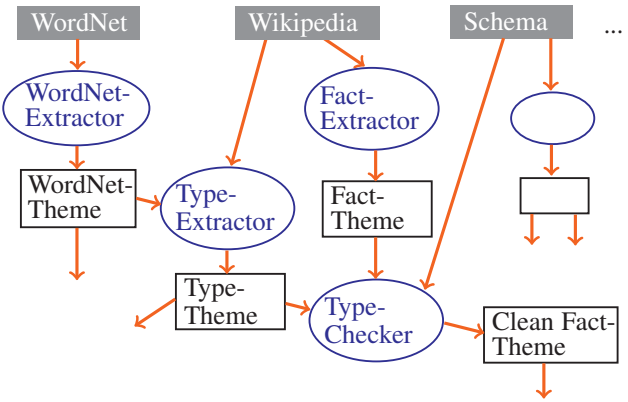


Figure 1: The YAGO2s Architecture

We have completely refactored the ontology extraction framework into a modular structure (Figure 1). There are 39 extractor modules. Each module receives a data source as input. Input sources are WordNet [Fel98], Wikipedia, WordNet Domains [BFMP04], the Universal WordNet [dMW09], and Geonames. Others can be added. Each extractor produces one or multiple *themes* as output. A theme is a set of RDF triples. For example, the module "WordNet-Extractor" receives WordNet as an input source, and produces an output theme called "WordNet-Theme". This output theme contains RDF triples extracted from WordNet. The data sources can also be files that contain handwritten data. The schema of YAGO, e.g., which defines the relations with their domains and ranges, is defined manually in YAGO. Such data can be supplied to the modules as input sources. This ensures that all modules operate on the same predefined schema. Modules can also take other themes as input. In the figure, the module "Type-Extractor" requires the theme "WordNet-Theme" as input theme. The module uses data from WordNet and Wikipedia to build the YAGO type system ("Type-Theme"). This theme can become again the input of other modules. This yields a dependency graph of extraction modules.

Modules can be added or removed ad libitum, as long as the dependencies are respected. In order to guarantee the data quality, we provide some modules that check the output themes

of other modules. The "Type-Checker", e.g., filters out statements that do not conform to the domain and range constraints. The output theme of the type checker is a cleaned theme, which following modules can use as input. A Deduplicator Module takes a similar role, deduplicating statements and entities. A Rule Module applies the deduction rules of YAGO2 to deduce implied facts (such as facts induced by symmetric relations). These modules ensure that every statement that makes it into the final YAGO themes has been deduplicated, and checked for type coherence. A revision checker signals if statements were extracted in a previous run of the system on a previous version of the data sources, but went missing in the current run. A sample of these statements can then be checked manually to see, e.g., if Wikipedia infobox attribute names have changed.

This architecture has a number of distinct advantages: First, it modularizes the information extraction process. Every team member can be responsible for one or multiple modules. Second, it modularizes the data. YAGO2s will be made available in theme slices, so that users can download just the themes they desire. Third, this architecture allows for efficient parallelization. Our scheduling system runs modules that do not depend on each other in parallel. Fourth, different from a truly distributed approach, our architecture helps keeping the data coherent by allowing the easy re-use of data-cleaning components across various sources. This is achieved by a predefined schema as input, and the checker modules that verify the output. This way, our architecture implements a controlled trade-off between a centralized approach and a distributed approach.

The native data format of YAGO2s is now Turtle. The fact identifiers of YAGO, which are used to attach time and space information to facts, are stored in the files in a commented line before the fact. This allows standard RDF engines to consume YAGO themes without the fact identifiers. We have also made YAGO's terminology fully RDF and OWL compliant. In addition, the new data set contains a theme with WordNet Domains [BFMP04], which give a topic structure to YAGO. Thus, it is now possible to ask for all entities related to, e.g., "geography". We have re-evaluated the ontology by manual sampling, in the same way as described in [HSBW12]. Overall, we evaluated over 3700 facts. Our evaluation confirmed again a fact accuracy of 95%. YAGO2s is available at `http://yago-knowledge.org`.

3 An Application: Flight Planning

The new architecture of YAGO2s makes it easier to integrate new data sources in a controlled environment. To demonstrate this, we added a new module to the framework that provides information about flights. This information can be extracted from Wikipedia pages about airports. These pages contain tables with the names of the flight companies that operate at the airport, together with their destination airports. We designed a simple information extractor that reads out these tables from Wikipedia. This piece of code acts as a module in our framework. Its output is checked by the type checker and the other checker modules, thus ensuring smooth integration with the rest of YAGO2s.

In this demo, we show how useful the new YAGO2s is with this module. Many commercial Web sites allow searching and booking flights. However, the user usually has to specify the departure airport and the destination airport. If the user lives close to several airports, this can lead to a combinatorial problem. As an example, take a user based in Saarbrücken,

a small city in the West of Germany. Assume that he wishes to go to the Ligurian cost, in Italy. Assume also that he is willing to make a trip of up to 3 hours to the airport. Then there are at least 11 airports that could serve as departure airports.[1] There are also at least 6 airports that are close to the Ligurian cost.[2] This yields a total of 66 possible connections from departure airports to destination airports. With current services, the user has to try out all of them until there is a suitable match. This process can easily take several days (as one author of this paper experienced).

With YAGO2s, this task can be simplified. YAGO2 contained already many airports and cities, together with their coordinates. With the new flight information module, YAGO2s knows also which companies offer flights between which airports. This allowed us to build a system that can suggest flight connections to the user. The user simply enters the city of departure and the city of arrival. Our system seeks all airports within a predefined distance of the departure city, and finds all direct flights to airports in the vicinity of the target city. Then the system displays all connections, along with their trajectory on a map. A search for "Saarbrücken to Genova", e.g., yields 4 possible flight connections, with 3 different airlines. We link these to the Web page of a travel company, so that the user can check flight availability and book the flight. This way, YAGO2s acts as an entrance portal to the commercial service, giving a true added value to the user. Our demo is available at `http://www.mpi-inf.mpg.de/yago-naga/yago/flights.html`.

References

[ABK+07] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A Nucleus for a Web of Open Data. In *ISWC*, 2007.

[BCS+07] M. Banko, M. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open Information Extraction from the Web. In *IJCAI*, 2007.

[BFMP04] L. Bentivogli, P. Forner, B. Magnini, and E. Pianta. Revising WordNet Domains Hierarchy. In *COLING Workshop on Multilingual Linguistic Resources*, 2004.

[CBK+10] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. Hruschka Jr., and T. Mitchell. Toward an Architecture for Never-Ending Language Learning. In *AAAI*, 2010.

[dMW09] Gerard de Melo and Gerhard Weikum. Towards a Universal Wordnet by Learning from Combined Evidence. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM 2009)*, pages 513–522, New York, NY, USA, 2009. ACM.

[DSSM10] L. Ding, J. Shinavier, Z. Shangguan, and D. McGuinness. SameAs Networks and beyond: Analyzing deployment status and implications of owl:sameAs in Linked Data. In *ISWC*, 2010.

[Fel98] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[HHM+10] H. Halpin, P. Hayes, J. McCusker, D. McGuinness, and H. Thompson. When owl:sameAs isn't the Same: An Analysis of Identity in Linked Data. In *ISWC*, 2010.

[HSBW12] J. Hoffart, F. Suchanek, K. Berberich, and G. Weikum. YAGO2: a spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence Journal*, 2012.

[SKW07] F. Suchanek, G. Kasneci, and G. Weikum. YAGO: A Core of Semantic Knowledge. In *WWW*, 2007.

[1] Ensheim, Zweibrücken, Hahn, Karlsruhe, Frankfurt, Stuttgart, Luxembourg, Strasbourg, 3 airports in Paris.
[2] Nice, Genova, Turin, Milan, Bergamo, Pisa.

EvenPers: Event-based Person Exploration and Correlation

Christian Kapp, Jannik Strötgen, Michael Gertz

Institute of Computer Science, Heidelberg University
69120 Heidelberg, Germany
c.kapp@stud.uni-heidelberg.de
{stroetgen,gertz}@informatik.uni-heidelberg.de

Abstract: Searching for people on the Internet is one of the most frequent search activities. In this paper, we present *EvenPers*, a system for the event-based exploration of persons and person similarities. We address challenges such as cross-document person name normalization and present a novel approach to calculate person similarities based on their event information. In our demonstration, we show several exploration scenarios illustrating the usefulness of *EvenPers* and its exciting functionality.

1 Motivation and Objectives

A very common activity on the Web is that users search for information about (prominent) people. A major problem in identifying documents that are relevant to a user's search query is that person names need to be disambiguated. That is, a person can be referred to by different expressions (including personal pronouns) in documents, which need to be identified as such and mapped to some *normalized* person name. A typical example is the different ways the president of the US is referred to in documents. All expressions relating to the president first need to be identified as such in documents and mapped to a single expression, ideally the full name of the person, before a ranking of documents relevant to the search query can be determined. Several approaches have been proposed for person name disambiguation, see, e.g., [Chr06, YIO+10].

In our approach, we go a step further and consider an *event-based context* in which person expressions occur in documents. The motivation is that a person can be well characterized by the events he or she was or will be involved in. For this, we use a simple yet powerful notion of event as a combination of a time and geographic expression, typically at the sentence level [SGJ11, SG12a]. The idea then is to combine such event information with person expressions, leading to the construction of a *person's event profile*. A search result then is not a list of relevant documents but a ranked list of events related to the person. Such a view leads to new functionality to explore information about a person, e.g., a search based on a certain time interval or geographic region of interest. A further novelty of our approach is that event-profiles for persons are used to determine persons that are similar based on the events they were involved in. Thus, our approach and system outlined in the following sections provide new functionality to search for and explore person information.

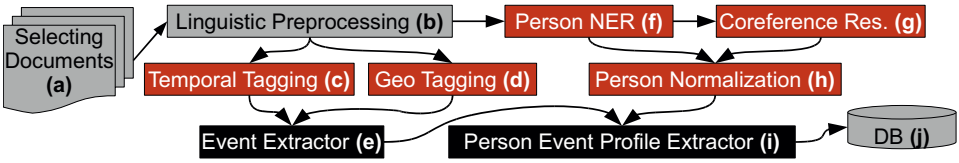


Figure 1: *EvenPers* document processing pipeline with extraction and normalization of temporal, geographic, and person information (red) and the event and person event profile extractors (black).

2 System Description

Processing Pipeline: Our document processing pipeline is depicted in Figure 1. After selecting documents from the corpus (a), linguistic preprocessing, such as sentence splitting, tokenization, and part-of-speech tagging, is performed (b). The results of this step are then available for further annotation tools. All temporal expressions (c) and geographic expressions (d) are extracted and normalized using HeidelTime [SG12b] and Yahoo! Placemaker[1]. These are then combined into events of the form $e = \langle t, dp_t, c_t, g, dp_g, c_g \rangle$ with normalized temporal and geographic values t and g, document/position information dp_t and dp_g, and confidence values for correct normalization c_t and c_g, respectively (e).

In parallel, named entity recognition for detecting person names is performed (f) by StanfordNER [FGM05] and the OpenNLP NER tool[2]. In the next step, several tools are applied to resolve coreferences (g), namely Arkref [HK09], Cherrypicker [RN09], and the Illinois Coreference Package [BR08]. Since there are hardly any sophisticated person name normalization tools, we developed our own tool for this task (h): Based on the extracted NER and coreference information of the previous applied tools, the person information is merged into person chains before Wikipedia and JRCNames[3] are checked for different name variations of each item of the person chain to associate a Wikipedia and/or JRCNames ID, if available. Depending on detection and normalization details, a confidence value c_p can be added to every reference to a person in addition to document/position (dp) and normalization information (id_p), resulting in a reference to a person as $p = \langle id, dp_p, c_p \rangle$.

Finally, the Person Event Profile Extractor (i) combines events with person information to create a person event profile $pep(p)$ for every person detected in the corpus. For this, co-occurrences of events and references to persons within a specified window size are determined in every document of the corpus, resulting in a person event profile of the form $pep(p) = \{\langle e_1, p_1 \rangle, ..., \langle e_n, p_n \rangle\}$. Note that for every item, a confidence value can be calculated depending on c_t, c_g, and c_p. All pep are stored in a PostGIS database (j).

Person Similarity Calculation: For determining the similarity between persons, we rely on a model for event-centric document similarities described in [SGJ11]. For every two

[1] Yahoo! PlaceMaker: http://developer.yahoo.com/geo/placemaker/
[2] OpenNLP: http://opennlp.apache.org/index.html
[3] JRCNames: http://langtech.jrc.it/JRC-Names.html

persons p_1 and p_2, event similarities $esim$ for the cross-product of the events in $pep(p_1)$ and $pep(p_2)$ are calculated based on the events' temporal and geographic granularities as detailed in [SGJ11]. Then, we build the sum over all $esim$ weighted by their confidence weighting factor (wf), and normalize the value by the size of $pep(p_1)$ and $pep(p_2)$ and the average confidence $avgc$, resulting in a person similarity function of the form:

$$personSim(p_1, p_2) := \frac{\sum_{e_i \in pep(p_1)} \sum_{e_j \in pep(p_2)} wf(e_i, e_j) \times esim(e_i, e_j)}{|pep(p_1)| \times |pep(p_2)| \times avgc}$$

3 Demonstration

Corpus: The prerequisites of the underlying data set for our demonstration are that the corpus contains (i) information of many different persons, (ii) about different times, (iii) with the information about the persons being somehow related to each other. Then, it should be possible to identify reasonable person similarities based on our approach described in Section 2. For this, we selected all documents with the same category tag (politics) of the New York Times corpus[4], resulting in 209,795 news documents with publication times between 1987 and 2007. These documents contain over 5 million personalized events about 82,616 different persons.

Demonstration Scenarios: We present the following demonstration scenarios, which we briefly explain based on the screenshot of *EvenPers* depicted in Figure 2:

(a) Searching for persons ("Helm") results in a hit list. After the user makes a selection ("Helmut Kohl"), the person's details are presented: a picture, the Wikipedia link (if available), the person's most important events and most similar persons (left side of Figure 2). On the map, the person's events are anchored at the places where the events occurred.

(b) In the *person-centric exploration scenario*, a second person is selected from the "most similar persons" list ("Bill Clinton"). His/her events are added to the map and shared events are highlighted in a special color. The events of the two persons can be directly compared with each other and explored using event snippets described in (d).

(c) In the *event-centric scenario*, the user selects an event on the map or in the "most important events" list. In our example, the user selected the event "1998-05-14 – Potsdam, Brandenburg, DE". The event information is presented in a snippet anchored at the event's location as shown in the figure and described next.

(d) Event snippets contain: (i) normalized time and place information, (ii) a list of persons sharing the event – the user can select a person from the list to compare it with the first person as in the person-centric scenario, (iii) a list of contexts, containing event occurrences of the persons under investigation, with highlighted time, place, and person information.

(e) Finally, the user can filter events based on their importance, confidence, and year.

As the scenarios demonstrate, *EvenPers* provides exciting functionality on exploring single persons and person similarities based on their events extracted from different documents.

[4]New York Times Corpus is available from LDC (http://www.ldc.upenn.edu/), catalog number LDC2008T19.

Figure 2: The *EvenPers* system: Exploring similarities between Helmut Kohl and Bill Clinton.

References

[BR08] Eric Bengtson and Dan Roth. Understanding the Value of Features for Coreference Resolution. In *EMNLP'08*, pages 294–303, 2008.

[Chr06] Peter Christen. A Comparison of Personal Name Matching: Techniques and Practical Issues. In *ICDM'06 Workshops*, pages 290–294, 2006.

[FGM05] Jenny R. Finkel, Trond Grenager, and Christopher Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *ACL'05*, pages 363–370, 2005.

[HK09] Aria Haghighi and Dan Klein. Simple Coreference Resolution with Rich Syntactic and Semantic Features. In *EMNLP'09*, pages 1152–1161, 2009.

[RN09] Altaf Rahman and Vincent Ng. Supervised Models for Coreference Resolution. In *EMNLP'09*, pages 968–977, 2009.

[SG12a] Jannik Strötgen and Michael Gertz. Event-centric Search and Exploration in Document Collections. In *JCDL'12*, pages 223–232, 2012.

[SG12b] Jannik Strötgen and Michael Gertz. Multilingual and Cross-domain Temporal Tagging. *Language Resources and Evaluation*, pages 1–30, 2012. 10.1007/s10579-012-9179-y.

[SGJ11] Jannik Strötgen, Michael Gertz, and Conny Junghans. An Event-centric Model for Multilingual Document Similarity. In *SIGIR'11*, pages 953–962, 2011.

[YIO+10] Minoru Yoshida, Masaki Ikeda, Shingo Ono, Issei Sato, and Hiroshi Nakagawa. Person Name Disambiguation by Bootstrapping. In *SIGIR'10*, pages 10–17, 2010.

DrillBeyond: Open-World SQL Queries Using Web Tables

Julian Eberius, Maik Thiele, Katrin Braunschweig and Wolfgang Lehner

Database Technology Group
Department of Computer Science
Dresden University of Technology
D-01062 Dresden
{firstname.lastname}@tu-dresden.de

Abstract: The Web consists of a huge number of documents, but also large amounts structured information, for example in the form of HTML tables containing relational-style data. One typical usage scenario for this kind of data is their integration into a database or data warehouse in order to apply data analytics. However, in today's business intelligence tools there is an evident lack of support for so-called situational or ad-hoc data integration. In this demonstration we will therefore present *DrillBeyond*, a novel database and information retrieval engine which allows users to query a local database as well as the web datasets in a seamless and integrated way with standard SQL. The audience will be able to pose queries to our DrillBeyond system which will be answered partly from local data in the database and partly from datasets that originate from the Web of Data. We will demonstrate the integration of the web tables back into the DBMS in order to apply its analytical features.

1 Open-World SQL Queries

The system we want to demonstrate offers a novel way of integrating web tables into regular query processing in a relational database. We present a modified RDBMS that is able to answer so-called open-world queries which are not restricted to the schema of the local database. Instead the user is allowed to use arbitrary attribute names that do not appear in the original schema. Consider the following running example query:

```
SELECT population, n_name, AVG(o_totalprice)
    FROM nation
    JOIN region ON n_regionkey=r_regionkey
    JOIN customer ON n_nationkey=c_nationkey
    JOIN orders ON c_custkey=o_custkey
WHERE
    r_name = 'AMERICA'
GROUP BY population, n_name
ORDER BY population
```

The `population` attribute which is used in the `SELECT` and `ORDER BY` clauses is not part of the TPC-H schema and therefore requires special processing. In the DrillBeyond system, missing attributes are translated into keyword queries that are run against an index of open datasets on the web. It will answer the query by substituting the missing attribute

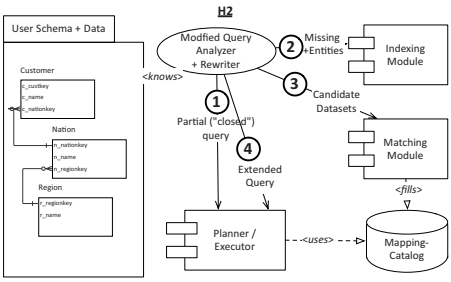


Figure 1: Architecture and Query Processing

Figure 2: Screenshot of the web front end used for the demonstration

values, e.g. for `population`, with values from the retrieved web datasets that are integrated into the local schema on the fly. In contrast to pure search systems, DrillBeyond is able to exploit the local schema and data as well as the context given by the SQL query to find the best matching web datasets. As usual with IR style approaches, the system will not be able to find the optimal dataset completely automatically, but instead needs to present the user with a ranked result list. Apart from identifying web datasets that can be potentially used to answer the open-world SQL query, the challenges lie in identifying the attributes of the web datasets that contain the correct values, as well as identifying those attributes that can be used to join the local table and the web data table.

2 System Architecture

The DrillBeyond system is implemented inside the open source RDBMS H2[1] by adding the following components: a *Modified Query Analyzer and Rewriter*, an *Indexing Module* and a *Schema/Instance Matching Module*. All other components of the system, such as the query optimizers, or join implementations, can be reused unmodified, as the output of the preprocessing steps is a standard SQL query referencing standard database objects. Figure 1 gives a global overview of the demo system, including the new DBMS components and query processing steps.

Modified Query Analyzer and Query Rewriting In a RDBMS, the query analyzer maps tokens from the SQL query to objects in the database, e.g., the token `n_name` to the corresponding attribute in the nation table (if this table is given in the respective `from` clause). If a token can not be mapped to a database object, an error is raised. For this demo, we modified H2's query analyzer to instead trigger a search for fitting datasets that can be used to answer the query. Specifically, the query processor will use the unknown token as

[1] `http://h2database.com`

one input for the search, but also the token's *context*, i.e., the related database objects and instance data. In our example, as the unknown `population` attribute is related to the `nation` table, we use instance data from the `nation` table to aid the search for a fitting join partner in the DrillBeyond index as described the following sections. Furthermore, the presented system also takes the specific query into account by applying local operations before looking for candidate datasets. In the running example, the selection on the region name is applied on the nation table before consulting the Indexing Module. In this way, the search for candidate datasets can return more specific results, while the amount of matching work that needs to be done can be minimized.
After collecting the set of relevant local tables and instances (the *context*), the analyzer calls the Matching and the Indexing Module, performs query rewriting and finally passes control to the regular optimizer and the executor. These new steps in query processing are depicted as numbers 1 to 3 in Figure 1. First, the extracted keyword token and its context are passed to the Indexing Module, which returns a list of candidate datasets. This list is passed to the Matching Module that checks which of those candidates can be joined with the local data at all, and if possible saves mappings in the catalog. In a third step the original query plan is rewritten to include a new attribute created from the candidates. In the following sections, we will give more details on the indexing and matching modules as well as on the web front used in the demo.

Indexing Module The Index Module keeps an local index of web datasets. It indexes the datasets metadata, such as title, context and attribute names as well as the text column values. For this demo, the index is realized using Lucene which supports, among others features, normalization/stemming and boolean keyword queries. We added synonym expansion via WordNet[2] to be able to identify additional candidate datasets. A lookup in the index is performed using the unknown tokens and their context, as passed from the query analyzer. The result ranking is a mixture of classical keyword-search ranking, e.g., comparing the query tokens to dataset metadata and attribute names, but also instance-based techniques, which are applied in the Matching Module to refine the ranking. Continuing our running example, the index lookup will identify the queried term `population`, its synonyms, and the selected American nation names such as Argentina, Brazil or the US, in several datasets, and pass them to the next stage.

Matching Module Since there are no foreign-key relationships between the local and open datasets, a join can only be performed when at least one matching column pair of the local table and the respective open dataset can be identified. Therefore, the *Matching Module* employs a set of classic schema and instance matching techniques, such as string similarity measures and external knowledge such as synonym dictionaries. By doing this we are able to rank the result candidates produced by the index lookup and to prune all datasets which can not be joined. The ranking is influenced mainly by the quality of the mapping, e.g., the monogamy and coverage of the created mapping between two datasets. If a join candidate is found, the instance level mappings between the matching attributes are stored in the mapping catalog to establish a foreign key relationship between the two

[2]http://wordnet.princeton.edu/

datasets. The stored mappings are later used to perform the joins to produce the actual query result. Continuing the example, the Matching module will bridge differences between the country names in TPCH and the candidate datasets, and will also prune datasets that do not contain all the necessary countries, e.g., datasets only about South American countries, as the mapping coverage will be lower in these cases.

Web Frontend In addition to the H2 back end, we implemented a web front end which enables the interactions with the user, such as presentation of the search results and selection of a fitting dataset from the candidates. Figure 2 gives an impression of this interface. The users can enter regular SQL queries and browse different variations of the query results, depending on which candidate dataset is used to answer the query. For each candidate, the front end will display the dataset's schema, the attributes matching the local table, the attributes (potentially) containing the missing values as well as sample rows of the query result when the respective candidate is chosen. Finally, for each open dataset the available metadata as indexed by the Indexing module can be viewed. This allows the users to make a more informed choice about which open datasets to use to complement their data.

3 Demo Walkthrough

In this live demonstration, users will be able to get a feeling for the potential of Web Data in data analytics by posing SQL queries including undefined attributes to the DrillBeyond system. Using an console or the web interface, they will be able to choose from different preloaded local databases to perform analysis on. The preloaded schemata include TPC-H and an IMDb sample. Then, the users can enter SQL queries on the chosen schemata, using open attributes as they see fit. The demo system will consult its index of open datasets, which for this demo, contains about one million web tables extracted from the English version of Wikipedia. Depending on the tool used, the audience will be able to study query results as one raw SQL result table on the console, or as individual query results depending on a selected candidate when using the web interface as shown in Figure 2. In the web front end they also have access to the metadata, schema and sample rows of the candidate datasets. A screencast demonstrating both the raw SQL console as well as the web front end is available on the web[3].
Please note that this demonstration is an extended version of [ETBL12] presented at VLDB'12.

References

[ETBL12] Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. DrillBeyond: Enabling Business Analysts to Explore the Web of Open Data. *PVLDB*, 5(12):1978–1981, 2012.

[3] `http://wwwdb.inf.tu-dresden.de/edyra/DrillBeyond`

Die „schlaue Stadt“ - Erzeugung virtueller Sensordaten für Smart City Anwendungen

Marcus Behrendt[1], Mischa Böhm[1], Marina Borchers[1], Mustafa Caylak[1], Lena Eylert[1], Robert Friedrichs[1], Dennis Höting[1], Kamil Knefel[1], Timo Lottmann[1], Andreas Rehfeldt[1], Jens Runge[1], Sabrina-Cynthia Schnabel[1], Stephan Janssen[2], Daniela Nicklas[3], Michael Wurst[4]

[1, 3]Universität Oldenburg

[2]OFFIS Institut für Informatik, Oldenburg

[4]IBM Research, Dublin

[1]pg-alise@informatik.uni-oldenburg.de

[2]stephan.janssen@offis.de

[3]dnicklas@acm.org

[4]mwurst@ie.ibm.com

Abstract: In der Smart City-Anwendung der Projektgruppe ALISE werden Verkehrs-, Wetter- sowie Energiedaten synthetisch generiert und verarbeitet, um Sensordaten und Auswirkungen von Ereignissen darzustellen. Das Gesamtsystem besteht aus einer verteilten Simulation, einem Datenstrommanagementsystem, einer Business Intelligence Lösung und einem Kartendienst. Ein Operation Center und Control Center bilden die Interaktionspunkte des Systems.

1 Motivation

Durch die zunehmende Vernetzung über das Internet und heutzutage allgegenwärtigen Sensoren entstehen auf der Welt riesige Datenmengen. Die logische Vernetzung dieser Daten wird durch Ansätze unterschiedlicher Unternehmen angestrebt, unter Begriffen wie „Smarter Planet“ oder „Smarter Neighborhood“ [SIE12, CIS12]. In der studentischen Projektgruppe ALISE (Advanced Live Integration of Smart City Environments) wird in Kooperation mit IBM Research Dublin eine Smart City-Anwendung für die Simulation einer Beispielstadt erstellt. Die „A Smarter Planet“-Initiative von IBM versucht verschiedene Sensordaten zu vernetzen. Mit Hilfe hochentwickelter Analysemethoden soll in Echtzeit aussagekräftiges Wissen abgeleitet werden, um einen „smarteren“ Planeten zu schaffen und Probleme zu lösen, die ohne dieses vernetzte Wissen kaum lösbar wären [IBM12]. Die Verarbeitung dieser Daten- und Datenstrommengen erfordert leistungsfähige Systeme, die häufig aufgrund nicht ausreichend verfügbarer (synthetischer) Sensordaten nur unzureichend getestet werden können. Um dieses Problem zu umgehen, können spezielle Datengeneratoren wie bspw. BerlinMOD [BDG09] oder der „Network-based Generator of Moving Objects“ [Bri02] verwendet werden, die synthetische Sensordaten erzeugen, welche in solche Systeme eingespeist werden können. Jedoch ist eine umfassende, ereignisgesteuerte Simulation von Verkehr,

Energie und Wetter in den genannten Systemen nicht gegeben. Der Schwerpunkt des studentischen Projektes liegt dabei auf der Generierung und Visualisierung von synthetischen Sensordaten auf Basis der Simulation einer Stadt. Diese umfasst die Bereiche des Verkehrs (wie z. B. den Individual- und den öffentliche Personen Nahverkehr), des Wetters und der Energie. Ein wichtiger Nachhaltigkeitsaspekt ist die Simulation von Energieverbrauchern und Energieerzeugern. Dezentrale Energieerzeuger, wie eine Solaranlage und eine Windkraftanlage, können in der schlauen Stadt verteilt und simuliert werden. Ebenso wird der Energieverbrauch der gesamten Stadt auf Grundlage von realen Lastkurven simuliert.
Das Alleinstellungsmerkmal der Simulation ist, dass sie durch Parameter und Ereignisse, wie z. B. Wetterereignisse oder Erhöhung der Verkehrsteilnehmer, über eine grafische Oberfläche beeinflusst werden kann. Die Auswirkungen dieser Änderungen wirken sich direkt auf die Simulation und auf die Sensordaten und deren Visualisierung aus.

2 Architektur

Die Architektur enthält Komponenten zur Generierung (Simulation einer Beispielstadt), Verarbeitung, Speicherung und Visualisierung von Sensordaten (siehe Abbildung 1). Das Gesamtsystem ist derart gestaltet, dass eine hohe Skalierbarkeit erreicht werden kann. Dazu werden einzelne Softwarekomponenten auf unterschiedliche Hardware verteilt. Die Verteilung geschieht sowohl horizontal als auch vertikal. Für die Generierung wird ein zuvor erstelltes Simulationsframework verwendet, das simulierte Objekte erzeugt. Simulierte Sensoren erfassen diese Objekte und senden Daten in Form von Datenströmen an ein Datenstrommanagementsystem. Dieses verarbeitet den Datenstrom und speichert sie für weitere Data Mining Analysen in einer Datenbank. Im Anschluss werden die Sensordaten zur Laufzeit in einer Business Intelligence Lösung sowie mit einem Kartendienst visualisiert.

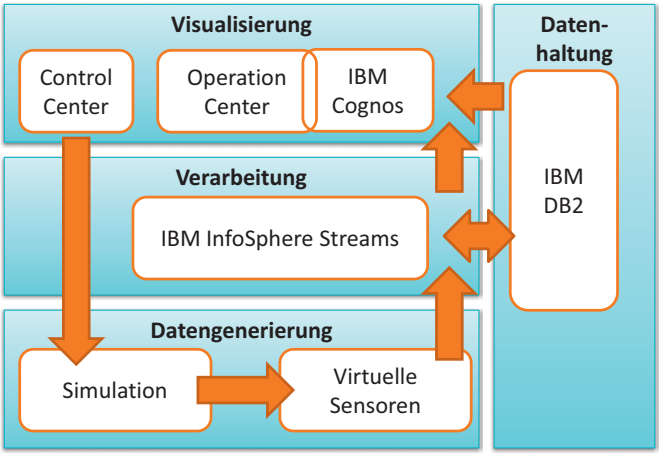


Abbildung 1: Architektur

Zur Speicherung historischer Daten wird eine IBM DB2 verwendet. Die Verarbeitung von Datenströmen wird mittels IBM InfoSphere Streams durchgeführt. Für die Visualisierung wird IBM Cognos genutzt, um Sichten und Statistiken zu erzeugen bspw. um die Verkehrsdichte von Hauptverkehrsknoten analysieren zu können. Des Weiteren wird Google Maps eingesetzt, um die Simulation und die dazugehörigen Sensordaten auf Karten anzuzeigen. Für die Simulierung der Beispielstadt werden Charakteristika von Echtdaten der Stadt Oldenburg (Oldb) hinzugezogen.

3 Demonstration

Gestartet, beeinflusst sowie überwacht werden die Simulationen durch zwei Webanwendungen, das Control Center und das Operation Center.

Das Control Center (CC) stellt den Startpunkt der Simulation dar und kann als geschützte Administrationskonsole beschrieben werden. Hier können entscheidende Parameter gesetzt werden, bevor die eigentliche Simulation gestartet wird (siehe Abbildung 2). Sobald die Simulation läuft, hat der Nutzer des CC die Möglichkeit die Simulation durch Ereignisse individuell zu beeinflussen, beispielsweise durch Variation der Objektanzahl oder auftretende Wetterereignisse. Die Auswirkungen dieser Ereignisse werden im Operation Center (OC) visualisiert. Das Modul stellt die wichtigsten Parameter der Simulation und ihrer Objekte übersichtlich dar. Zentral zeigt das OC eine Karte mit den Simulationsobjekten (siehe Abbildung 3). Außerdem können Detailinformationen der einzelnen Objekte abgefragt werden. Mit dem Gesamtsystem kann somit das Potential von Smart City-Anwendungen demonstriert werden. Zur Laufzeit kann die Dichte und Anzahl von Sensoren erhöht oder verringert werden, so dass sich beispielsweise Entscheider in einer Stadt ein Bild davon machen können, welche Informationsvorteile sie beispielsweise von der Integration oder Installation weiterer Sensoren in der Stadt erlangen können. Zudem können Smart City-Anwendungen auf diesem System auf Skalierbarkeit bezüglich der Aktualisierungsrate und der Anzahl der Sensoren getestet werden.

Abbildung 2: Control Center

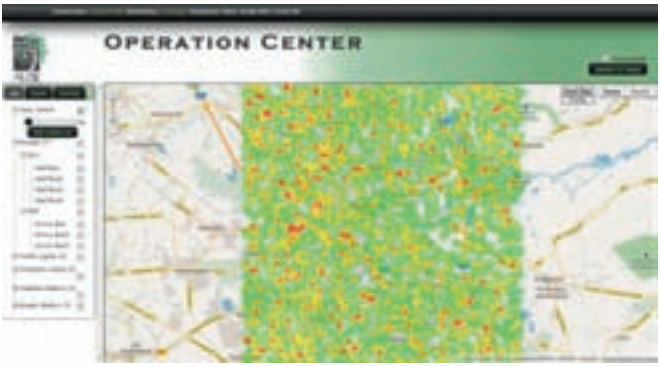


Abbildung 3: Operation Center

Das System soll nach derzeitigen Schätzungen 75.000 Verkehrsobjekte in einem Graphen mit 18.600 Knoten und 19.890 Kanten simulieren können. Der Durchsatz der von den 40.000 virtuellen Sensoren erzeugten Daten soll dabei etwa 100.000 Tupel der Form `<sensor_Id : INT32, sensor_type_Id : INT32, timestamp : INT64, measured_value : FLOAT32>` pro Sekunde betragen. Mit diesen angestrebten Leistungsdaten soll das zu entwickelnde System in nahezu Echtzeit laufen.

4 Zusammenfassung

Dieser Beitrag zeigt einen Überblick über die Hintergründe, den Aufbau und die Architektur einer Smarter City-Anwendung zur Erzeugung und Verarbeitung von virtuellen Sensordaten. Der Einsatz dieser Anwendung bietet nicht nur den Vorteil sensible Verkehrs-, Wetter- und Energiedaten zu generieren, sondern ermöglicht zusätzlich die Auswirkungen von Ereignissen, wie z. B. starken Niederschlag, zu betrachten. Diese Funktionen können vom Benutzer mit dem Control Center gesteuert und durch das Operation Center analysiert werden. Zusätzlich können Belastungstests von Datenstrommanagementsystemen durchgeführt und angezeigt werden.

Literatur

[BDG09] Behr, T.; Düntgen, C.; Güting, R.: BerlinMOD: A Benchmark for Moving Object Databases. In: The VLDB Journal 18:6 (2009), 1335-1368.

[Bri02] Brinkhoff, T.: A Framework for Generating Network-Based Moving Objects. In: GeoInformatica, Vol. 6, No. 2, 2002, pp. 153-180.

[CIS12] Cisco, Smart+Connected Communities: http://www.cisco.com/web/strategy/smart_connected_communities.html (24.09.2012)

[IBM12] IBM, Ein Planet der intelligenten Städte: http://www.ibm.com/smarterplanet/de/de/overview/visions/index.html (24.09.2012)

[SIE12] Siemens, Smarter Neighborhoods Smarter City: http://www.usa.siemens.com/sustainable-cities/pdf/smarter-neighborhoods-smarter-city.pdf (24.09.2012)

ProQua: Ein Probabilistisches Datenbanksystem für die Auswertung von Ähnlichkeitsanfragen auf unsicheren Datengrundlagen

Sebastian Lehrack, Sascha Saretz und Christian Winkel

Brandenburgische Technische Universität Cottbus
Institut für Informatik
Postfach 10 13 44
D-03013 Cottbus, Deutschland
{slehrack, ssaretz, cwinkel}@informatik.tu-cottbus.de

Abstract: *ProQua* ist ein neuartiges probabilistisches Datenbanksystem, welches die Auswertung von gewichteten logikbasierten Ähnlichkeitsbedingungen auf einer unsicheren Datenbasis zum Ziel hat. Die wesentlichen Leistungsmerkmale von ProQua werden anhand eines Bespielszenarios aus dem Umfeld der Archäologie präsentiert.

1 Motivation

Das neu entwickelte probabilistische Datenbanksystem *ProQua*[1] wurde als Kombination von Information Retrieval-Techniken und Datenbanktechnologien entworfen. Führende Datenbankforscher haben eine solche Verknüpfung im letzten Claremont-Report[2] [Agr08] als ein wichtiges Forschungsziel formuliert.

In der Vergangenheit haben traditionelle Datenbanksysteme eine Anfrage gegen ein einzelnes Datentupel entweder zu *wahr* oder *falsch* ausgewertet. Diese sehr restriktive Art der Auswertung kann jedoch die Anfragebedürfnisse vieler Anwender bezüglich Vagheit und unsicheren Bedingungen nicht erfüllen.

Ein leistungsfähiger Ansatz für die Integration von Ungenauigkeit und Ähnlichkeit stellen logikbasierte Anfragesprachen dar, welche Ähnlichkeitsprädikate der Art „Preis ist möglichst niedrig“ bzw. „Alter ist um 50 Jahre“ einbeziehen. Datentupel erfüllen die so gebildeten *komplexen* Ähnlichkeitsbedingungen mit einem bestimmten *Score-Wert* aus dem Intervall $[0; 1]$, der den jeweiligen Grad der Erfüllung repräsentiert.

Logikbasierte Ähnlichkeitsbedingungen können sowohl auf einer *sicheren*, als auch einer *unsicheren* Datengrundlagen ausgewertet werden [LSS11]. In der letzten Dekade sind *probabilistische Datenbanken* für die Verwaltung von großen unsicheren Datenbeständen in den Fokus der Forschung gerückt [SORK11]. In einer probabilistischen Datenbank wird konzeptionell jedes Datentupel mit einer Eintrittswahrscheinlichkeit annotiert. Sie drückt

[1] ProQua steht für probabilistisch und quantenlogisch-basiertes Datenbanksystem.
[2] Das Database Research Self-Assessment Meeting findet alle fünf Jahre statt.

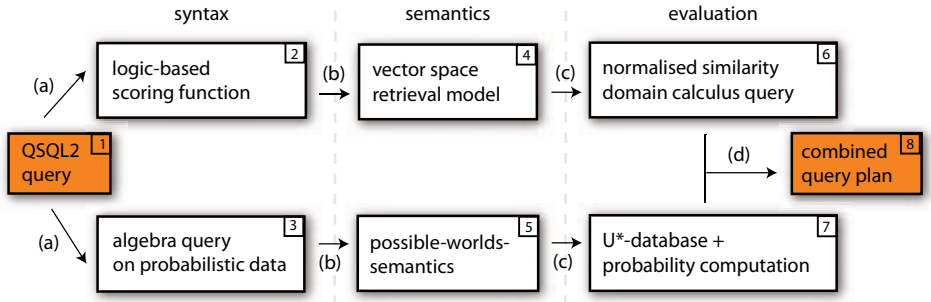


Abbildung 1: Grundkonzept von ProQua

aus mit welcher Wahrscheinlichkeit das jeweilige Tupel zu einer bestimmten Datentabelle bzw. zu einem berechneten Anfrageergebnis gehört.

In vorangegangenen Arbeiten [LS11a, LS11b] wurde ein erweitertes probabilistisches Datenmodell als Basis für die Entwicklung von ProQua präsentiert. ProQua ist das einzige probabilistische Datenbanksystem, das *komplexe* logikbasierte Ähnlichkeitsanfragen sowie die Gewichtung von Teilanfragen durch seiner Anfragesprache QSQL2 unterstützt [LSS12, LS10].

2 Grundlegende Konzepte

In diesem Abschnitt sollen die wesentlichen Grundkonzepte von ProQua, sowie deren Zusammenspiel, siehe Abb. (1), skizziert werden.

Der Startpunkt für die Anfrageverarbeitung ist eine gegebene QSQL2-Anfrage [LSS12, LS10]. Diese beruht im wesentlichen auf den bekannten SQL-Sprachkonstrukten. Zusätzlich können komplexe Ähnlichkeitsbedingungen und probabilistischen Tabellen verwendet worden sein. Zur Verarbeitung dieser Anfrage werden in einem ersten Schritt die entsprechenden syntaktischen QSQL2-Anfragekomponenten (i) in eine logikbasierte Bewertungsfunktion [LS12b] und (ii) in eine relationale Algebraanfrage abgebildet. Grundsätzlich basieren diese Anfrageklassen auf ihren eigenen semantischen Modellen. Auf der einen Seite wird der Semantik von logikbasierte Bewertungsfunktionen mittels einer probabilistischen Interpretation eines geometrischen Retrieval-Modells festgelegt [LS11a, LS11b]. Zum anderen wird die bekannte Possible-Worlds-Semantik für die Behandlung von Algebraanfragen auf probabilistischen Daten angewendet. Neben den Standardoperationen können beide Anfragetypen ebenfalls gewichtete Teilanfragen bzw. -bedingungen besitzen [Leh12a].

Auf der Grundlage des geometrischen Retrieval-Modells wird eine logikbasierte Bewertungsfunktion als eine normalisierte Bereichskalkülanfrage ausgewertet, welche ggf. um Ähnlichkeitsprädikate erweitert wurde [LS12b]. Dagegen beruht die Auswertung einer Algebraanfrage auf einem neuen Repräsentationsystem für probabilistische Datenbanken,

```
select aid, type, culture
from    (  select aid, culture
           from ArteExp
         union[ 0.9, 0.4 ]
           select aid, culture
           from ArteMat
        ) origin
     inner join
        ( select *
          from Arte
          where ( sond ~ 10 or[ 0.3,
                  0.8 ] age ~ 300 )
        ) prop
     on ( origin.aid = prop.aid )
```

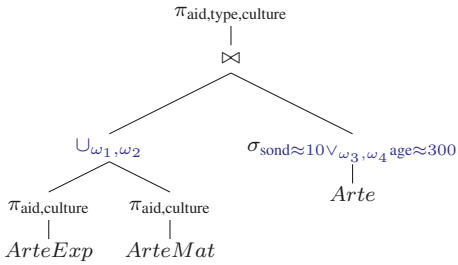


Abbildung 2: Beispielanfrage aus dem OpenInfRA-Szenario: QSQL2-Anfrage (links) und abgeleiteter Anfragebaum (rechts)

den sogenannten *U*-Datenbanken*. Diese nutzen u.a. Ereignismuster zur Verwaltung von komplexen Tupelereignissen [Leh12b]. Abschließend wird unter der Anwendung eines Top-k-Filters [LS12a] eine Reihenfolge der Antworttupel mittels eines kombinierten Anfrageplanes erzeugt.

3 Beispielszenario

Um die wesentlichen Anfragetypen von ProQua zu demonstrieren wurde ein Online-Demo unter

http://dbis.informatik.tu-cottbus.de/ProQua/

zur Verfügung gestellt. Das hier verwendete Beispielszenario ist durch die Neuentwicklung des CISAR-Projektes[3] motiviert, welches als Internet-basiertes Geo-Informationssystem für Archäologie und Gebäudegeschichte entwickelt worden ist. Die in ProQua entwickelten Technologien werden umfassend in dem Nachfolgesystem *OpenInfRA* eingesetzt.

In dem stark vereinfachten Anwendungsbeispiel werden die deterministische Tabelle *Artefacts* (*Arte*) und die zwei probabilistischen Tabellen *Artefacts classified by experts* (*ArteExp*) und *Artefacts classified by material* (*ArteMat*) verwendet. In der Datentabelle *Arte* werden Informationen über mehrere Artefakte gespeichert, die während einer archäologischen Ausgrabung gefunden worden sind. Dabei wird mittels der *Sondage*-Nummer (Attribut *sond*) die geographische Fundstelle eines Artefaktes beschrieben.

Des Weiteren gaben mehrere Experten eine Expertise über die Ursprungskultur für ein Artefakt in der Tabelle *ArteExp* ab. Diese Zuordnungen werden durch einen Konfidenzwert aus dem Intervall $[0; 1]$ quantifiziert.

Neben diesen subjektiven Bewertungen werden auch objektive Methoden für die Bestimmung der Ursprungskultur eingesetzt. Diese *archäometrischen Verfahren* vertrauen dabei auf verschiedene Verfahren der Materialanalyse, siehe Tabelle *ArteMat*.

[3] http://www.dainst.org/en/project/cisar/

Basierend auf diesen Tabellen werden im Online-Demo verschiedene Beispielanfragen diskutiert. Unter anderem kann folgende Anfrage an die Beispieldatenbank gestellt und evaluiert werden: *Bestimme alle Artefakte mit ihren möglichen Ursprungskulturen. Dabei soll sich die entsprechende Fundstelle in der Nähe der Sondage* 10 *befinden bzw. das Artefaktalter soll ungefähr* 300 *Jahre betragen.*

Zusätzlich kann der Anwender den Einfluss verschiedener Teilanfragen und -bedingungen durch gewichtete Operatoren, wie z. B. $\mathrm{and}[\theta_1, \theta_2]$, individualisieren. Die Gewichtsvariablen θ_i kommen dabei aus dem Intervall $[0; 1]$, wobei 0 für *keine* und 1 für *volle* Relevanz der Teilanfrage steht. Die Gewichtung $[1, 1]$ ist somit äquivalent zum jeweils ungewichteten Fall. In Abb. (2) sind eine entsprechende QSQL2-Anfrage sowie die abgeleitete Anfragestruktur in Form eines Anfragebaumes zu sehen. Dort wird bei der Vereinigung in der ersten Unterabfrage die Expertenmeinung im Verhältnis $0.9 : 0.4$ gegenüber der materiellen Analyse favorisiert.

Danksagung: Sebastian Lehrack wurde innerhalb der Projekte SCHM 1208/11-1 und SCHM 1208/11-2 von der Deutschen Forschungsgesellschaft unterstützt.

Literatur

[Agr08] Agrawal et al. The Claremont report on database research. *SIGMOD Rec.*, 37:9–19, September 2008.

[Leh12a] Sebastian Lehrack. Applying Weighted Queries on Probabilistic Databases. In *CIKM*, 2012.

[Leh12b] Sebastian Lehrack. Ereignismuster für die Verwaltung von komplexen Tupelereignissen in Probabilistischen Datenbanken. In *Grundlagen von Datenbanken*, Seiten 65–70, 2012.

[LS10] Sebastian Lehrack und Ingo Schmitt. QSQL: Incorporating Logic-Based Retrieval Conditions into SQL. In *DASFAA*, Seiten 429–443, 2010.

[LS11a] Sebastian Lehrack und Ingo Schmitt. A Probabilistic Interpretation for a Geometric Similarity Measure. In *ECSQARU*, Seiten 749–760, 2011.

[LS11b] Sebastian Lehrack und Ingo Schmitt. A Unifying Probability Measure for Logic-Based Similarity Conditions on Uncertain Relational Data. In *NTSS*, Seiten 14–19, 2011.

[LS12a] Sebastian Lehrack und Sascha Saretz. A Top-k Filter for Logic-Based Similarity Conditions on Probabilistic Databases. In *ADBIS*, Seiten 268–281, 2012.

[LS12b] Sebastian Lehrack und Sascha Saretz. Evaluating Logic-Based Scoring Functions on Uncertain Relational Data. *JIDM*, 3(3):348–363, 2012.

[LSS11] Sebastian Lehrack, Sascha Saretz und Ingo Schmitt. QSQLp: Eine Erweiterung der probabilistischen Many-World-Semantik um Relevanzwahrscheinlichkeiten. In *BTW*, Seiten 494–513, 2011.

[LSS12] Sebastian Lehrack, Sascha Saretz und Ingo Schmitt. QSQL2: Query Language Support for Logic-Based Similarity Conditions on Probabilistic Databases. In *RCIS*, Seiten 1–12, 2012.

[SORK11] Dan Suciu, Dan Olteanu, Christopher Ré und Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

PeRA: Individual Privacy Control in Intelligent Transportation Systems

Martin Kost, Raffael Dzikowski, Johann-Christoph Freytag

DBIS Group
Humboldt-Universität zu Berlin
[kost|dzikowsk|freytag]@informatik.hu-berlin.de

Abstract: In the domain of Intelligent Transportation Systems (ITS) manufacturers and service providers start to implement and deploy plenty of (new) applications running on a vehicle. These applications involve the user and external services. Therefore, we must incorporate mechanisms providing the individual for controlling his/her privacy. Existing approaches only consider to control the event of data access using a central instance. In contrast, we consider to implement individual privacy requirements for the complete data flow of distributed systems. The Privacy-enforcing Runtime Architecture (PeRA) provides a holistic privacy protection approach, which implements user-defined privacy policies. A data-centric protection chain ensures that ITS components process data according to attached privacy policies. PeRA instances constitute a distributed privacy middleware, which evaluates privacy policies to mediate data access by applications. The PeRA architecture includes an integrity protection layer to create a distributed policy enforcement perimeter between ITS nodes, which prevents the circumvention of policies. We implemented the PeRA architecture as a proof-of-concept prototype.

1 Introduction

Designing and implementing co-operative mobile systems that comply with current and future privacy regulations is a great challenge today. Evolving Intelligent Transportation Systems (ITS) provide cooperative applications which implement an improved functionality such as enhanced travel services, driving support, and transportation optimization. These applications exchange information about participating individuals (e. g., vehicle owners and drivers); thus, impacting the privacy of persons. Uncontrolled information flows potentially allow for privacy infringements (e. g., generating movement profiles).

For identifying possible privacy threats and appropriate protection requirements, we analyzed and applied domain independent privacy principles on the ITS domain; especially the functional requirements and processes [Die12]. Thereby, one of the challenges which we address is to prevent an attacker from circumventing the defined privacy constraints within a distributed system. For instance, if we apply a policy enforcement mechanism we have to guarantee that the policies of the individuals as well as the application code will not be manipulated. Additionally, we have to guarantee the privacy-compliant execution

of applications which consists of (standard and user-defined) operations. For addressing these challenges our approach requires privacy policies resulting from a comprehensive privacy analysis. Based on existing solutions such as *Hippocratic databases* [AKSX02], we designed and implemented the privacy middleware *Privacy-enforcing Runtime Architecture* (PeRA) which realizes the identified requirements.

In the following, we introduce the concepts of our ITS privacy middleware PeRA and describe our demonstration scenario. Our demo setting consists of the ITS nodes (1) motorcar, (2) truck, (3) Road Side Unit (RSU), and (4) traffic control center. We implemented and distributed the ITS applications (a) Moving Map, (b) Intersection Collision Detection, (c) Traffic Status, and (d) Fleet Management on these nodes. Running the scenarios we will demonstrate—by visualizing the resulting effects—how an individual may configure policies in order to control the processing of his/her data within this distributed system.

2 PeRA Concepts

Most technical proposals for privacy preservation in ITS only support single applications. We developed a policy-based privacy enforcement architecture which provides an application independent privacy middleware for ITS. Moreover, current solutions for protecting privacy implement mechanisms to control the event of accessing data from a central instance such as a database management system. In contrast, our architecture PeRA controls data processing for the complete data flow; i.e., we include events such as data communication and data processing by different applications or remote nodes.

PeRA provides a policy enforcement perimeter that realizes a data-centric approach for privacy protection. Subjects get control of their data by declaring/configuring privacy policies which restrict how applications may process their data. For instance, the following policy statement specifies a context—which is defined by a set of constraints—together with operations which are permitted within this context, and a reference to the new policy.

```
Policy-ID="Example-Privacy-Policy-1"
Context(node-type="traffic control center" and ...) {
 Permit access On location From db.vehicle With
    PostCondition(anonymity-value="10")
 Post-Policy="Example-Post-Policy-A"}
```

All data is combined with an immutable set of privacy policies upon creation; e. g., we couple all GPS data of a vehicle which is sent to the traffic control center and stored in its local database *db* with the previous described policy *Example-Privacy-Policy-1*.

Mandatory privacy control (MPC) components ensure that applications only perform policy compliant operations on the data. To prevent data processors from circumventing the MPC, we introduce the MPC integrity protection (MIP) layer [KWD+11]. The MIP layer stores data securely and encrypts data for information exchange between PeRA instances. It monitors the integrity of MPC components and only grants data access if all MPC components are in a trusted state. The MPC components mediate all data access and processing. An application poses an operation request as a query to the Privacy Control Monitor (PCM). The PCM evaluates the privacy policies of affected data items. Based on the eval-

uation result, a query is rejected or executed. Also, the MPC may perform additional data transformations on the data to meet the privacy requirements specified in attached policies. For instance, a result set might be perturbed to ensure a certain anonymity set size. We guarantee for all data that leaves the PCM to be policy compliant. However, once outside the control of the PeRA, we cannot guarantee policy enforcement anymore. Therefore, external applications may not gain data access on the required level of detail. Thus, we integrated an application sandbox, the Controlled Application Environment [Die12] (CAE). Application parts running inside the CAE are heavily restricted in their communication and resource access capabilities. Controlled applications may have detailed data access, which is mediated and controlled by the CAE and PCM.

3 ITS Demonstration Scenario

We use a real world ITS scenario to demonstrate the privacy protection capabilities of the PeRA prototype. The current scenario, depicted in Figure 1, is an extension of a previous demonstration [KWD+11]. Each of the nodes (vehicles, RSU, and the traffic control center) runs a separate PeRA instance.

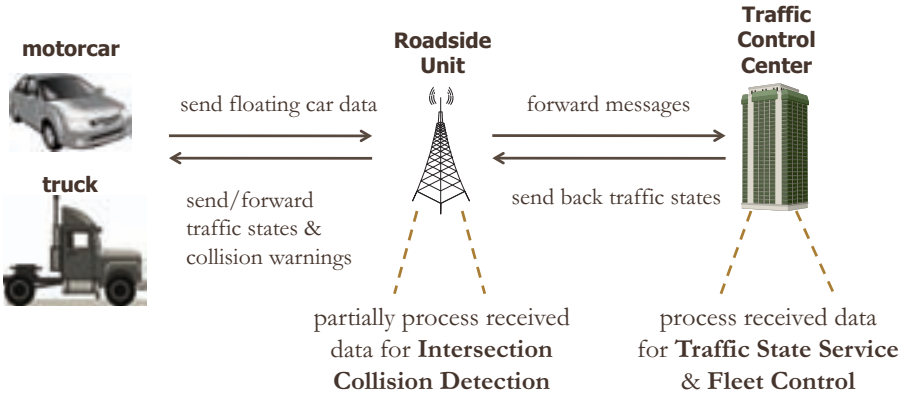


Figure 1: A Visualization of the ITS Demonstration Scenario.

In our setup, each vehicle runs a *moving map* application which displays the current location of the vehicle on a map. Furthermore, the vehicles send floating car data (FCD) records (of pre-recorded location tracks) to the RSU. These FCD records contain information about location, time, traffic status (light, normal, dense), license plate, and cargo. The RSU mediates the communication between vehicles and the traffic control center and provides an intersection collision detection (ICD) functionality. Thereby, the *ICD* application sends collision warnings to impacting vehicles and the *moving map* application of the RSU displays vehicles in its range. In the backend, the traffic control center provides traffic information/management services. A *vehicle tracking* application requests single FCD records to provide real-time tracking of single vehicles on a map. Fleet management

and freight tracking are typical purposes for such an application. Furthermore, a *traffic status* application provides a map showing the real-time traffic situation in a road network.

For all personal information the drivers may dynamically configure their privacy preferences. We provide an abstract user interface with a slider (for selecting a privacy protection level), check boxes, and textual descriptions in order to simplify the policy specification. The interface provides the following options: (1) permit all data processing; (2) permit only the specified data flow of the scenario; (3) no data processing is permitted; (4) individual privacy configuration: a) the single applications at the different nodes are permitted or not, b) full details for fleet management on/off. User settings are subsequently translated into corresponding privacy policies which the PeRA instance then permanently attaches to new/imported data. Thus, the user determines how applications can use the submitted FCD records. PeRA instances evaluate the attached privacy policies and permit an execution of policy compliant operations on the data only. In general, policies describe the permitted operations on certain data items, possibly including additional constraints, such as a certain degree of obfuscation.

The demo system offers two views on the showcased scenario to foster better understanding of the prototype's policy enforcement concepts and effects as well as architecture internals. The *application views* visualize what data is available to specific applications with different purposes. The *information flow view* shows the processing flow inside a PeRA instance and effects of policy-based decisions. Given a request, we construct the corresponding data flow graph enhanced with selected privacy properties.

4 Conclusions

In this paper, we address the challenge of designing and implementing ITS applications for distributed systems in a privacy protecting manner. The described PeRA concepts solve this issue by providing a privacy middleware that gives the individuals the control about his/her data. Our implemented prototype ensures privacy policy compliant data processing throughout an Intelligent Transportation System. Using an ITS scenario we show how PeRA supports a wide range of ITS applications and illustrate the fine-grained control mechanisms of our privacy policy enforcement.

References

[AKSX02] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic Databases. In *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.

[Die12] Dietzel, S. et al. CANE: A Controlled Application Environment for Privacy Protection in ITS. In *12th Int. Conf. on ITS Telecommunications (ITST 2012)*. IEEE, 2012.

[KWD+11] M. Kost, B. Wiedersheim, S. Dietzel, F. Schaub, and T. Bachmor. PRECIOSA PeRA: Practical Enforcement of Privacy Policies in Intelligent Transportation Systems. In *Proc. of the Demo. Session at the Fourth ACM Conf. on Wireless Network Sec.*, 2011.

GI-Edition Lecture Notes in Informatics

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Lan-guage to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Business 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Er-fahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometrics and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment

P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society

P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications

P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven

P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik

P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004

P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration

P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Adhoc-Netze Medienintegration

P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government

P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen

P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality

P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Ranneberg, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments

P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für inforrmatische Bildung

P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen

P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit

P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and ist Applications

P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005

P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolffried Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web

P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik

P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)

P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)

P-69 Robert Hirschfeld, Ryszard Kowalcyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005

P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)

P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005

P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): “Heute schon das Morgen sehen“

P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology

P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture

P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz

P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006

P-78 K.-O. Wenkel, P. Wagner, M. Morgenstern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce

P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS´06

P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006

P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics

P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications

P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems

P-86 Robert Krimmer (Ed.): Electronic Voting 2006

P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik

P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006

P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur

P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006

P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006

P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1

P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2

P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen

P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies

P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices

P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik

P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik

P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten

P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments

P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)

P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme

P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007

P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops

P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications

P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures

P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1

P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2

P-111 Christian Eibl, Johannes Magenheim, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik

P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis

P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)

P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics

P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walther (Eds.) German conference on bioinformatics GCB 2007

P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007

P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and manegement The 4th International Conference on Grid Service Engineering and Management GSEM 2007

P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007

P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications

P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges

P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik

P-122 Walid Maalej, Bernd Bruegge (Hrsg.) Software Engineering 2008 - Workshopband Fachtagung des GI-Fachbereichs Softwaretechnik

P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)

P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Speciel Interest Groups PARS and PARVA

P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT:
Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung

P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)

P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008

P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany

P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems

P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung

P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting. CC

P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik

P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 1

P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 2

P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik –
Aktuelle Forschungsergebnisse

P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008

P-137 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures

P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008

P-139 Georg Herzwurm, Martin Mikusz (Hrsg.)
Industrialisierung des Software-Managements
Fachtagung des GI-Fachausschusses Management der Anwendungsentwick-lung und -wartung im Fachbereich Wirtschaftsinformatik

P-140 Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)
IMF 2008 - IT Incident Management & IT Forensics

P-141 Peter Loos, Markus Nüttgens, Klaus Turowski, Dirk Werth (Hrsg.)
Modellierung betrieblicher Informations-systeme (MobIS 2008)
Modellierung zwischen SOA und Compliance Management

P-142 R. Bill, P. Korduan, L. Theuvsen, M. Morgenstern (Hrsg.)
Anforderungen an die Agrarinformatik durch Globalisierung und Klimaveränderung

P-143 Peter Liggesmeyer, Gregor Engels, Jürgen Münch, Jörg Dörr, Norman Riegel (Hrsg.)
Software Engineering 2009
Fachtagung des GI-Fachbereichs Softwaretechnik

P-144 Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)
Datenbanksysteme in Business, Technologie und Web (BTW)

P-145 Knut Hinkelmann, Holger Wache (Eds.)
WM2009: 5th Conference on Professional Knowledge Management

P-146 Markus Bick, Martin Breunig, Hagen Höpfner (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Entwicklung, Implementierung und Anwendung
4. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2009)

P-147 Witold Abramowicz, Leszek Maciaszek, Ryszard Kowalczyk, Andreas Speck (Eds.)
Business Process, Services Computing and Intelligent Service Management
BPSC 2009 · ISM 2009 · YRW-MBP 2009

P-148 Christian Erfurth, Gerald Eichler, Volkmar Schau (Eds.)
9th International Conference on Innovative Internet Community Systems
I^{2}CS 2009

P-149 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
2. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-150 Jürgen Münch, Peter Liggesmeyer (Hrsg.)
Software Engineering
2009 - Workshopband

P-151 Armin Heinzl, Peter Dadam, Stefan Kirn, Peter Lockemann (Eds.)
PRIMIUM
Process Innovation for Enterprise Software

P-152 Jan Mendling, Stefanie Rinderle-Ma, Werner Esswein (Eds.)
Enterprise Modelling and Information Systems Architectures
Proceedings of the 3rd Int'l Workshop EMISA 2009

P-153 Andreas Schwill, Nicolas Apostolopoulos (Hrsg.)
Lernen im Digitalen Zeitalter
DeLFI 2009 – Die 7. E-Learning Fachtagung Informatik

P-154 Stefan Fischer, Erik Maehle Rüdiger Reischuk (Hrsg.)
INFORMATIK 2009
Im Focus das Leben

P-155 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2009:
Biometrics and Electronic Signatures
Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-156 Bernhard Koerber (Hrsg.)
Zukunft braucht Herkunft
25 Jahre »INFOS – Informatik und Schule«

P-157 Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, Peter Stadler (Eds.)
German Conference on Bioinformatics 2009

P-158 W. Claupein, L. Theuvsen, A. Kämpf, M. Morgenstern (Hrsg.)
Precision Agriculture Reloaded – Informationsgestützte Landwirtschaft

P-159 Gregor Engels, Markus Luckey, Wilhelm Schäfer (Hrsg.)
Software Engineering 2010

P-160 Gregor Engels, Markus Luckey, Alexander Pretschner, Ralf Reussner (Hrsg.)
Software Engineering 2010 – Workshopband
(inkl. Doktorandensymposium)

P-161 Gregor Engels, Dimitris Karagiannis Heinrich C. Mayr (Hrsg.)
Modellierung 2010

P-162 Maria A. Wimmer, Uwe Brinkhoff, Siegfried Kaiser, Dagmar Lück-Schneider, Erich Schweighofer, Andreas Wiebe (Hrsg.)
Vernetzte IT für einen effektiven Staat
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2010

P-163 Markus Bick, Stefan Eulgem, Elgar Fleisch, J. Felix Hampe, Birgitta König-Ries, Franz Lehner, Key Pousttchi, Kai Rannenberg (Hrsg.)
Mobile und Ubiquitäre Informationssysteme
Technologien, Anwendungen und Dienste zur Unterstützung von mobiler Kollaboration

P-164 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2010: Biometrics and Electronic Signatures Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-165 Gerald Eichler, Peter Kropf, Ulrike Lechner, Phayung Meesad, Herwig Unger (Eds.)
10th International Conference on Innovative Internet Community Systems (I^2CS) – Jubilee Edition 2010 –

P-166 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
3. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-167 Robert Krimmer, Rüdiger Grimm (Eds.)
4th International Conference on Electronic Voting 2010
co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-168 Ira Diethelm, Christina Dörge, Claudia Hildebrandt, Carsten Schulte (Hrsg.)
Didaktik der Informatik
Möglichkeiten empirischer Forschungsmethoden und Perspektiven der Fachdidaktik

P-169 Michael Kerres, Nadine Ojstersek Ulrik Schroeder, Ulrich Hoppe (Hrsg.)
DeLFI 2010 - 8. Tagung der Fachgruppe E-Learning der Gesellschaft für Informatik e.V.

P-170 Felix C. Freiling (Hrsg.)
Sicherheit 2010
Sicherheit, Schutz und Zuverlässigkeit

P-171 Werner Esswein, Klaus Turowski, Martin Juhrisch (Hrsg.)
Modellierung betrieblicher Informationssysteme (MobIS 2010)
Modellgestütztes Management

P-172 Stefan Klink, Agnes Koschmider Marco Mevius, Andreas Oberweis (Hrsg.)
EMISA 2010
Einflussfaktoren auf die Entwicklung flexibler, integrierter Informationssysteme
Beiträge des Workshops der GI-Fachgruppe EMISA (Entwicklungsmethoden für Informationssysteme und deren Anwendung)

P-173 Dietmar Schomburg, Andreas Grote (Eds.)
German Conference on Bioinformatics 2010

P-174 Arslan Brömme, Torsten Eymann, Detlef Hühnlein, Heiko Roßnagel, Paul Schmücker (Hrsg.)
perspeGKtive 2010
Workshop „Innovative und sichere Informationstechnologie für das Gesundheitswesen von morgen“

P-175 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 1

P-176 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 2

P-177 Witold Abramowicz, Rainer Alt, Klaus-Peter Fähnrich, Bogdan Franczyk, Leszek A. Maciaszek (Eds.)
INFORMATIK 2010
Business Process and Service Science – Proceedings of ISSS and BPSC

P-178 Wolfram Pietsch, Benedikt Krams (Hrsg.)
Vom Projekt zum Produkt
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschafts-informatik (WI-MAW), Aachen, 2010

P-179 Stefan Gruner, Bernhard Rumpe (Eds.)
FM+AM`2010
Second International Workshop on Formal Methods and Agile Methods

P-180 Theo Härder, Wolfgang Lehner, Bernhard Mitschang, Harald Schöning, Holger Schwarz (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW)
14. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS)

P-181 Michael Clasen, Otto Schätzel, Brigitte Theuvsen (Hrsg.)
Qualität und Effizienz durch informationsgestützte Landwirtschaft, Fokus: Moderne Weinwirtschaft

P-182 Ronald Maier (Hrsg.)
6th Conference on Professional Knowledge Management
From Knowledge to Action

P-183 Ralf Reussner, Matthias Grund, Andreas Oberweis, Walter Tichy (Hrsg.)
Software Engineering 2011
Fachtagung des GI-Fachbereichs Softwaretechnik

P-184 Ralf Reussner, Alexander Pretschner, Stefan Jähnichen (Hrsg.)
Software Engineering 2011
Workshopband
(inkl. Doktorandensymposium)

P-185 Hagen Höpfner, Günther Specht, Thomas Ritz, Christian Bunse (Hrsg.)
MMS 2011: Mobile und ubiquitäre Informationssysteme Proceedings zur 6. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2011)

P-186 Gerald Eichler, Axel Küpper, Volkmar Schau, Hacène Fouchal, Herwig Unger (Eds.)
11th International Conference on Innovative Internet Community Systems (I^2CS)

P-187 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
4. DFN-Forum Kommunikations-technologien, Beiträge der Fachtagung 20. Juni bis 21. Juni 2011 Bonn

P-188 Holger Rohland, Andrea Kienle, Steffen Friedrich (Hrsg.)
DeLFI 2011 – Die 9. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. 5.–8. September 2011, Dresden

P-189 Thomas, Marco (Hrsg.)
Informatik in Bildung und Beruf
INFOS 2011
14. GI-Fachtagung Informatik und Schule

P-190 Markus Nüttgens, Oliver Thomas, Barbara Weber (Eds.)
Enterprise Modelling and Information Systems Architectures (EMISA 2011)

P-191 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2011
International Conference of the Biometrics Special Interest Group

P-192 Hans-Ulrich Heiß, Peter Pepper, Holger Schlingloff, Jörg Schneider (Hrsg.)
INFORMATIK 2011
Informatik schafft Communities

P-193 Wolfgang Lehner, Gunther Piller (Hrsg.)
IMDM 2011

P-194 M. Clasen, G. Fröhlich, H. Bernhardt, K. Hildebrand, B. Theuvsen (Hrsg.)
Informationstechnologie für eine nachhaltige Landbewirtschaftung
Fokus Forstwirtschaft

P-195 Neeraj Suri, Michael Waidner (Hrsg.)
Sicherheit 2012
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 6. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)

P-196 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2012
Proceedings of the 11th International Conference of the Biometrics Special Interest Group

P-197 Jörn von Lucke, Christian P. Geiger, Siegfried Kaiser, Erich Schweighofer, Maria A. Wimmer (Hrsg.)
Auf dem Weg zu einer offenen, smarten und vernetzten Verwaltungskultur
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2012

P-198 Stefan Jähnichen, Axel Küpper, Sahin Albayrak (Hrsg.)
Software Engineering 2012
Fachtagung des GI-Fachbereichs Softwaretechnik

P-199 Stefan Jähnichen, Bernhard Rumpe, Holger Schlingloff (Hrsg.)
Software Engineering 2012
Workshopband

P-200 Gero Mühl, Jan Richling, Andreas Herkersdorf (Hrsg.)
ARCS 2012 Workshops

P-201 Elmar J. Sinz Andy Schürr (Hrsg.)
Modellierung 2012

P-202 Andrea Back, Markus Bick, Martin Breunig, Key Pousttchi, Frédéric Thiesse (Hrsg.)
MMS 2012:Mobile und Ubiquitäre Informationssysteme

P-203 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
5. DFN-Forum Kommunikations-technologien
Beiträge der Fachtagung

P-204 Gerald Eichler, Leendert W. M. Wienhofen, Anders Kofod-Petersen, Herwig Unger (Eds.)
12th International Conference on Innovative Internet Community Systems (I2CS 2012)

P-205 Manuel J. Kripp, Melanie Volkamer, Rüdiger Grimm (Eds.)
5th International Conference on Electronic Voting 2012 (EVOTE2012)
Co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-206 Stefanie Rinderle-Ma, Mathias Weske (Hrsg.)
EMISA 2012
Der Mensch im Zentrum der Modellierung

P-207 Jörg Desel, Jörg M. Haake, Christian Spannagel (Hrsg.)
DeLFI 2012: Die 10. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V.
24.–26. September 2012

P-208 Ursula Goltz, Marcus Magnor, Hans-Jürgen Appelrath, Herbert Matthies, Wolf-Tilo Balke, Lars Wolf (Hrsg.) INFORMATIK 2012

P-209 Hans Brandt-Pook, André Fleer, Thorsten Spitta, Malte Wattenberg (Hrsg.) Nachhaltiges Software Management

P-210 Erhard Plödereder, Peter Dencker, Herbert Klenk, Hubert B. Keller, Silke Spitzer (Hrsg.) Automotive – Safety & Security 2012 Sicherheit und Zuverlässigkeit für automobile Informationstechnik

P-211 M. Clasen, K. C. Kersebaum, A. Meyer-Aurich, B. Theuvsen (Hrsg.) Massendatenmanagement in der Agrar- und Ernährungswirtschaft Erhebung - Verarbeitung - Nutzung Referate der 33. GIL-Jahrestagung 20. – 21. Februar 2013, Potsdam

P-213 Stefan Kowalewski, Bernhard Rumpe (Hrsg.) Software Engineering 2013 Fachtagung des GI-Fachbereichs Softwaretechnik

P-214 Volker Markl, Gunter Saake, Kai-Uwe Sattler, Gregor Hackenbroich, Bernhard Mit schang, Theo Härder, Veit Köppen (Hrsg.) Datenbanksysteme für Business, Technologie und Web (BTW) 2013 13. – 15. März 2013, Magdeburg

P-215 Stefan Wagner, Horst Lichter (Hrsg.) Software Engineering 2013 Workshopband (inkl. Doktorandensymposium) 26. Februar – 1. März 2013, Aachen

The titles can be purchased at:

Köllen Druck + Verlag GmbH
Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn
Fax: +49 (0)228/9898222
E-Mail: druckverlag@koellen.de

Gesellschaft für Informatik e.V. (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in cooperation with GI and to publish the annual GI Award dissertation.

Broken down into
• seminars
• proceedings
• dissertations
• thematics
current topics are dealt with from the vantage point of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure high quality contributions.

The volumes are published in German or English.

Information: http://www.gi.de/service/publikationen/lni/

ISSN 1617-5468
ISBN 978-3-88579-608-4

The BTW 2013 in Magdeburg is the 15th conference of its kind reflecting the broad range of academic research and industrial development work within the German database community. This year's conference focuses on a broad range of database topics covering information extraction and integration, data analytics, web data management, service-oriented Architectures, cloud computing ,and virtualization. This volume contains contributions from the refereed scientific program, the refereed industrial program, and the refereed demo program. Furthermore, the dissertation award winner and three keynotes are presented in this volume.