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 co-operation with GI and to publish the annual GI Award dissertation.

Broken down into the fields of
• Seminar,
• Proceedings
• Dissertations
• Thematics
current topics are dealt with from the fields of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure the high level of the contributions.

The volumes are published in German or English

Information: http://www.gi-ev.de/service/publikationen/lni/

The BTW 2007 conference on Database Systems in Business, Technology and Web took place in Aachen from March 7th to 9th. This volume contains 12 long as well as 12 short papers selected for presentation at the conference, 8 industrial contributions, 3 papers or abstracts for the invited talks, 8 contributions to the demonstrations program, and 2 short papers written by the winners of the two dissertation awards. To assure scientific quality, the selection was based on a strict and anonymous review process. The subject areas covered by the papers include query processing, data exploration, indexing, Web and XML, query optimization, evolution and integration, distributed databases, data transformation, mobile and grid applications.
Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.)

Datenbanksysteme in Business, Technologie und Web (BTW)

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

07.-09.03.2007
in Aachen

Gesellschaft für Informatik e.V. (GI)
**Vorwort**


Im wissenschaftlichen Bereich wurden folgende Themen adressiert:

- Anfrageverarbeitung und Datenexploration
- Indexierung
- Modellierung
- Web / XML
- Anfrageoptimierung
- Evolution und Integration
Das Industrieprogramm behandelte folgende Bereiche:

- Verteilte Datenbanken
- Datentransformation und Anfrageoptimierung
- Mobile und Grid-Anwendungen

Nach guter Tradition wurden im Rahmen der BTW auch zwei Preise für hervorragende Dissertationen im Datenbankbereich vergeben:

Dr. Jens Teubner wurde für seine von Prof. Torsten Grust an der TU München betreute Dissertation “Pathfinder: XQuery Compilation Techniques for Relational Database Targets” ausgezeichnet.

Dr. Martin Theobald bekam die Auszeichnung für seine von Prof. Gerhard Weikum am Max-Planck-Institut für Informatik in Saarbrücken betreute Arbeit über “TopX: Efficient and Versatile Top-k Query Processing for Text, Structured, and Semistructured Data”.


Aachen, Darmstadt, München, im Januar 2007

Alfons Kemper, Vorsitzender des Programmkomitees
Harald Schöning, Vorsitzender des Industriekomitees
Thomas Rose, Vorsitzender des Demonstrationskomitees
Matthias Jarke und Thomas Seidl, Tagungsleitung
Christoph Brochhaus und Christoph Quix, Organisationskomitee
Tagungsleitung:
Matthias Jarke, RWTH Aachen und Fraunhofer FIT
Thomas Seidl, RWTH Aachen

Organisationskomitee
Christoph Brochhaus, RWTH Aachen
David Kensche, RWTH Aachen
Hedi Klee, RWTH Aachen

Thomas Rose, Fraunhofer FIT
Christoph Quix, RWTH Aachen
Irene Wicke, RWTH Aachen

Organisationskomitee
Christoph Brochhaus, RWTH Aachen
David Kensche, RWTH Aachen
Hedi Klee, RWTH Aachen

Thomas Rose, Fraunhofer FIT
Christoph Quix, RWTH Aachen
Irene Wicke, RWTH Aachen

Studierendengesellschaft
Stefan Conrad, Univ. Düsseldorf
Hagen Höpfner, IU Bruchsal

Koordination Workshops
Klaus Turowski, Univ. Augsburg

Technische Unterstützung
Elmar Berger, Fraunhofer FIT (Conference Management Toolkit)
Klemens Böhm, Univ. Karlsruhe (ProceedingsBuilder)
Jutta Mülle, Univ. Karlsruhe (ProceedingsBuilder)

Programmkomitees:

wissenschaftliche Beiträge:
Hans-Jürgen Appelrath, Univ. Oldenburg
Wolf-Tilo Balke, Univ. Hannover
Michael Böhlen, Univ. Bozen/Bolzano
Klemens Böhm, Univ. Karlsruhe
Christian Böhm, LMU München
Stefan Conrad, Univ. Düsseldorf
Peter Dadam, Univ. Ulm
Jens Dittrich, ETH Zürich
Klaus R. Dittrich, Univ. Zürich
Burkhard Freitag, Univ. Passau
Christoph Freytag, HU Berlin
Norbert Fuhr, Univ. Duisburg-Essen
Götz Graefke, HP
Theo Häder, Univ. Kaiserslautern
Sven Helmer, Birbeck, London
Andreas Henrich, Univ. Bamberg
Olaf Herden, BA Stuttgart/Horb
Andreas Heuer, Univ. Rostock
Matthias Jarke, RWTH Aachen
Carl Christian Kanne, Univ. Mannheim

Alfons Kemper, TU München, Vorsitz
Ralf Klamma, RWTH Aachen
Christoph Koch, Univ. des Saarlands
Birgitta König-Ries, Univ. Jena
Wolfgang Lehrer, TU Dresden
Volker Linnemann, Univ. Lübeck
Bertram Ludäscher, UC Davis, USA
Stefan Manegold, CWI, Amsterdam
Wolfgang May, Univ. Göttingen
Klaus Meyer-Wegener, Univ. Erlangen-Nürnberg
Felix Naumann, HU Berlin
Andreas Oberweis, Univ. Karlsruhe
Reinhard Pichler, TU Wien
Erhard Rahm, Univ. Leipzig
Manfred Reichert, Univ. Twente
Norbert Ritter, Univ. Hamburg
Gunter Saake, Univ. Magdeburg
Kai-Uwe Sattler, TU Ilmenau
Eike Schallehn, Univ. Magdeburg
<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ralf Schenkel</td>
<td>MPI Saarbrücken</td>
</tr>
<tr>
<td>R. Ingo Schmitt</td>
<td>Univ. Magdeburg</td>
</tr>
<tr>
<td>Marc Scholl</td>
<td>Univ. Konstanz</td>
</tr>
<tr>
<td>Holger Schwarz</td>
<td>Univ. Stuttgart</td>
</tr>
<tr>
<td>Nicole Schweikardt</td>
<td>HU Berlin</td>
</tr>
<tr>
<td>Bernhard Seeger</td>
<td>Univ. Marburg</td>
</tr>
<tr>
<td>Thomas Seidl</td>
<td>RWTH Aachen</td>
</tr>
<tr>
<td>Günther Specht</td>
<td>Univ. Innsbruck</td>
</tr>
<tr>
<td>Uta Störl</td>
<td>Hochschule Darmstadt</td>
</tr>
<tr>
<td>Ralf Schenkel</td>
<td>MPI Saarbrücken</td>
</tr>
<tr>
<td>R. Ingo Schmitt</td>
<td>Univ. Magdeburg</td>
</tr>
<tr>
<td>Marc Scholl</td>
<td>Univ. Konstanz</td>
</tr>
<tr>
<td>Holger Schwarz</td>
<td>Univ. Stuttgart</td>
</tr>
<tr>
<td>Nicole Schweikardt</td>
<td>HU Berlin</td>
</tr>
<tr>
<td>Bernhard Seeger</td>
<td>Univ. Marburg</td>
</tr>
<tr>
<td>Thomas Seidl</td>
<td>RWTH Aachen</td>
</tr>
<tr>
<td>Günther Specht</td>
<td>Univ. Innsbruck</td>
</tr>
</tbody>
</table>

**Industrieprogramm:**
- Christian König, Microsoft
- Achim Kraiss, SAP
- Volker Markl, IBM
- Manfred Päßler, IBM
- Thomas Ruf, GFK
- Harald Schöning, Software AG, Vorsitz
- Joachim Thomas, UBS

**Demonstrationen:**
- Manfred Jeusfeld, Univ. Tilburg
- Wolfgang Klas, Univ. Wien
- Frank Leymann, Univ. Stuttgart
- Rainer Manthey, Univ. Bonn
- Andreas Oberweis, Univ. Karlsruhe
- Marco Pötke, sd&m
- Thomas Rose, Fraunhofer FIT, Vorsitz

**Gutachter für die Dissertationspreise:**
- Christoph Freytag, HU Berlin
- Andreas Heuer, Univ. Rostock
- Alfons Kemper, TU München
- Wolfgang Lehner, TU Dresden
- Gunter Saake, Univ. Madgeburg
- Marc Scholl, Univ. Konstanz
- Bernhard Seeger, Univ. Marburg
- Thomas Seidl, RWTH Aachen

**Externe Gutacher:**
- Matthias Beck
- Karsten Borgwardt
- Stefan Brecheisen
- Cristian Duda
- Michael Felderer
- Ingo Frommholz
- Tobias Geis
- Marco Grawunder
- Richard Hackelbusch
- Christoph Heinz
- Martin Husemann
- Hanna Köpcke
- Iryna Kozlova
- Kathleen Krebs
- Peer Kröger
- Martin Kuhlemann
- Peter Kunath
- Stefan Kurz
- Jens Lechtenbörger
- Carolin Letz
- Dominic Müller
- Cristian Pérez de Laborda
- Alexey Pryakhin
- Mario Pukall
- Matthias Renz
- Sebastian Richly
- Michael von Riegen
- Anke Schneidewind
- Matthias Schubert
- Uta Störl
- Christoph Sturm
- Maik Thiele
- Patrick Ziegler
Inhaltsverzeichnis

Eingeladene Beiträge

Christian Jensen (Aalborg University): *When the Internet Hits the Road* 2
Rudolf Munz (SAP AG): *Datenmanagement für SAP Anwendungen* 17
Götz Graefe (HP Labs): *Hierarchical locking in B-tree indexes* 18

Wissenschaftliches Programm

*Anfrageverarbeitung & Datenexploration*

Klaus Berberich, Srikanta Bedathur, Gerhard Weikum (Max-Planck-Institut für Informatik Saarbrücken): *Efficient Time-Travel on Versioned Text Collections* 44
Wolf-Tilo Balke, Wolf Siberski (Universität Hannover), Ulrich Güntzer (Universität Tübingen): *Getting Prime Cuts from Skylines over Partially Ordered Domains* 64
Matthias Beck, Sven Radde, Burkhard Freitag (Universität Passau): *Ranking von Produktempfehlungen mit präferenz-annotiertem SQL* 82
Svetlana Mansmann, Florian Mansmann, Marc H. Scholl, Daniel A. Keim (Universität Konstanz): *Hierarchy-driven Visual Exploration of Multidimensional Data Cubes* 96

*Indexierung*

Götz Graefe (HP Labs): *Algorithms for merged indexes* 112
Christian Böhm, Peter Kunath, Alexey Pryakhin, Matthias Schubert (Universität München): *Effective and Efficient Indexing for Large Video Databases* 132
Martin Lühring, Kai-Uwe Sattler (Technische Universität Ilmenau), Eike Schalllehn (Universität Magdeburg), Karsten Schmidt (Technische Universität Kaiserslautern): *Autonomes Index Tuning - DBMS-integrierte Verwaltung von Soft Indexen* 152
Modellierung

Boris Stumm, Stefan Dessloch (Universität Kaiserslautern): Change Management in Large Information Infrastructures - Representing and Analyzing Arbitrary Metadata

Susanne Busse (Technische Universität Berlin), Johann-Christoph Freytag (Humboldt-Universität zu Berlin): Entwurf von Informationsintegrationssystemen auf der Basis der Merkmalsmодellierung

Jürgen Göres, Stefan Dessloch (Universität Kaiserslautern): Towards an Integrated Model for Data, Metadata, and Operations

Boris Glavic, Klaus Dittrich (Universität Zürich): Data Provenance: A Categorization of Existing Approaches

Web & XML

Christian Mathis (Universität Kaiserslautern): Integrating Structural Joins into a Tuple-Based XPath Algebra

Holger Steinhaus (Universität Magdeburg), Klemens Böhm, Stephan Schosser (Universität Karlsruhe): Anreizmechanismen für Peer-to-Peer Web Crawling unter Berücksichtigung bösartiger Teilnehmer

Ralf Schenkel, Fabian Suchanek, Gjergji Kasneci (Max-Planck-Institut für Informatik Saarbrücken): YAWN: A Semantically Annotated Wikipedia XML Corpus

Jens-Peter Dittrich, Lukas Blunschi, Markus Färber, Olivier René Girard, Shant Kirakos Karakashian, Marcos Antonio Vaz Salles (ETH Zürich): From Personal Desktops to Personal Dataspaces: A Report on Building the iMeMex Personal Datspace Management System


Anfrageoptimierung

Thomas Neumann, Sebastian Michel (Max-Planck-Institut für Informatik Saarbrücken): Algebraic Query Optimization for Distributed Top-k Queries

Elke Achtelt, Christian Böhm, Peer Kröger, Peter Kunath, Alexey Pryakhin, Matthias Renz (Universität München): Efficient Reverse k-Nearest Neighbor Estimation
Stefanie Rinderle, Martin Jurisch (Universität Ulm), Manfred Reichert (University of Twente): *On Deriving Net Change Information From Change Logs - The DELTALAYER-Algorithm* 364

Soufyane El Allali, Daniel Blank, Martin Eisenhardt, Andreas Henrich, Wolfgang Müller (Universität Bamberg): *Untersuchung des Einflusses verschiedener Bild-Features und Distanzmaße im inhaltsbasierten P2P Information Retrieval* 382

*Evolution & Integration*

Stefan Dorendorf (Universität Jena): *Kosten und Nutzen von Datenbankreorganisationen: Grundlagen, Modelle, Leistungsuntersuchungen* 397

Fabian Groffen, Martin Kersten, Stefan Manegold (CWI Amsterdam): *Armada: a Reference Model for an Evolving Database System* 417

Andreas Thor, Toralf Kirsten, Erhard Rahm (Universität Leipzig): *Instance-based matching of hierarchical ontologies* 436

Frank Legler (IBM), Felix Naumann (Hasso-Plattner-Institut Potsdam): *A Classification of Schema Mappings and Analysis of Mapping Tools* 449

*Dissertationspreise*

Jens Teubner (Technische Universität München): *Pathfinder: XQuery Compilation Techniques for Relational Database Targets* 465

Martin TheoBALD, Ralf Schenkel, Gerhard Weikum (Max-Planck-Institut für Informatik Saarbrücken): *TopX - Efficient and Versatile Top-k Query Processing for Text, Semistructured, and Structured Data* 475

*Industrieprogramm*

*Verteilte Datenbanken*

Jürgen Angele (ontoprise GmbH), Michael Gesmann (Software AG): *The Information Integrator: using Semantic Technology to provide a single view to distributed data* 486
Thomas Legler, Wolfgang Lehner (Technische Universität Dresden), Andrew Ross (SAP AG): Der Einfluss der Datenverteilung auf die Performanz eines Data Warehouse 502

Dean Jacobs, Stefan Aulbach (Technische Universität München): Ruminations on Multi-Tenant Databases 514

Mobile und Grid-Anwendungen

Olaf Buck (PietschCONSULT GmbH), Robert Gajcy (Resco GmbH), Volker Linnemann (Universität Lübeck): Mobiles Computer Aided Facility Management 522

Andreas Walter (Forschungszentrum Informatik Karlsruhe), Klemens Böhm, Stephan Schosser (Universität Karlsruhe): Überlegungen zur Entwicklung komplexer Grid-Anwendungen mit Globus Toolkit 532

Datentransformation & Anfrageoptimierung

Stefan Krompaß, Stefan Aulbach, Alfons Kemper (Technische Universität München): Data Staging for OLAP- and OLTP-Applications on RFID Data 542

Matthias Böhm, Jürgen Bittner (SQL GmbH), Uwe Wloka (Hochschule für Technik und Wirtschaft Dresden), Dirk Habich, Wolfgang Lehner (Technische Universität Dresden): Ein Nachrichtentransformationsmodell für komplexe Transformationsprozesse in datenzentrischen Anwendungsszenarien 562

Alexander Behm (Berufsakademie Stuttgart / IBM Germany), Volker Markl, Peter Haas (IBM Almaden Research Center), Keshava Murthy (IBM Menlo Park): Integrating Query-Feedback Based Statistics into Informix Dynamic Server 582

Demonstrationsprogramm


Till Haselmann, Jens Lechtenbörger, Gottfried Vossen (Universität Münster): DataWarehouse Detective: Schema Design Made Easy 606

Alexander Hilliger von Thile (DaimlerChrysler Research and Technology): Managing the Desktop DocumentDataspace 609

XII
Martin Wimmer, Martina-Cezara Albutiu, Alfons Kemper, (Technische Universität München), Maarten Rits, Volkmar Lotz (SAP Research): Efficient Access Control for Composite Applications

Jan Rittinger, Jens Teubner, Torsten Grust (Technische Universität München): Pathfinder: A Relational Query Optimizer Explores XQuery Terrain

Katja Hose, Christian Lemke, Jana Quasebarth, Kai-Uwe Sattler (Technische Universität Ilmenau): SmurfPDMS: A Platform for Query Processing in Large-Scale PDMS

Armin Roth (Humboldt-Universität zu Berlin), Felix Naumann (Hasso-Plattner-Institut Potsdam): System P: Completeness-driven Query Answering in Peer Data Management Systems

Christian Grün, Alexander Holupirek, Marc H. Scholl (Universität Konstanz): Visually Exploring and Querying XML with BaseX
Eingeladene Beiträge
When the Internet Hits the Road

Christian S. Jensen

Department of Computer Science, Aalborg University, Denmark
http://www.cs.aau.dk/~csj

Abstract: The Internet has recovered from the dot-com crash of the early 2000’s and now features an abundance of new, innovative technologies and services. We are also witnessing the emergence of a communication and computing infrastructure that encompasses millions of people with mobile devices, such as mobile phones, with Internet connectivity. This infrastructure will soon enable the Internet to go mobile.

This paper describes the background and aspirations of a new research project that is concerned with data management aspects of innovative mobile Internet services. It is argued that mobile services will be context aware, and the project devotes particular attention to geographical context awareness. The project will adopt a prototyping approach where services are built and exposed to users, and where data management challenges are identified and addressed. The paper describes the evolving service platform that supports the approach chosen, it describes some of the data management techniques being integrated into the service platform, and it describes research guidelines that the project aims to follow.

1 Introduction

The Internet is teeming with new and innovative technologies and services. Many of these are fueled by Google-like business models where services are made available on a global scale and are free to their users, being paid for instead by other means such as advertisement. Many services build on community concepts. Blogs and RSS feeds are everywhere. Services that cater to our needs for factual information include Wikipedia. For images, there are services such as flickr.com, photo.net, and plazes.com. For music, there is last.fm, pandora.com, and The Hype Machine. For video, youtube.com is probably the most visible.

To give an impression of the richness of the services currently available, here are some additional video-related services: Angry Alien, AnimeEpisodes.Net, Badjojo, Blastro, Blennus, Blip.tv, Bofunk, Bolt, Break.com, Castpost, CollegeHumor, Current TV, Dachix, Dailymotion, Danerd, DailySixer.com, DevilDucky, Double Agent, eVideoShare, EVT1, FindVideos, Free Video Blog, Google Video, Grinvi, Hiphopdeal, iFilm, Keiichi Anime Forever, Kontraband, Lulu TV, Metacafe, Midis.biz, Music.com, MusicVideoCodes.info, MySpace, MySpace Video Code, Newgrounds, NothingToxic, PcPlanets, Pixparty, PlsThx, Putfile, Revver, Sharkle, SmitHappens, StreetFire, That Video Site, Totally Crap, VideoCodes4U, VideoCodesWorld, VideoCodeZone, vidilife, Vimeo, vSocial, Yikers, and Zip-
pyVideos (visit these at your own peril; I have only visited a few of them).

Second Life at secondlife.com features a virtual world that is being built and owned by its residents, of which there are currently more than 2 million. While the world is virtual, it is also real: it costs real money to “live” in this virtual world; it has its own currency that can be exchanged for real money; and one can make and spend money in this virtual world. Of course, there has already been a lawsuit in the real world about a property deal in the virtual world.

The above is just a small snapshot of what is currently available. While many other services could be singled out, these services collectively represent a good view of how a range of technologies are being used for the creation of innovative and, seemingly, commercially viable services. In parallel with these developments on the conventional Internet, an infrastructure is emerging that will enable the mobile Internet.

In particular, driven in large part by swift and sustained advances in computing and communication technologies, an infrastructure is emerging that contains vast quantities of computing and sensing devices that are Internet-worked by means of wireless communication technologies.

Notably, we are witnessing continued improvements in the capabilities of consumer electronics such as mobile phones, personal digital assistants, laptop computers, cameras, mp3 players, watches, navigation systems, and driver assistance systems. Existing types of systems and devices combine and new types with new functionalities emerge. The performance and performance/price ratios associated with key technologies utilized by such systems and devices continue to improve, which promises an increased proliferation.

Mobile phones are of particular interest. Most new phones are Internet-enabled, i.e., they have built-in data communication capabilities. The bandwidth is currently limited to well below 100 kbit/s for GPRS and EDGE [C06] networks and below 400 kbit/s for 3G [N03] networks. With the introduction of HSDPA [N06], with up to 14.4 Mbit/s, and WiMax [W06], with speeds of up to 10 Mbit/s, this is slated to change.

The current pricing for mobile Internet access, often in excess of 1€/MB, remains quite high. Flat-rate subscriptions are possible with some mobile providers, but again at a relatively high price. This is expected to change in the future—flat-rate subscriptions may be priced near today’s wireline broadband subscriptions. However, substantially cheaper mobile Internet access may still be some years away.

Geo-positioning is also becoming increasingly available. For example, network assisted GPS promises to eliminate the excessive power consumption of GPS receivers, thus rendering GPS practical for outdoor, battery-powered devices. Support for navigation will thus increasingly find its way into mobile phones. Some such devices already exist, e.g., the HP iPAQ hw6915, and others have been announced, e.g., the Nokia E95 which is expected to be available during the first quarter of 2007.

The first satellite of the Galileo positioning system has already been launched, and Galileo is expected to be operational around the turn of the decade [W07]. Galileo will offer better positioning than does GPS with respect to several aspects, including the accuracy, penetration, and time to fix [B06]. For example, the best-case accuracy (without the use
of ground stations) of Galileo is 45 cm as opposed to 2 m for GPS. Next generation GPS will also offer better positioning, and Galileo and GPS are expected to be interoperable.

This emerging infrastructure has the potential for enabling entirely new, geo-enabled services that were either not relevant or of little use in fixed desktop computing settings.

The range of possible applications and services is virtually limitless. For example, it includes services related to traffic and transportation such as “fleet” management, including emergency vehicle dispatching and hazardous cargo and traffic offender tracking; road-pricing where payment is dependent on where, when, and how much a vehicle drives; and other “metered” services, such as insurance and parking. It includes services that warn drivers about accidents, slow-moving vehicles, and icy and slippery road conditions ahead. It also includes a wider range of safety-related services, such as services that track senile senior citizens, tourists traveling in potentially dangerous environments, and prisoners serving time at home. Next, it includes the oft-mentioned point-of-interest services that offer information concerning gas stations, hospitals, etc. It also includes the emerging and challenging area of games and “-tainment” (edu-, info-, enter-) services. One theme is to move games from occurring in a virtual world behind a small computer screen to instead occurring in reality. Virtual objects, e.g., treasures (or caches, cf. geocaching [G07]), monsters, and bullets, are given geographical coordinates, and the coordinates of the real participants are also known by the system. This arrangement then enables games that aim to find treasures, catch or escape monsters, and hit with (virtual!) bullets.

This paper covers aspects of the streamspin project [S07] that has as its objective to identify and provide solutions to data management challenges on the mobile Internet. In particular, the project aims to identify fundamental data management services that will be needed by many mobile services; the project aims then to subsequently provide solutions.

The project will adopt a prototyping approach where innovative mobile services are prototyped and made available to users. In doing so, it is an objective to understand and learn from the services and technologies found on the conventional Internet—we hope this will enable us to prototype novel and viable mobile services. To facilitate the approach, we will build a testbed platform that will allow us to rapidly develop novel services and to experiment with data management aspects of these. The platform will fuel an Internet portal through which services are made available to users.

Section 2 argues that the mobile Internet is different from the conventional Internet and describes some of the differences. In also points to consequences of these differences. Section 3 then covers the streamspin Internet portal and service platform, including the functionality of the portal and the architecture of the platform. Section 4 adopts a geographical focus and describes the geo-related content and context that are relevant for mobile services. Next, Section 5 discusses self-imposed guidelines for the development of new data management technologies in the research project. Finally, Section 6 summarizes the paper.
2 The Mobile Internet is Different!

Users of the conventional Internet by far and large sit in front of a desktop or laptop computer with a relatively large screen and a convenient qwerty keyboard. And they are in a controlled environment, typically either at work or at home.

The mobile Internet is markedly different. Here, the services are typically delivered to mobile devices with small screens and either without a qwerty keyboard or with a very inconvenient one. The user is out and about, so the use situation is very varied.

The user may be sitting in a cafe or may be in a meeting. Or the user is on the move: by foot, using a collective means of transport, on bicycle, or driving a car. The user may be traveling in a familiar environment or may be in unfamiliar surroundings and alone. In many use situations, the user’s main focus of attention is not the service, but rather something else, e.g., navigating safely in traffic. The user is often engaged in a particular, primary activity.

These characteristics of the use situations are likely to have several consequences. In particular, many mobile services will aim to assist the user with the user’s primary activity. It thus becomes important to be able to anticipate the user’s needs. The varied use situations offer important clues as to what the primary activities are. The use situations and likely activities, and thus user needs, may be captured in a user’s context.

Also, it is important to utilize as best as possible the more constrained interaction between the mobile device and its user, as caused by the small screen, inconvenient keyboard, and, at times, disruptive surroundings. For example, doing a conventional Google search on a mobile phone is possible, but far from convenient in all use situations. Put differently, it is important to deliver the right service at the right time, with minimal system interaction. Again, the notion of context is useful.

Although context awareness is particularly important for mobile services, there is no general agreement on what mobile context is—rather, the notion of context is open ended, and its substance should probably depend on its intended use. It may include relatively static elements such as the user’s demographic data (e.g., age, education, gender) and interests and preferences (e.g., I like sushi and Thai food, and I prefer outdoor products from Arc’teryx and Falke). It may include a collection of profiles that the user has defined and may activate and deactivate. For example, a user may define a “I am hungry” profile, a “I want to shop” profile, and a “Keep me informed of sports news” profile.

The user’s current location-related data is also a possible element of a user’s context. As discussed earlier, accurate geo-positioning is becoming increasingly feasible. The location is an example of an attractive type of context, namely the type of context that can be acquired automatically, without requiring any activity on the part of the user. A GPS device will report not only a location, but also a heading and a speed.

A related kind of geo-context is the user’s destination, and yet another is the route that takes the user from the current location to the destination. Routes are interesting for two reasons. First, a mobile user typically travels towards a destination (rather than moving around, aimlessly), and a user often follows the same route when going from one location
to another. For example, a user typically travels on the same route from home to work. Second, routes are significant as context for many services. For example, a service that knows the route of a user may alert the user about road conditions, e.g., congestion and accidents, on the route ahead, while not bothering the user with conditions that do not relate to the user’s route. Routes may also be used when a user requests the locations of “nearby” points of interest. Thus, a service may suggest gas stations to the user that are near to the user’s route, rather than merely to the user’s current location.

Another consequence of the characteristics of the mobile Internet is that many mobile services will be so-called push services, i.e., services that a user registers for and that then automatically are provided to the user when certain conditions are met (e.g., [FZ98]). An example illustrates this. Assume that a user has registered for a traffic-condition service. The service knows the user’s current location, destination, and route to the destination. These elements are part of the user’s context. The location is provided by a tracking system that utilizes a GPS device. The destination and route are provided by a navigation system or predicted by a route-monitoring system that has accumulated a record of the user’s past travel behavior. When travel conditions are normal, the service is inactive—only when an accident or some other event has led to an anomalous condition that affects the user’s current travel does the service become active and supplies the user with, e.g., re-routing information or an updated arrival time prediction.

3 The streamspin Portal and Testbed Platform

We describe first the functionality of the portal, then describe the testbed platform that enables the functionality made available via the portal. Both the portal and platform exist in early prototype versions, and they will evolve throughout the course of the project.

3.1 Portal Functionality

The streamspin Internet portal is an open, context-aware, and regulated portal for mobile content and services. The portal is open in the sense that any registered user may supply and use content and services. Services may be developed using an application programming interface that is made available to developers, but services may also be created by completing service creation templates. For example, parents that need to transport their kids in connection with soccer practice and matches may complete a web form at home that will then generate a coordination service. The templates also allow easy setup of widely usable streams based on personal content. Examples include email, RSS and blog push. Finally, the portal also allows service providers to plug in their own templates that then allow users to create, and subsequently subscribe to, services that are delivered by the service providers. For example, it might be natural for a company to supply a customized email service to its employees by using this functionality.
The portal is context aware in the sense that users can enter demographic data and can create and activate profiles, as described briefly in the previous section. In addition, the users can supply dynamic context data, such as the current location, to the portal as well as to services subscribed to via the portal.

The portal is regulated in the sense that only legal content and services are allowed. In addition, the services registered at the portal must respect the contexts of the users. Thus, services should not push content to a user who has activated the “Do not disturb me” profile.

The portal acts as an intermediary between service and content providers, on the one side, and the users, one the other. The former then submit and register content and services, while the latter supply their context and subscribe to services. A main task of the platform underlying the portal then is to find matches between contexts and services and content. In our example from earlier, the platform must find matches between the route ahead for a user and events that occur. A match exists when an event is likely to affect the driving conditions of the user.

The platform is intended to serve as a convenient testbed for the exploration of new concepts in relation to mobile service delivery. Thus, the platform is designed to be open and extendable.

From a technical perspective, the platform supports two quite different kinds of services, so-called portal services and third-party services.

Portal services are the simplest kind. The platform receives streams of content from content providers. These may include RSS-feeds with traffic information, weather information, news, etc. Content broadcast via TMC [T07] also fits into this category. For each user who has registered for content, the platform matches all incoming content against the user’s context and then passes matches on to the user. To control the matching, the content providers are offered the opportunity to tag all content with metadata obtained from a hierarchy provided by the platform. With portal services, no third parties have access to the context of a user.

The third-party services differ from the portal services in that they by default pass contents directly to the users. This is natural for communication services, e.g., e-mail, and for high-quality services, e.g., different location-based services. The providers of third-party services also have access to the users’ contexts. Thus, a third-party service can provide a “guided” city tour for tourists or provide points of interest that are near to the service users.

3.2 Testbed Platform Architecture

The overall four-layer system architecture underlying the streamspin portal is shown in Fig. 1. Each box is a software component and an upper layer uses the layers below to provide a given functionality.

The information layer at the bottom has four main components. The context encompasses
location and time (which may be aggregated into routes). The layer also contains software components for demographic data and profiles. As discussed earlier, a user may have a number of profiles, one or more of which are active. Finally, content is provided by public or private content and service providers.

Next, the matching layer accomplishes the matching of the content against the user’s context, including the user’s demographic data and active profiles. The platform allows the users to specify additional user-defined filters. This is to enable further personalization of the content provided to a user.

The third layer is the delivery layer, which is responsible for the actual delivery of content to the mobile devices. In this layer, the buffering and expiration component is quite important. It ensures that content for a device that is offline is delivered next time the device becomes online if the content remains relevant. This component is also responsible for ensuring that old (expired) content is not delivered to a device.

The delivery layer offers multiple ways of delivering content to the devices. Overall, it applies both push or pull principles. Each user can specify a preferred means of obtaining content based on, e.g., the technical capabilities of the user’s terminal and the mobile services available.

The three layers discussed so far all reside on the server side. The fourth and final layer, the application layer, resides on the client side, i.e., on the mobile devices. This layer’s responsibility is to receive the content from the server side and display it on the device. To enable this functionality, a user only has to download a single program from the server and install this on the device. This program is capable of receiving and presenting the content delivered by all services. The addition and deletion of services on the server thus does not affect the software installed on the clients.
The platform currently supports devices that use the Microsoft Mobile 5.0 operating system (with the .Net Compact Framework 2.0) as well as Java enabled devices (with MIDP 2.0).

4 Data Management Foundations with a Geographical Focus

This section adopts a data centric perspective on mobile services. It discusses the various types of geographical content and context of relevance for mobile services, including business content (e.g., point-of-interest data), generic geo-content, also termed infrastructure, and user-specific geo-context.

We are working to integrate support into the *streamspin* platform for mobile services that involve these types of data. This includes support for positioning by other means than GPS, it includes tracking services, and it includes services that enable the prediction of a user’s geographical context such as the user’s destination and route. The section borrows from an earlier paper by the author [Je06]. Two recent books are recommended as further readings [GS05, VS04].

4.1 A Data Centric Perspective

The data centric perspective may be formulated as follows: By capturing pertinent aspects of reality in digital form—in semantically rich and appropriately organized structures, and with powerful update and retrieval techniques available—an ideal foundation for delivering a wide range of mobile services is obtained.

The idea is that a service request by a mobile user translates into queries against the database envisioned in the statement. A key challenge in the delivery of mobile services then becomes a data processing problem.

The phrase “digital mirror of reality” has been used for describing the envisioned database. While this phrase is intuitive and paints the right kind of mental picture, it only partially reflects the desirable capabilities of this database, which go well beyond simply being a mirror. In particular, the database may capture past states of reality and one or more perceived future states, in addition to the current state. In more technical terms, the database supports the valid-time aspect of data. Further, if accountability is a concern, the database may include an incorruptible record of its past states. In technical terms, this is called transaction-time support.

Next, the database and the database management system used may not be a single relational or object-relational database stored in a centralized system. Rather, the database and system may well be distributed and heterogeneous in a number of respects. For example, the data may be physically distributed and may not adhere to the same common schema or data model. The control and data processing may also be distributed.
4.2 Infrastructure and Business Content

The delivery of geo-enabled mobile services in practice is dependent on relevant content being available. Examples of content include weather data; traffic condition data, including information about accidents and congestion; information about sights and attractions, e.g., for tourists; information about hotel rooms, etc. available for booking; and information about the current locations of populations of service users.

The management of such content includes several aspects. An information technology infrastructure must exist that is capable of capturing the content and capable of absorbing the content as it is made available, while being able simultaneously to make the content available to services.

We may distinguish between two types of content: the geographical infrastructure itself and all the other, "real" content that may be given geographical references and that must reference the infrastructure. Points of interest exemplify real content.

The geographical infrastructure, or geo-content, concerns the geographical space itself, with hills, lakes, rivers, fjords, etc. It also concerns the road networks for use by vehicles and the transportation infrastructures for, e.g., pedestrians, trains, aircraft, and ships. The infrastructure for vehicles is of high interest because users may frequently be either constrained to, or at least using, this infrastructure.

Geo-content is important. Users think of the real content as being located in a transportation infrastructure, and they access the content via the infrastructure. For example, the location of a point of interest is typically given in terms of the road along which it is located, and directions for how to reach the location are given in terms of the transportation infrastructure.

For the delivery of a range of geo-enabled mobile services, it is particularly important that a representation of the road infrastructure is available that supports multiple functions, including content capture; content update and querying, including route planning and way finding; and user display. This representation may be composed of several constituent representations [Je04].

It is common practice to specify the location of some content relative to the nearest kilometer post along a specific road. For example, the entry to a parking lot may be indicated by a road, a kilometer post on that road, and an offset. So-called linear referencing may be used when capturing such content.

A weighted, directed-graph representation may also be used that represents a quite abstract view of an infrastructure. This representation ignores geographical detail, but preserves the topology, and it may be used for connectivity-type queries, such as route guidance and way finding.

Next, a geo-representation is also needed that captures the geographical coordinates of the road infrastructure. With this representation, it is possible to map a location given in terms of geographical coordinates, e.g., from GPS receivers or point-on-a-map-and-click interfaces, to a location in the infrastructure. Finally, these three representations must be connected.
All the “real” content encompasses any content that may reference, directly or indirectly, the geographical infrastructure. A museum, a store, or a movie theatre may have both a set of coordinates and a location in the road network. This type of content is open-ended and extremely voluminous. For example, it may include listings of movies currently running in the movie theatre; it may include seat availability information for the different shows; and it may include reviews of the movies. Often, the real content is the primary interest of the users.

Content is generally dynamic. This applies to road networks, where road construction and accidents change the characteristics of the networks with varying degrees of permanence. Other content is also dynamic. Examples abound. New stores open and existing stores relocate or close. The opening hours of a facility may change. The program of a movie theater changes. The sales available in a store change. This dynamicity of content implies that a representation of content must be designed to accommodate updates.

Content is more or less dynamic. The content that derives from the sampling of the positions of moving objects belongs at the highly dynamic end of the spectrum. Capturing the present positions, and possibly the past as well as anticipated future positions, of a large population of mobile users requires special techniques, as discussed next.

4.3 Geo-Context—Locations, Destinations, Routes, and Trajectories

User-specific geo-context is another kind of content. Among such content, the current position of a service user is the traditional geo-context used in location-based services.

To maintain an up-to-date record of the current position of a service user, we may envision a scenario where a central server maintains a representation of the user’s movement and where the local client, e.g., a mobile phone, is aware of the server-side representation. The client frequently compares its GPS position to the server-side position, and when the two differ by a threshold slightly smaller than the accuracy required, an update is issued to the server, which then revises its representation of the client’s movement and sends this new representation to the client [CJP05]. This arrangement, termed shared-prediction-based tracking, aims to reduce the number of updates needed in order to main a current position at a given accuracy.

Different representations of a user’s movement result in different rates of update. We consider several possible representations in turn. First, we may represent the movement of a user as a constant function, i.e., as a point. With this representation, an update is needed every time the user has moved a (Euclidean) distance equal to the threshold away from the previous position. This is a simple representation, and it may be useful when the user is barely moving or is moving erratically within an area that is small in comparison to the area given by the threshold used.

Second, we may represent the movement by a linear function, i.e., by a vector. When the user exceeds the threshold, the user sends the current GPS location and the current speed and direction (which GPS receivers also provide) to the server. The server then uses this information to predict the user’s to-be-current positions.
Third, we may utilize the infrastructure in the representation of a user’s movement. This requires that we are able to locate the user with respect to the infrastructure. One possibility is to assume that the user is moving at constant speed along the road on which the user is currently located. We may use the GPS speed as the constant speed, and we may assume that the user stops when reaching the end of the current road segment. Depending on the lengths of the segments, this representation can be expected to be better or worse than the vector representation. However, for realistic segments, this representation has the potential for outperforming the vector representation.

Next, we may use the route of the user in place of the segments. As argued earlier, most humans who travel do so towards a known destination; and being creatures of habit, and perhaps for efficiency, we tend to follow routes we have previously followed. Therefore, we will frequently be able to predict correctly the route on which a service user travels [BJ07]. Using the correct route in place of a road segment means that the number of updates needed to maintain a user’s position with the desired accuracy decreases further. Indeed, updates occur only because of incorrectly predicted speeds—no updates are caused by incorrectly predicted “locations.” It should also be observed that if a route is predicted incorrectly, e.g., because the user makes a turn, this does not lead to a breakdown. Rather, this simply forces an update and a new prediction.

The infrastructures currently available for mobile services support the accumulation of GPS data from vehicles. Based on this data, it is possible to gradually create usage patterns for vehicles that consist of the routes traveled by the vehicles along with usage metadata, which are temporal patterns that describe for each vehicle and route when the route is being used by that vehicle [BJ07]. For example, a pattern may specify that a route is being used in the morning on weekdays. The resulting route and destination data may subsequently be used in services. By also attaching travel speeds to routes, we obtain trajectories, which are routes “lifted” into the time dimension [GS05].

5 Research Challenges

We proceed to present six guidelines for conducting research that we are trying to follow in our ongoing research. These are intended to apply to the area of geo-enabled, mobile services, but are to varying degrees also applicable to other areas of research. (Reference [Je06] relates the guidelines to past research that the author has been involved in.)

For application-oriented research, estimate the time of application and formulate expectations to the reality as of that time; then design for that reality.

Different research activities may aim to be applicable within different time horizons. For research that is intended to be applicable in the near term, we must take care to only make assumptions that are met by current infrastructures. Assumptions concern the available computing and storage capabilities of mobile terminals, the available communication technologies, the available positioning technologies, the available digital road networks, and existing legislation.
Other research may only be applicable in the longer term and perhaps only indirectly. Such research may be more speculative and somewhat more removed from specific applications. Some of the results may not offer the final answers, but may serve as a foundation for further work.

Ensure that at least one appropriate architectural setting exists or may be envisioned for the research contribution.

For research that is expected to have practical application in the short term, the architectural setting is likely to be a concern. For other research, it may be sufficient to ensure that an appropriate architectural setting exists or can be envisioned. And for yet other research, architectural settings may not be an important concern.

Invent solutions for composable functionality.

When research on query processing in relation to moving objects first took off, the efficient processing of many basic types of queries had yet to be explored in the new moving-object settings. Examples included one-time and continuous range queries, nearest-neighbor queries, and reverse nearest-neighbor queries, to name but a few.

As techniques for the processing of these basic types of queries accumulate, it is natural that attention shifts to as yet unexplored or lightly explored types of queries. A potential pitfall is that we start producing highly optimized solutions to very specialized types of queries. This path is not advisable, as the prospects of these solutions finding practical applications are likely to decrease with the degree of specialization of the functionality.

Prioritize versatile and robust solutions over specialized and brittle, although possibly highly performant, solutions.

One lesson to be learned from current, commercial data management technology and existing applications is that versatile and robust solutions have better chances of finding practical use than do very specialized ones, even if these exhibit very high performance in some cases. The objective of a query optimizer is quite modest: it should avoid the clearly inefficient ways of computing a query and identify one good way of executing the query. (Even meeting this objective can be a challenge.)

In the area of data processing for moving objects, the parameter space—the numbers of parameters and parameter settings that characterize a data processing workload—is very large. One consequence is that there is room for many solutions that offer superior performance for certain settings, but may be clearly inefficient for other settings.

The recommendation is that we try to aim for solutions that are versatile in terms of the functionality they offer and that are robust in terms of the settings. A solution—for which there exist other solutions so that for every possible parameter setting, at least one of these other solutions has twice the performance—may still be preferable if it is much more robust than its competitors.
Design query processing techniques that exploit the entire context.

When we design query processing techniques, we should try to use as arguments all the context that we can reasonably expect to have available. So if we can assume to know the likely route of a moving vehicle, we may suggest restaurants or gas stations to the driver that are near to the expected route, rather than merely to the driver’s current location, which is the best a service can do if it ignores the route. Moreover, utilizing knowledge of the road network, we can use network distances as opposed to Euclidean distances in our calculations, and we can augment the answers with travels and detour distances and with suggested routes to the points of interest returned.

Pay attention to both query performance and update performance.

Many indexing and query processing techniques for geographical data were originally developed for largely static data. For example, R-trees do not contend well with workloads with frequent updates. In contrast, mobile-service application scenarios exist that are characterized by frequent position updates. This puts focus on techniques that are capable of supporting workloads consisting of frequent updates as well as queries, and it puts focus on studies of the trade-off between query performance and update performance [Je02, JS02].

Updates of moving-object positions correspond to the sampling of continuous, position-valued variables. Different services may tolerate different position inaccuracies. For example, a localized-weather service may tolerate a relatively high degree of inaccuracy without this affecting the functionality of the service, while a navigation service is dependent on more accurate positions.

One approach to taking advantage of the different accuracy tolerances of different services is to perform updates only when needed to maintain the accuracies needed. Indexing and query processing techniques should be able to exploit this approach to updates. Also, by forming predictions of the future movements of the objects, the numbers of updates can be further reduced.

6 Summary

A new and exciting Internet is emerging—the mobile Internet. While it is difficult to predict the character of this Internet, this paper argues that the mobile Internet will entail services that are intended to aid their users with the varying, primary tasks that they aim to accomplish while being mobile. It argues that services should be able to anticipate the primary tasks of the users and should reduce the interaction with the mobile devices needed in order to benefit from the services. The paper also suggests that mobile services to a larger extent than conventional Internet services will be so-called push services. Context awareness is suggested as a key characteristic of mobile services.

The paper describes a mobile service platform and associated portal that are being developed in a new project that aims to explore data management foundations for mobile
services. The idea is to learn from and apply some of the many concepts underlying novel services already available on the conventional Internet for the creation of novel mobile services. Such services are then prototyped and exposed to users via the portal. Based on lessons learned from this activity, novel data management services are being added to the platform and portal that the mobile services may exploit to realize their functionality while ensuring that the platform is scalable. The project will focus on geographical data management services, e.g., services that capture the geographical aspect of data and the users’ geo-contexts. In an attempt to invent technology that is as relevant for practice as possible, six self-imposed guidelines have been identified.

The platform and portal exist in prototype versions and are intended to evolve throughout the duration of the project that is scheduled to last for 4 years. We are currently integrating tracking and route-recording services into the platform (as described in Section 4.3), and we are also adding new service creation templates to the system.

Acknowledgments

I would like to thank the other three current key participants of the streamspin project: Kenneth H. Pedersen, Kristian Torp, and Rico Wind. I would also like to acknowledge those other local colleagues with whom I currently conduct research on the topics covered in this paper: Agne Brilingaitė, Alminas Ėivilis, Xuegang Huang, Harry Lahrmann, Hua Lu, Stardas Pakalnis, Torben Bach Pedersen, Simonas Šaltenis, Albrecht Schmidt, Laurynas Speičys, Dalia Tiešytė, Nerius Tradišauskas, and Man Lung Yiu.

My interactions with these dedicated and capable individuals have shaped the thoughts and views expressed in this paper, although only I should be held accountable for any inadequacies of these.

References


*References containing URLs are valid as of January 1, 2007.*
Datenmanagement für SAP Anwendungen

Rudolf Munz
Product Support
SAP AG
Dietmar-Hopp-Allee 16
69190 Walldorf
rudolf.munz@sap.com


Das SAP Datenmanagement basiert aber nicht ausschließlich auf einer SQL DBMS Abstraktion. Um die Performanz und die Skalierbarkeit der SAP Anwendungen sicherzustellen, gibt es zusätzlich eine Reihe von spezialisierten Datencontainern: Table Buffer, Content Server, liveCache sowie den BI Accelerator. Die Motivation für ihre Entwicklung, ihre technischen Grundprinzipien, die praktischen Erfahrungen mit ihnen und mögliche Weiterentwicklungen werden im Vortrag erläutert.

Beim praktischen Einsatz von DBMS ist heute weniger ihre Funktionalität aus der Sicht der Anwendungsentwicklung als vielmehr ihr Betriebskonzept, d. h. ihre Administrationssicht entscheidend. Hier gibt es steigende Kundenerwartungen bezüglich der Administrationsfreiheit und kontinuierlichen Verfügbarkeit sowie neue DBMS-Einsatzszenarien durch künftige Software-Lieferkonzepte wie z. B. „Software-as-a-Service“.
Hierarchical locking in B-tree indexes

Goetz Graefe
HP Labs
Goetz.Graefe@HP.com

Three designs of hierarchical locking suitable for B-tree indexes are explored in detail and their advantages and disadvantages compared.

Traditional hierarchies include index, leaf page, and key range or key value. Alternatively, locks on separator keys in interior B-tree pages can protect key ranges of different sizes. Finally, for keys consisting of multiple columns, key prefixes of different sizes permit a third form of hierarchical locking.

Each of these approaches requires appropriate implementation techniques. The techniques explored here include node splitting and merging, lock escalation and lock de-escalation, and online changes in the granularity of locking. Those techniques are the first designs permitting introduction and removal of levels in a lock hierarchy on demand and without disrupting transaction or query processing.

In addition, a simplification of traditional key range locking is introduced that applies principled hierarchical locking to keys in B-tree leaves. This new method of key range locking avoids counterintuitive lock modes used in today’s high-performance database systems. Nonetheless, it increases concurrency among operations on individual keys and records beyond that enabled by traditional lock modes.

1 Introduction

Since the 1970s, hierarchical locking has served its intended purpose: letting large transactions take large (and thus few) locks and letting many small transactions proceed concurrently by taking small locks. Multi-granularity locking was one of the great database inventions of that decade.

Hierarchical locking is widely used for database indexes. The standard lock hierarchy for B-tree indexes starts by locking the table or view, then locks the index or an index partition, and finally locks a page or individual key. This design is implemented in many database systems where it works reliably and efficiently.

1.1 Problem overview

With disk sizes approaching 1 TB and with very large databases and indexes, this hierarchy of locking begins to show a flaw. Specifically, the currently common step from locking an index to locking its individual pages or keys might prove too large. There may be millions of pages and billions of keys in an index. Thus, if a lock on an entire index is too large and too restrictive for other transactions, thousands and maybe millions of individual locks are required. If a transaction touches multiple indexes and if there are many concurrent transactions, the count of locks multiplies accordingly.

---

1 Palo Alto, CA. Much of this research was performed while employed by Microsoft.
Each lock structure takes 64-128 bytes in most implementations. This includes a
doubly linked list for the transaction and one for the lock manager’s hash table, where
each pointer nowadays requires 8 bytes. In addition, there is a list for transactions waiting
for a lock conversion (e.g., shared to exclusive) and a list of transactions waiting to ac-
quire an initial lock on the resource. Thus, millions of locks take a substantial amount of
memory from the buffer pool, from sort operations for index creation and index mainte-
nance, and from query processing.

Acquiring and releasing a single lock costs thousands of CPU cycles, not only for
memory allocation but also for searching the lock manager’s hash table for conflicting
locks, and not only for instructions but also for cache faults and pipeline stalls in the
CPU. Thus, millions of locks require billions of CPU cycles, which is why hierarchical
locking and automatic lock escalation are very important for commercial workloads.

A single ill-advised lock escalation suffices, however, to reduce the multi-
programming level to one. Consider, for example, a transaction that reads some fraction
of a B-tree index and updates some selected entries. Depending on the fraction read, tens
or hundreds of such transactions operate concurrently on a single B-tree index. If a single
one of them exceeds its lock escalation threshold and indeed escalates to an index lock,
no other transaction can update even a single B-tree entry, the multi-programming level
collapses and server throughput drops accordingly.

1.2 Solution overview

Neither abandoning hierarchical locking nor partitioning large indexes into small
partitions solves this problem satisfactorily. The solution advanced in this research rede-
fines the hierarchy of resources locked. It introduces two new designs with additional
lock granularities between an entire index and an individual page or key, and compares
strengths and weaknesses of traditional locking hierarchies with the two new designs.

All three designs let large scans lock entire indexes such that no conflict can invali-
date a scan when it is almost complete. They also let transaction processing lock individ-
ual records or keys. In addition, the two new designs let queries lock index fractions of
various sizes. Thus, the original spirit of hierarchical locking is preserved and indeed
strengthened by eliminating the dilemma between locking an entire index and acquiring
thousands or millions of individual locks.

In contrast to prior lock hierarchies, the two new designs support online changes to
the granularity of resources available for locking. Thus, they permit to start with trad i-
tional locks on indexes and keys only and to introduce or remove larger locking granules
if, when, where, while, and as much as warranted by the workload.

In addition, a new design for key range locking is introduced. Compared to existing
methods, it relies on hierarchical locking in a principled way. It is simpler than traditional
designs for key range locking as it avoids counter-intuitive lock modes such as “Range-
N” (discussed below). Nonetheless, it permits higher concurrency in B-tree leaves. It re-
quires new lock modes, but eliminates as many, and the new lock modes are based on
established sound theory. Due to these advantages, the new design for key range locking
has value independent of the proposed techniques for locking large sections of B-trees.

The following sections first review related work and some preliminaries, then de-
scribe each of the three alternative designs for hierarchical locking in B-tree indexes and,
and finally offer some conclusions from this effort.
2 Related work

Multiple prior efforts have investigated and refined multi-granularity locking as well as key range locking in B-tree indexes. The techniques proposed in subsequent sections do not require understanding of the related work reviewed in this section, and some readers may wish to skip ahead. The details given here are provided for those readers who wish to compare and contrast prior techniques with the present proposals.

2.1 Related work on multi-granularity locking

Multi-granularity locking has been a standard implementation technique for commercial database systems for more than a quarter century [GLP 75]. Other hierarchical concurrency control methods [C 83] have not been widely adopted.

Multi-granularity locking is often restricted to hierarchies. The hierarchy levels usually include table, index, partition, page, record, key, or a subset of these. The choice of lock level can be based on explicit directives by the database administrator or by the query developer or it can be based on cardinality estimates derived during query optimization. The theory of multi-granularity locking is not restricted to these traditional hierarchies, however. A premise of this research is that alternative hierarchies deserve more attention than they have received in the past.

A radically different approach has also been advocated in the past. In predicate locking, locks contain query predicates, and conflict detection relies on intersection of predicates [EGL 76]. In precision locking [JBB 81], read locks contain predicates whereas write locks contain specific values. Conflict detection merely applies predicates to specific values. Precision locking may be seen as predicate locking without the cost and complexity of predicate intersection.

These techniques target the same problem as hierarchical locking, i.e., locking appropriate large data volumes with little overhead, but seem to have never been implemented and evaluated in commercial systems. The key-prefix locks in Tandem’s NonStop SQL product [GR 93] might be closest to an implementation of precision locks. Note that Gray and Reuter explain key-range locking as locking a key prefix, not necessarily entire keys, and that “granular locks are actually predicate locks” [GR 93].

“In addition to the ability to lock a row or a table’s partition, NonStop SQL/MP supports the notion of generic locks for key-sequenced tables. Generic locks typically affect multiple rows within a certain key range. The number of affected rows might be less than, equal to, or more than a single page. When creating a table, a database designer can specify a "lock length" parameter to be applied to the primary key. This parameter determines the table’s finest level of lock granularity. Imagine an insurance policy table with a 10-character ID column as its primary key. If a value of “3” was set for the lock length parameter, the system would lock all rows whose first three bytes of the ID column matched the user-defined search argument in the query.” [from SB 97; see also BS 95]

Gray [G 06] recalls a design in which locking a parent or grandparent node in S mode protects the entire range of all leaf nodes underneath. A lock on the root page locks the entire index. IS locks are required on the entire path from the root to a leaf. A leaf-to-leaf scan locks appropriate parent and ancestor nodes. Exclusive locks work similarly.

At first sight, these locks may seem equivalent to key range locks on separator keys. Both methods adapt gracefully to very large tables and indexes. However, there are im-
important differences. For example, before a leaf page can be locked in S mode, its parent page needs to be locked in IS mode. Both locks are retained until transaction commit. The IS lock prohibits any X lock on that parent node and thus prevents any update there, e.g., splitting a different leaf page under the same parent or splitting the parent itself. While holding IS locks on the root page, user transactions prevent any kind of update of the root node, e.g., splitting a child node or load balancing among the root’s child nodes.

In addition, System R had “a graph for physical page locks” [GPL 75] but no further public documentation seems to exist [G 06].

2.2 Related work on key range locking

In the final System R design, a lock on a key implicitly locks the gap to the next key value; locking a key and locking the gap to the next key is the same [BG 06]. The conceptual hierarchy applies to fields within a record but does not consider the gap between keys a lockable resource separate from a key.

The Aries family of techniques includes two methods for concurrency control for B-tree indexes, key value locking [M 90] and index management [ML 92]. The former is quite subtle as it relies not only on locks but also on latches and on instant locks. The latter establishes that a lock on a row in a table can cover the index entries for that row in all the indexes for the table.

Lomet’s design for key range locking [L 93] builds on Aries [M 90, ML 92] but leaves multiple opportunities for further simplification and improvement. The present work addresses those pointed out here.

- Most importantly, multi-granularity locking is considered for general ranges and keys, but is described in detail only for keys in leaf pages.
- Lock manager invocations are minimized by identifying multiple granules with the same key value and by introducing additional lock modes, but the B-tree level is omitted in the lock identifier which implies that locking key ranges in interior B-tree nodes is not possible.
- Intention locks (e.g., IS, IX) on the half-open interval \([K_{i-1}, K_i]\) permit absolute locks on the key value \(K_i\), but no provision is made to lock the open interval \([K_{i-1}, K_i)\) separately from the key value, and appropriate concurrency can be achieved only by using unconventional and counter-intuitive lock modes.
- Protecting a gap between two keys by locking the prior or next key is discussed and next-key locking is chosen, but it seems that prior-key locking is more suitable for B-trees with suffix truncation for separator keys [BU 77], i.e., half-open intervals of the form \([K_i, K_{i+1})\).
- Application of key range locking to heap files is mentioned, but not generalized to ranges larger than immediately successive record identifiers.
- Locking complex objects with sets of components by means of range locks is considered, but neither developed in detail nor related to B-tree indexes.
- Locking individual pairs of key and record identifier even if the representation stores each distinct key value only once is discussed, but locking such a key value with all its record identifiers is neither appreciated for its beneficial interaction with query predicates nor generalized to locking arbitrary key prefixes in a hierarchy.
Finally, deletion using “ghost” records is mentioned, i.e., separating row deletion in a user transaction and key removal in a subsequent clean-up action, but instead of applying this idea symmetrically to insertions, “instant” insertion locks are employed.

Microsoft SQL Server is a sample commercial database system using key range locking [M 06]. Many aspects of the design and its implementation match Lomet’s description [L 93]. It locks the open interval between two keys together with the interval’s high boundary key. This is evident in the “RangeI-N” mode, which is an insertion lock on the open interval with no lock on the boundary.

Each key value identifies locks on both the key value and the gap to the neighboring key such that a single invocation of the lock manager may obtain a combined lock. These savings are enabled by introduction of additional lock modes. This technique is employed in multiple ways, including “schema stability” and “schema modification” locks on tables. Schema stability read-locks a table’s catalog information, e.g., during query optimization. In order to save lock manager invocations during query execution, both S and X locks on a table imply a schema stability lock. Schema modification write-locks both the catalogs and the data, e.g., during schema changes. Schema, data, and their combination could be modeled as a hierarchy in the locking protocol. However, while locking the schema without locking the data is useful, e.g., during query optimization, it is not immediately obvious what it means to lock the data without protecting the schema against modifications by other transactions.

In SQL Server’s implementation of key range locking, the set of combined lock modes is not equivalent to multi-granularity locking. This is evident in several missing lock combinations, e.g., an exclusive lock on the gap between keys with no lock on the key value, i.e., there is no “RangeX-N” mode. This mode would permit one transaction to delete a key within an interval while another transaction modifies (the non-key portions of) the record at the interval’s high boundary. Similarly, a query with an empty result cannot lock the absence of a key without locking at least one existing key value, i.e., there is no “RangeS-N” mode. Finally, due to the missing “RangeS-X” and “RangeS-N” modes, it is not possible that one query inspects a key range and then updates the boundary key while another query merely inspects the key range.

Despite these missing lock modes, SQL Server uses as many as 6 lock modes of the form “RangeP-Q”, where P can be S, X, or I, i.e., shared, exclusive, or insertion lock modes, and Q can be S, X, or N, i.e., shared, exclusive, or no lock. In addition, SQL Server employs update locks [GR 93, K 83] on key values but not on ranges. Nonetheless, update locks add another 3 lock modes of the form “RangeP-U”, where P again can be S, X, or I.

SQL Server relies on short-term locks to guard key ranges against insertion of phantom records. Ghost records and system transactions (see below) are used for deletion but not for insertion. SQL Server employs lock escalation from keys or pages to an entire

---

2 In a way, schema stability protects an entire complex object with rows in multiple tables and records in multiple indexes. We plan on generalizing this idea to user-defined complex objects in future work.

3 There are exceptions to this general statement. For example, one transaction’s query may continue while another transaction adds a new constraint to a table. We plan on investigating concurrency among schema operations and data operations in future work.
index, based on an initial threshold and an incremental threshold if earlier lock escalation attempts failed. SQL Server does not support lock de-escalation. Rollback to a savepoint does not reverse lock escalation or lock conversion.

IBM’s multi-dimensional clustering [BPM 03, PBM 03] is a special implementation of partitioning that includes the ability to lock an entire “block” (partition) or individual records within a block, albeit without lock escalation from record to block. The only commercial database system that supports lock de-escalation seems to be Oracle’s (formerly Digital’s) RDB product [J91], although only in the special case of “lock caching” within nodes of a shared-disk cluster.

3 Assumptions

Assumptions about the database environment are designed to be very traditional.

B-tree indexes map search keys to information. This information might represent a row in a table or a pointer to such a row, i.e., B-trees may be clustered or non-clustered indexes. Each B-tree entry fits on a page; for larger objects, B-tree variants with a byte count rather than a search key added to each child pointer [CDR 89, SL 88] may be an attractive storage format but are not explored further here.

B-tree pages range from 4 KB to 64 KB, although the discussion applies to pages of any size. The number of records per page may be in the tens, most likely is in the hundreds, and in extreme cases is in the thousands.

Prefix and suffix truncation [BU 77] might be applied to leaf entries and to separator keys, but hardly affect the discussion at hand. Other compression methods, e.g., order-preserving run-length encoding or dictionary compression [ALM 96], affect the representation of keys and records but do not change the substance of the discussion, neither the techniques nor the conclusions.

<table>
<thead>
<tr>
<th>Latches</th>
<th>Locks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separate …</td>
<td>Threads</td>
</tr>
<tr>
<td>Protect …</td>
<td>In-memory data structures</td>
</tr>
<tr>
<td>Modes</td>
<td>Shared, exclusive</td>
</tr>
<tr>
<td>Duration</td>
<td>Critical section</td>
</tr>
<tr>
<td>Deadlock …</td>
<td>Avoidance</td>
</tr>
<tr>
<td>… by …</td>
<td>Coding discipline, “lock leveling”</td>
</tr>
</tbody>
</table>

Table 1. Latches and locks.

The database implementation observes the separation of latches and locks summarized in Table 1. This separation lets a single thread serve multiple connections and transactions and it lets a single transaction use multiple threads in a parallel query or index operation. In-memory data structures include the in-memory images of database pages.
The database contents include only those characteristics observable with ordinary "select" queries. This excludes database pages, distribution of records within pages, existence and contents of non-leaf pages in B-trees, etc. Thus, non-leaf pages do not require locks and are protected by latches only. The remainder of this paper focuses on locks.

In contrast to user transactions, "system transactions" modify only the database representation but not its contents and therefore permit certain optimizations such as commit without forcing the transaction log. Logging and recovery rely on standard write-ahead logging [G 78, GR 93, MHL 92].

A common use of system transactions is to remove ghost records left behind after deletions by user transactions. Separation of logical deletion (turning a valid record into a ghost) and physical removal (reclaiming the record’s space) serves three purposes, namely simplified rollback for the user transaction if required, increased concurrency during the user transaction (locking a single key value rather than a range), and reduced overall log volume. If the ghost removal can capture record deletion and transaction commit in a single log record (perhaps even including transaction start), there is never any need to log undo information, i.e., the deleted record’s non-key contents.

Lock escalation and de-escalation are required or very desirable due to unpredictable concurrency contention and due to inaccuracy of cardinality estimation during query optimization. Lock escalation, e.g., from locking individual pages to locking an entire index, reduces overhead for queries with unexpectedly large results. It saves both invocations of the lock manager and memory for managing the locks.

Lock de-escalation reduces contention by relaxing locks held by active transactions. It requires that each transaction retain in transaction-private memory the information required to obtain the appropriate fine-grain locks. For example, even if a transaction holds an S lock on an entire index, it must retain information about the leaf pages read as long as de-escalation to page locking might become desirable or required.

Initial large locks combined with on-demand lock de-escalation can improve performance, because detail locks can be acquired without fear of conflict and thus without search in the lock manager’s hash table [GL 92]. Also note that management of such locks in transaction-private memory is very similar to management of locks in a shared lock manager’s hash table. Therefore, propagation of locks from private memory to the shared lock manager is quite similar to bulk updates of indexes, which has been found to improve fault rates in CPU caches (and in the buffer pool for on-disk indexes) and thus performance and scalability.

For large index-order B-tree scans, interior B-tree nodes guide deep read-ahead. Therefore, accessing those nodes and their separator keys do not incur any extra I/O, and it is conceivable to lock those nodes or their separator keys if desired.

Some operations benefit from fence keys, i.e., when a node is split, copies of the separator key posted in the parent are retained in the two sibling nodes. Fence keys aid B-tree operations in multiple operations, including key range locking and defragmentation [G 04]. Prefix and suffix truncation minimize fence key sizes [BU 77].

All B-tree entries are unique. It is not required that the declared search keys are unique; however, the entries must have identifying information such that a row deletion leads to deletion of the correct B-tree entry. Standard solutions add the row pointer to the sort order of non-clustered indexes or a “uniquifier” number to keys in clustered indexes.
Finally, locks on the low boundary of a gap protect the gap, e.g., against insertion of phantoms [GLP 76]. In other words, the description below employs “prior-key locking” rather than “next-key locking.”

4 Traditional locking hierarchies

Traditional locking in B-tree indexes is the basis with which the two new designs of hierarchical locking will be compared. Locking of units larger than individual B-trees, e.g., databases and tables, is ignored in this discussion, because locking those does not affect hierarchical locking within B-trees. Partitioning is also ignored; if a table or an index is partitioned, the focus is on the B-tree that represents a single partition of a single index. If such a B-tree cannot be locked because it can only be locked implicitly by locking a larger unit such as a table or an index with all its partitions, the other database contents covered by those locks is ignored and the lock is called an index lock nonetheless.

The standard techniques lock the index and then leaf pages, keys, or both. The choice among page and key locks can be built into the software or might be available to database administrators. In the latter case, a change requires a short quiescence of all query and update activity for the affected table or index.

4.1 Locks on keys and ranges

Both proposed methods for hierarchical locking in B-trees depend on key range locking. Therefore, it may be useful to first introduce a simplification for traditional key range locking that relies more directly on traditional multi-granularity locking [GLP 75, K 83], avoids irregular complexities such as instant locks and insert locks [L 93], and increases concurrency when applied to keys in B-tree leaves. While helpful for hierarchical locking in B-trees discussed in later sections, it also is a simplification and improvement for traditional key range locking applied solely to keys in leaves.

For maximal concurrency and for maximal simplicity, hierarchical lock modes on keys should be designed so that a key and a key range together can be intention-locked and such that the gap between two keys and an individual key value can each be locked separately. In order to optimize performance of lock acquisition and release, it makes sense to implement not two levels of lockable resources (comparable to the standard example with pages and files) but as a single resource with many possible lock modes [L 93]. This technique should be an implementation optimization only rather than lead to the introduction of new locking semantics such as range insert locks.

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>X</th>
<th>IS</th>
<th>IX</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>X</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IS</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IX</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 1. Traditional lock compatibility matrix.

Figure 1 shows a traditional lock compatibility matrix. For this discussion, the intention locks apply only to the combination of key value and gap between keys. The absolute locks can apply to the key value, the gap between keys, or their combination.
Using the locks in this lock compatibility matrix, the key value, the gap between keys, and the combination of key value and gap are three separate resources that must be locked using different locks and thus separate invocations of the lock manager code. The optimized design exploits the fact that the key value serves as the identifier for each of the three resources in the lock manager’s hash table, which permits employing only one resource and one invocation of the lock manager at the expense of additional lock modes.

In the optimized design, which is strictly equivalent to traditional locking for separate resources, an S or X lock covers the combination of key value and gap between keys. The new lock modes in Figure 2 lock a key and the gap between two keys separately.

In order to avoid any possible confusion with combination lock modes such as SIX, which really represents two locks on the same resource such as a file, these new lock modes should be thought of and pronounced with the words “key” and “gap” added. For example, SO should be pronounced “key shared” and SX should be pronounced “key shared, gap exclusive.” Intention locks on the combination of key and gap, though implied by these new lock modes in the obvious way, are not part of the name.

The new lock mode SO locks the combination in IS mode, the key in S mode, and the gap not at all. A OS lock on key Ki locks the open interval (Ki, Ki+1) yet keeps the key value Ki unlocked. SX locks the combination in IX mode, the key in S mode, and the gap in X mode. A lock held in this mode is most likely the result of a lock conversion rather than a single request; as is the XS mode. All other lock modes are defined similarly.

Figure 2 shows the lock compatibility matrix for the new lock modes. It is important to stress that these lock modes are merely a straightforward application of the traditional theory of hierarchical locking [GLP 75, K 83], and thus less complex and less prone to mistakes by developers during implementation and maintenance than traditional key range locking [L 93, M 90].

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>X</th>
<th>SO</th>
<th>OØ</th>
<th>XØ</th>
<th>OX</th>
<th>SX</th>
<th>XS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>X</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SO</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>OØ</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>XØ</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>OX</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SX</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>XS</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Figure 2. Lock modes for key values and gaps.

These new lock modes are identified by a key value but they really represent traditional locks on the key value itself, the gap between key values, and the combination of key value and gap. IS and IX modes are not shown because they do not apply to key values in this scheme other than implied by the new lock modes. The derivation of the values in the compatibility matrix follows directly from the construction of the lock modes.

For example, SO and OØ are compatible because both take intention locks on the combination of key value and gap, and one transaction locks the key value in S mode whereas the other transaction locks the gap in X mode. Similarly, XØ and OØ are compatible, as are SO and SX. A specific use of the OØ lock is to ensure absence of a key without inhibiting updates on the locked key. An alternative design inserts a ghost key for the absent key and retains a lock only on the specific key value.
New lock modes based on other basic modes, e.g., update locks [GR 93, K 83] or escrow locks [GZ 04, O 86], can be added in a similar way. In fact, it seems promising to integrate automatic derivation of lock modes such as SØ from a basic set of operational lock modes (e.g., S, X) and the set of components identified by the same database item (e.g., key value, gap between keys, and their combination) in a way similar to Korth’s lock derivation techniques [K 83]. Automatic derivation seems far more promising for the lock modes in Figure 2 than for the lock modes of traditional key-range locking.

The strict separation of locks on keys and gaps enables concurrency not supported by traditional key range locking. The earlier discussion of lock modes omitted in SQL Server [M 06] includes several examples of concurrent operations that are correct but are not enabled. The new design enables all of them. The essential reason is that the presented design follows the traditional theory for hierarchical locking and thus provides fully independent locks on keys and gaps. Nonetheless, SQL Server and the proposed design require the same number of new lock modes (6) and of lock manager invocations.

4.2 Key insertion using ghost records

Insertion of a new key and record can be implemented as a system transaction. This is analogous to key deletion implemented by turning a valid B-tree record into a ghost record with asynchronous ghost removal using a system transaction. The system transaction for key insertion leaves behind a ghost record that the user transaction can lock in key value mode and then turn into a valid record to effect the logical insertion [GZ 04].

The value of using a system transaction for key insertion is that it releases its lock on the gap between pre-existing keys when it commits. The user transaction holds a lock only on the new key value, not on the gap. Holding this XØ lock until it commits, the user transaction can modify the record from a ghost to a valid record. In the case of a rollback, the ghost record remains and can be removed like any ghost record, e.g., upon request by a future user transaction that requires more space in the B-tree leaf.

In the meantime, another user transaction or a system transaction may lock the gap defined by the new key. Using a ØX lock on the new key (really locking the gap only), the other transaction may insert yet another key, for example. Thus, this design supports high insertion rates within a page and an index, without any need to introduce irregular complexities such as instant locks and insert locks [L 93] that do not fit the traditional theories of two-phase locking and of hierarchical locking [GLP 75, K 83].

Moreover, the system transaction locks only the gap into which it inserts the new key; it has no need to lock the pre-existing key value that identifies the gap. Thus, another transaction can concurrently read the record with the pre-existing key. It can update and even delete the record if deletion is implemented by turning the valid record into a ghost record. Neither concurrent readers nor concurrent updates are possible in locking schemes without locks on only the open interval between keys, e.g., traditional key range locking [L 93, M 90, M 06]. The system transaction inserting a new key into a gap is compatible with any of these actions on the pre-existing key as long as the key is not erased from the B-tree. Thus, interference among neighboring keys is minimized by use of ghost records and system transactions.

---

4 In fact, since a rollback operation never needs to roll back itself, the ghost can be removed as part of rolling back the user transaction, if desired.
This system transaction can be very efficient. Forcing the transaction log on commit is not required [GZ 04], and a single log record can cover transaction start, key insertion, and transaction commit. In workloads that append many B-tree keys with predictable keys, e.g., order numbers in an order-entry application, a single system transaction can create multiple ghost records such that multiple user transactions each can find an appropriate ghost record that merely needs updating, without creating a new record or a new lockable resource. Note that this technique applies not only to unique indexes on order numbers but also to indexes with multiple records for each order number, e.g., an index on order details or a merged index holding both orders and order details [G 07].

4.3 Splitting and merging key ranges

Consider a user transaction that first ensures the absence of a key value and then attempts to insert a new record with that key. In the first step, the user transaction obtains a shared lock on a key range. The second step requires an exclusive lock on that range, albeit only briefly. If the second step employs a system transaction, the two transactions and their locks conflict with each other.

One possible solution is to ignore lock conflicts between a system transaction and the transaction that invoked it – this can lead to complex conditions if one or both of the transactions roll back. Another possible solution is to avoid the system transaction in this special case, letting the user transaction perform the entire insertion – this requires complex code to recognize the correct cases.

The preferred solution is to enable inserting a new ghost key into an existing key range even while it is locked. In order to ensure that existing transactions are not materially affected, their locks on the key range must be duplicated such that they cover the two ranges resulting from the split as well as the newly inserted key. In the specific example above, a new ghost record and its key are inserted and all locks on the original key range are duplicated for the newly created key range including the new key itself.

The newly created locks are inserted into the lock manager and into the appropriate transactions’ data structures, with a few differences to the standard procedures. First, lock conflicts with concurrent transactions are not possible, so there is no need to search the lock manager’s hash table for conflicting locks. Second, these locks cannot be released during a partial transaction rollback to a savepoint. Instead, these locks must be retained during rollback, unless the lock on the original range is also released. Third, if locks are recorded in the transaction log for re-acquisition during log analysis after a crash, these newly created locks appear at an appropriate point in the recovery log with respect to the locks they protect in the next-lower B-tree level. Thus, the recovery procedure must be augmented to ensure that the lock hierarchy is respected during log analysis and recovery.

While perhaps tempting, the reverse operation is not always correct. A ghost record must not be erased while it is locked, because the lock might indicate that deletion of the key is not yet committed. If, however, only key ranges are locked but the ghost record’s key value is not, and if the locks on the ranges below and above the ghost key do not conflict, then the ghost record can be erased and sets of locks combined.
4.4 Summary

The presented design is, in a way, “radically old.” The contribution is identifying appropriate existing techniques and combining them into a simple and sound design that permits more concurrency than traditional key range locking.

The proposed design applies the theory of multi-granularity and hierarchical locking to keys and gaps in B-tree leaves. By doing so, it permits concurrency not supported by prior designs, yet it eliminates multiple counter-intuitive lock modes. It then applies a second existing technique to reduce the count of lock manager invocations by introducing derived lock modes. The proposed design benefits from this technique as much as traditional designs for key range locking. Finally, introduction of new B-tree keys using system transactions and ghost records simplifies the implementation of highly concurrent insertions and mirrors a standard implementation technique for key deletion.

5 Locks on separator keys

Key range locking can be applied more broadly than it is today. Specifically, in addition to its use in B-tree leaves, it can be used in interior B-tree pages. A lock on a separator key represents the entire key range from one separator key to its next neighbor (at the same B-tree level). Thus, a key range lock at the level of the leaves’ parents is similar in some ways with a traditional lock on an entire leaf page. Both intention locks and absolute locks can be employed at that level, again similar to intention locks and absolute locks on B-tree leaf pages in the traditional hierarchy.

Like key range locks on keys in B-tree leaves, key range locks in non-leaf B-tree nodes are special forms of predicate locks [EGL 76]. Arguments and proofs for the correctness of key range locking in non-leaf B-tree nodes are based on established techniques for predicate locks. For example, splitting and merging of nodes and of ranges must ensure that appropriate predicate locks are preserved and protected.

Locking key ranges at the parent level results in locking overhead and concurrency behavior similar to traditional locking of leaf pages. A key range lock at the grandparent level, however, covers a much larger key range. Depending on a node’s fan-out, a range lock at the grandparent level may save tens or hundreds of locks at the parent level, just like a key range lock in a parent node covers tens or hundreds of individual keys. For even larger B-tree indexes and queries, locking can start even higher in the tree.

Each level adds some overhead for acquisition and release of intention locks, even for transactions that actually touch only a single key. Fortunately, the B-tree level at which locking starts can be tuned according to the data and the application at hand, from locking keys in the leaves only to locking a key range at every level from the root to the leaves. Mechanisms for changing this level dynamically and online are discussed below.

5.1 Techniques

In contrast to locking in B-tree leaves, it is less useful in interior B-tree nodes to separate locks on key values and locks on gaps between keys. Thus, only standard lock modes (Figure 1) apply to those keys, and cover the half-open range from the locked separator key (inclusively) to the next separator key (exclusively).
A “high fence key” in each B-tree node, i.e., a copy of the separator key in the node’s parent, may delimit the range protected by a lock on the highest key value in a node. Some B-tree implementations already retain such a fence key, e.g., \( B^{\text{link}} \)-trees [LS 97]. Similarly, a “low fence key” may readily provide the key value to lock in order to protect the key range of a node’s left-most child. Thus, each node should also retain a “low fence key.” Many systems already retain this key [GR 93] even if they do not employ fence keys for multiple purposes [G 04].

Locks on separator keys in parents are different from locks on keys in leaves, even if the key values are the same. Thus, their identifier in the lock manager includes the index identifier, the key value, and the node level within the B-tree. Node levels are commonly stored in each node’s page header, with level 0 assigned to leaves, if for no other reason than for use during physical consistency check of the tree structure. Note that this identification of locks does not refer to physical location, e.g., a page identifier, such that locks remain valid if a separator key migrates when a node splits or when two nodes merge.

### 5.1.1 Splitting and merging nodes

When a node splits, a new separator key is inserted in the node’s parent. The required steps with respect to existing locks are precisely those necessary to insert a new ghost key into a key range even while it is locked. This procedure permits page splits at all B-tree levels without interrupting the flow of transactions and ensures that no transaction’s locked range is modified due to a change in the physical B-tree structure.

When two nodes merge, the opposite procedure applies. As two key ranges merge in the parent node, transactions end up holding locks on the combined range, as if they had locked both original ranges equally. It is required to verify, therefore, that the combined set of locks does not include conflicts, e.g., an S lock on one original range and an IX lock on the other original range\(^5\). In such cases, the merge operation must be delayed.

Load balancing between two neighboring nodes can be accurate in its locking but fairly complex due to the shift in key ranges, or it can be made fairly simple. The accurate method determines the locks needed on the parents from the locks held on the child entries that migrate. The simple method models load balancing as a merge operation followed by a split. The actual implementation and manipulation of data structures is, of course, not tied to these steps. The likely result is that some transactions hold more locks than they need. However, given that load balancing operations are rare (if implemented at all) and that the alternative procedure is complex, the simple procedure is preferable.

### 5.1.2 Lock escalation and de-escalation

While mechanisms to split and merge key ranges are required, lock escalation and de-escalation are optional, although they add significant value to locks on separator keys.

By default, locking may start with key range locks in the leaves’ parents, which is most similar to traditional page locking. Alternatively, it may start at the grandparent

\(^5\) In the instant of merging the two ranges, transactions do not conflict even if their lock modes do. Thus, it might seem safe to permit merge operations even if locks conflict. However, transactions might perform additional operations based on locks on the combined range, and conflicts among these additional operations would remain undetected.
level in order to cover larger key ranges and require fewer locks in large range scans. For
the discussion here, assume that initially locking starts with intention locks on separator
keys at the grandparent level (e.g., IS locks) and continues with absolute locks on separa-
tor keys at the parent level (e.g., S locks). In the B-tree sketched in Figure 3, this means
intention locks (IS or IX) on key ranges in grandparent node a, absolute locks (S or X) on
key ranges in parent node c, and no locks in leaf nodes e, f, and g.

![Figure 3. Lock escalation and de-escalation.](image)

If so, either lock escalation (to absolute locks in the grandparents) or de-escalation
(to intention locks in the parents and absolute locks in the leaves) may be required. Which
one is required at what time depends on the workload, and in fact may differ
within a single B-tree, e.g., an index on some time attribute in a data warehouse.

For lock escalation, the intention lock at the grandparent must become an absolute
lock. In Figure 3, the only remaining locks will be absolute locks (S or X) on key ranges
in grandparent node a. The locks at the parent level, now obsolete due to the absolute
lock at the grandparent level, may be erased. Alternatively, they may be retained in the
lock manager or in transaction-private memory for future lock de-escalation, which may
be desired due to increasing contention or due to a partial transaction rollback.

For lock de-escalation, the absolute locks at the parent level must become intention
locks. In Figure 3, intention locks will be used in grandparent node a as well as parent
node c, and absolute locks will be used only in the leaves e, f, and g. This requires the
opposite procedure and relies on proactive gathering of leaf-level key locks omitted dur-
ing initial processing. After these leaf-level locks have been propagated from transaction-
private memory to the global lock manager’s hash table, the lock on the separator key in
the parent can be relaxed to an intention lock.

Partial transaction rollback probably should not reverse lock de-escalation because
reversal requires upgrading a lock (e.g., from IS to S in node c in Figure 3), which might
fail due to locks held by concurrent transactions. In fact, after lock de-escalation due to
contention by concurrent transactions, it is likely that such a lock upgrade during partial
transaction rollback will fail.

5.1.3 Granularity changes

An entirely different kind of locking adjustment is a change in the lock hierarchy. In
the case of locking separator keys in a B-tree index, this means changes in the set of B-
tree nodes where key ranges must be locked. For example, the discussion above assumed
locks on separator keys in the leaves’ grandparents and parents but not the great-
grandparents. However, this can be adjusted dynamically and online, i.e., without disrup-
tion of transaction processing.
Techniques for granularity changes may be useful beyond locking in B-trees. They apply whenever a specific granularity of locking is to be activated or deactivated online. Nonetheless, modifying the granularity of locking is a new notion not used or described elsewhere, as are techniques for doing so online.

Like lock escalation and de-escalation, these techniques are not mandatory for a functional implementation of locks on separator keys. However, these techniques may be required to minimize locking overhead for both online transaction processing and large range queries, in particular for workloads that change over time.

It seems most convenient to augment each interior B-tree node with a label to indicate whether or not keys in that node must be locked. The label applies to each node individually, not to entire B-tree levels. In fact, it even seems possible to label each individual separator key, an idea to be explored in the future.

In Figure 3, for example, assume that grandparent node a is not labeled and no locks are taken there, and that parent node c is labeled such that key range locks are taken on separator keys there.

Setting this label on a node, e.g., on grandparent node a in Figure 3, can be either immediate or delayed. The advantage of the immediate method is that a transaction may request lock escalation even to a node in which no transaction has acquired locks yet; the advantage of the delayed method is less search in the lock manager’s hash table.

The immediate method searches in the lock manager’s hash table for active transactions that ought to hold intention locks. These transactions can be found by searching for locks on keys in the node’s children (e.g., nodes b, c, and d in Figure 3). For all such transactions, appropriate intention locks for the appropriate separator keys in the present node are inserted into the lock manager. While the search for such transactions is expensive, acquisition of the new intention locks is very fast as there is no need to search for conflicting locks (intention locks never conflict with other intention locks). The correctness of the immediate method relies on its correct acquisition of all intention locks that would have been held already if the label had been set on the node before any of the active transaction began.

The delayed method forces all new transactions to acquire intention locks and prevents acquisition of absolute locks in the node until all older transactions have completed. To do so, it employs two system transactions. The first one labels the node, thus forcing all future transactions to acquire intention locks while descending the B-tree. A second system transaction obtains IX locks on all keys in the node, observes the set of currently active transactions, waits until the last one of those has finished, and then commits. For the duration of these IX locks, no transaction may acquire an absolute lock. The correctness of the delayed method relies on a system transaction holding IX locks on all keys on behalf of all active transactions, whether those transactions require them or not.

When a non-leaf node is split, e.g., grandparent node a or parent node c in Figure 3, both resulting nodes inherit the label from the original node. A split operation can proceed even while a granularity change is on-going, whether the immediate or delayed method is employed. For two nodes to merge, they must be labeled equally first, i.e., a modification in the granularity of locking might be needed prior to a merge operation.

Removing a node’s label, e.g., on parent node c in Figure 3, also employs a system transaction that acquires IX locks on all keys in the nodes, erases the node’s label, and commits. If necessary, it waits for other transactions to release their absolute locks, if
any, on those keys. Requesting lock de-escalation by those other transactions permits faster completion; once their absolute locks are relaxed to intention locks, there is no conflict with the desired IX locks.

Both setting and removing a node’s label are very local operations that hardly affect data availability. Note that new user transactions can acquire intention locks within a node while a system transaction holds IX locks, because IX locks do not conflict with other intention locks, only with absolute locks.

The set of nodes labeled for locks on separator keys has few restrictions. For example, it is possible that a node is labeled but some of its siblings are not, or that a leaf’s parent is not labeled but its grandparent is (skip-level locking). These examples seem counter-intuitive, but they might be appropriate incremental states while an entire B-tree is converted to a different granularity of locking, or they might be permanent states tuned for skew in the data distribution or in the access pattern. Nonetheless, skip-level locking might not be a good idea because it raises the complexity of lock escalation, de-escalation, and granularity changes.

While powerful, these mechanisms require governing policies. For transaction processing systems, an initial policy might start with locking keys in the leaves only and locking separator keys in interior B-tree nodes only inasmuch as necessary for lock escalation. In other words, non-leaf nodes are labeled as described above only on demand. Similarly, labels are removed after their benefit ceases, i.e., nodes remain labeled only during demand. For relational data warehousing, locks in parent and grandparent nodes seem like a reasonable default.

5.2 Advantages

The most salient advantage of locking separator keys in non-leaf B-tree nodes is that the method scales with the size and the height of the index tree. The stepping factor is fairly uniform across all levels of the index, even in an index with a non-uniform key value distribution. Typical values for the stepping factor may be 100 to 1,000; the latter value requiring large pages or very short keys, e.g., due to aggressive prefix and suffix truncation. Thus, the database administrator or the automatic tuning component can adjust the granularity of locking very accurately, much better than in the rigid traditional locking hierarchy of index, page, and key.

In addition, lock escalation and de-escalation permit graceful adjustments to precision and overhead of locking during query execution, even in cases of inaccurate cardinality estimation during query optimization. For unpredictable or time-variant workloads, the nodes at which key range locks are taken can be adjusted with very little disruption of online transaction processing. Thus, there is a real promise that the method can be made automatic, incremental, online, yet robust.

Even if multi-level hierarchies are not exploited and ranges are locked only in leaf nodes and their parent nodes, the method has some advantages over traditional locking of leaf pages and leaf keys. Specifically, there never is any doubt about which lock covers which other lock. Thus, there is no need for lock migration during leaf splits and merges, with the attendant code simplification and performance improvements.
5.3 Disadvantages

Despite the advantages over traditional locking for B-tree indexes, some queries and update still suffer with this strategy for hierarchical locking. While simple range queries will likely need only tens or maybe hundreds of locks, or orders of magnitude less than with traditional hierarchical locking, more complex yet common query clauses may still require a thousand locks or more. Consider, for example, an index on columns (a, b) and the query predicate “where a in (3, 9, 15, 21)”. There are eight range boundaries to consider in this simple query, and each range boundary requires a choice whether to lock a larger range using few locks or to lock precisely using many fine-grain locks.

5.4 Opportunities

While the mechanisms described above are reasonably straightforward, their usage is an opportunity for further research, e.g., the number of locks required in average and worst cases. This analysis should cover not only selections but also joins, e.g., not only “between” queries but also “in” queries with lists of values or with nested queries. This evaluation ought to consider page splits near (rather than: at) a full node’s center, as already recommended for effective prefix truncation [BU 77].

Research into policies and automatic self-tuning – starting level, performance metrics to observe, thresholds for lock escalation, lock de-escalation, and changes in the granularity of locking – could speed adoption of the proposed mechanisms in commercial systems. Nonetheless, if there is a global switch to force locking of keys or pages only and to prevent changes in the granularity of locking, the proposed mechanisms can always be restricted to a behavior very similar to current systems, thus providing a safe fall-back if a novel automatic policy fails.

Finally, one might design an alternative concurrency control method in which transactions lock large key ranges by locking interior B-tree pages, i.e., physical pages instead of logical key ranges. During searches and updates of existing pages, this approach is similar to locking separator keys. It is quite different, however, with respect to lock migration during page split and merge operations, during defragmentation, and during write-optimized operation [G 04].

5.5 Summary

The proposed design for hierarchical locking in indexes exploits the tree structure of B-trees. Traditional absolute locks and intention locks on separator keys in interior B-tree nodes protect all entries in their key ranges. Data skew is considered automatically, just as it is in the insertion, deletion, and search logic of B-trees with respect to concurrency control. Workload skew can be accommodated by lock escalation and lock de-escalation.

In order to minimize locking overhead during transaction processing, locking can be limited to keys in a B-tree’s leaf level. Mixed operation, i.e., concurrent large queries and small updates in a real-time data warehouse, can be accommodated by adjusting the locking granularity dynamically. Modifying a database’s lock hierarchy online is a novel technique that may seem particularly well suited to key range locking in B-tree indexes but is applicable to any form of hierarchical or multi-granularity locking.
6 Locks on key prefixes

The prior method exploited the tree structure for multi-level hierarchical locking; the second alternative for hierarchical locking in B-tree indexes ignores the tree structure and instead exploits the key structure for the same purpose. Consider, for example, a B-tree index for a multi-column compound key with columns \((a, b, c, d)\) and with specific values \((a_0, b_0, c_0, d_0)\). Any leading prefix such as \((a_0, b_0)\) can serve as a resource that permits locking all index entries with keys starting with these specific values.

The promise of this method is that it matches precisely with query predicates, e.g., a query clause “where \(a = a_0\) and \(b = b_0\)”. More complex predicates map to only a few very precise locks. For example, a query with the clause “where \(a\) in \((a_0, a_1, a_2)\) and \(b\) in \((b_0, b_1, b_2, b_3)\)” requires only 12 locks, independent of the sizes of the table and its index. Thus, locking key prefixes may be competitive with predicate locking and precision locking but without any need for predicate evaluation for concurrency control.

At the same time, this method promises efficient locking for single-row updates, because it is possible to lock a single B-tree entry (recall that all B-tree entries are unique in order to permit accurate deletion and maintenance). Thus, hierarchical locking based on key prefixes promises to match very well with both queries and updates in both transaction processing and data warehousing.

Locking key prefixes matches so well with query predicates because it is, in fact, a special form of predicate locking. Thus, arguments and proofs for the correctness of locking key prefixes rely on established techniques for predicate locks [EGL 76].

6.1 Techniques

A lock on a specific value, say the two-column prefix \((a_0, b_0)\), covers all B-tree keys starting with these values. In order to prevent phantom records in serializable transaction isolation, however, non-existent key values also need locking.

In order to solve this problem, locks can cover only a specific prefix, the gap between two actual prefix values, or both. As in traditional key range locking in leaf pages, this can be modeled using an additional level in the locking hierarchy, e.g., those shown in Figure 1. For a smaller number of lock manager invocations, the actual implementation may use the artificial lock modes and their compatibility matrix shown in Figure 2.

Existence of specific key values in a B-tree can be decided only after a B-tree search has found the right index leaf. Thus, key locking starts only after the navigation from the B-tree root to the leaf. As discussed earlier, this navigation is protected by latches, not locks, including the inspection of the current leaf contents.

6.1.1 Insertion and deletion of values

While exact-match lookup in a B-tree, e.g., during an index nested loops join, benefits from precise locking and thus from key prefixes, insertion and deletion of keys might require key range locking. Specifically, during insertion, if a key prefix needs to be locked, the lock mode depends on the existence of a prior B-tree entry with the same prefix value as the new record. If such a prior B-tree entry exists, a key value lock suffices. If no such entry exists, a range lock on the prior key is needed. Note that this lock only
needs to cover the open interval between the two pre-existing actual prefix values; there is no need to lock either of these values.

The initial insertion can be a system transaction that only inserts a ghost record, leaving it to the user transaction to turn that ghost record into a valid record. While the system transaction requires a range lock, the user transaction needs to lock only the new key value, not the gap between keys. Thus, the key range is locked only for a very short time.

While this description is very similar to the earlier discussion of key range locking of unique keys in B-tree leaves, it applies here to key ranges defined by key prefixes. For example, if a locked key prefix matches the definition of complex objects clustered in the B-tree by common key prefixes [G 07], a new complex object can be inserted without ever locking its neighboring objects or their components.

Deletion follows a similar pattern. A user transaction might simply mark a B-tree entry a ghost, leaving it to a system transaction to erase the record from the B-tree, or a user transaction might immediately erase the record and the key. Erasing a record requires a lock on the record’s key in exclusive mode, thus ensuring that no other transaction holds any kind of lock on it. Moreover, the gap between two neighboring keys must remain locked in order to ensure successful transaction rollback if required. Gaps must be locked at all levels in the hierarchy for which a distinct value disappears.

6.1.2 Lock escalation and de-escalation

Whereas insertion and deletion of values are required, lock escalation and de-escalation are optional, as are granularity changes to be discussed shortly.

While key insertion and deletion are slightly subtle, lock escalation and de-escalation are particularly simple in this design, for two reasons. First, the granularity of locking and the currently active components of the lock hierarchy are uniform across the entire index. Second, structural modifications such as page splits of the B-tree do not affect the set of locks required. Both reasons, of course, are based on the complete separation of physical B-tree structure and locking hierarchy.

The sequence of locking granularities can be defined in very general ways. For example, it is not required that each column in the key defines a lock granularity. Skipping a column might be advisable if the column has only very few distinct values. A column with an extremely large number of distinct values could conceivably be modeled using multiple lock levels, e.g., the first three digits in a zip code or in a social security number can be modeled as a separate level in the locking hierarchy by treating them as a column in their own right, even if only for the purposes of locking.

Given these considerations, lock escalation and de-escalation follow the fairly obvious pattern. For lock escalation, an intention lock is upgraded to an absolute lock for the appropriate short key prefix, and the prior absolute locks can be forgotten or retained for possible subsequent lock de-escalation. For de-escalation, appropriate information must be retained during the transaction’s prior activities such that appropriate locks on longer prefixes can be inserted into the lock manager’s hash table, and the absolute lock on the short prefix can be downgraded to an intention lock.
6.1.3 Granularity changes

When locking separator keys in interior B-tree nodes, modifying the granularity of locking affects one node at a time. When locking key prefixes, the granularity of locking must be modified uniformly for the entire index. Thus, changes in the granularity of locking affect the entire B-tree, even for very large indexes.

With this exception, the procedure for changing the granularity of locking is quite similar for the two new strategies for hierarchical locking in B-tree indexes. When adding a new granularity of locking, both immediate and delayed methods can be designed.

In the delayed method, a system transaction locks all distinct key values at the new granularity of locking across an entire index. In contrast, hierarchical locking based on separator keys can modify the granularity of locking one B-tree node at a time.

Enumeration of all distinct values for a specific key prefix is a standard problem in relational query processing, e.g., for a query such as “select distinct a from …” based on an index on columns (a, b). While standard solutions exist under various names, e.g., as multi-dimensional B-tree access [LJB 95], the run-time expense seems impractical for adding a granularity of locking to a B-tree index.

The immediate method analyzes existing locks in order to acquire only those locks that are truly needed for specific active transactions. If hierarchical locking based on key prefixes is more precise than alternative locking methods and thus there are fewer locks to consider, this analysis can be faster than in hierarchical locking based on separator keys. An important difference, however, is that granularity changes apply to locking on separator keys one node at a time, whereas they apply to locking key prefixes for an entire index all at once.

Therefore, the delayed method does not seem promising for hierarchical locking based on key prefixes, and only the immediate method seems feasible, albeit not very practical. This is in contrast to hierarchical locking of separator keys, where both methods seem practical and appropriate in different circumstances.

Removal of an existing granularity of locking needs to wait for the release of all absolute locks at that granularity. A possible implementation technique is to wait for an IX lock competing with each such absolute lock held by a user transaction. Identifying all such locks requires search either in the lock manager’s hash table or in the index.

To summarize, while it is possible to adjust the granularity of locking when locking key prefixes and when locking separator keys, the implementation mechanisms and tradeoffs are quite different. Whereas locking separator keys only on demand and only during demand appears to be a promising approach, a similarly promising approach is not apparent for locking key prefixes.

6.2 Advantages

Hierarchical locking of key prefixes has several very attractive characteristics. All of them reflect the match between query predicates and key prefixes. One overall advantage is that all of these characteristics are achieved with a single mechanism. This mechanism permits locking large ranges or individual B-tree records, and thus supports equally well decision support, transaction processing, and applications with similar access patterns.

First, this design matches well with equality predicates and “in” predicates, which are very common both in single-table selections and in joins. A single lock can often
cover precisely the index entries needed, such that consistency and serializability among such a set of index entries are guaranteed, even in transaction isolation levels weaker than strict serializability. This aspect makes the locking strategy ideal for non-unique indexes including indexes on foreign key columns. If the leading key column is not specified and multiple index probes are required [LJB 95], a single lock per probe will suffice.

Second, if master-detail clustering is supported based on equality of search columns in a B-tree [G 07], locking key prefixes permits locking complex objects. For example, if orders and order details are co-located within a B-tree index based on equal values in the common column of order numbers, locking an order number locks an entire order object and all its component records. A single lock can also cover a large object, e.g., a customer with multiple orders, shipments, invoices, and payments. Insertion and deletion of one complex object does not require any lock on its neighbors.

Finally, locking key prefixes is a good match for partitioned B-trees [G 03b], i.e., B-trees with an artificial leading key column that indicates partitions in indexes and run numbers during sorting. The first granularity of locking is the partition identifier, which permits efficient operations on partitions such as merging partitions, creating a partition during data import, or dropping a partition while purging data from a data warehouse.

### 6.3 Disadvantages

This approach to hierarchical locking in B-tree indexes also has some disadvantages. Their common theme is the difficulty to set good defaults for the granularity of locking.

First, there is the danger of needing thousands of individual locks for thousands of distinct values, e.g., in index nested loops join or other forms of nested iteration. Thus, this locking strategy seems practical only if lock escalation is supported. Even then, a leading column with millions of unique values may be troublesome without dynamic changes in the hierarchy of locks.

Second (and amplifying the first point), the number of unique values may not be known due to old statistics or due to a missing range predicate on the leading index column. The number of unique values may be surprisingly large, leading to many individual locks. It may also be surprisingly small, leading to coarse locks that might restrict concurrency more than anticipated or desired.

Third, the effect of data skew are unclear, yet skew seems ubiquitous in real data.

Finally, there does not seem to be an opportunity to adjust the locking strategy locally to match any skew in the actual data values or in the workload. In order to achieve local adjustments in the granularity of locking similar to those described for key range locking on separator keys, an additional control table is needed that indicates which specific key prefixes should be locked with a granularity different than the default. Introduction of a control table for concurrency control seems like an unacceptable increase in the complexity of implementation, testing, maintenance, tuning, user education, and post-deployment vendor support.

### 6.4 Opportunities

Some of the disadvantages above could possibly be addressed by locking not only prefix columns but parts thereof. For example, if a column is a four-byte integer, each individual byte could be interpreted as a column of its own for the purpose of locking.
This idea is reminiscent of bit-sliced indexes [OQ 97] and, like those, can interact beneficially with range predicates.

This idea might alleviate the problem of key columns with a very large number of unique values, but it does not address data skew. Keys could be compressed, however, e.g., using order-preserving Huffman coding, in order to maximize the entropy of each stored bit. In that case, data skew might be less of a problem, although skew in the access pattern remains an unsolved issue.

Another opportunity based on disassembling a column is to apply hierarchical locking based on key prefixes not only to indexes but also to heap files. Specifically, components of a record identifier including file identifier, page identifier, slot number, and their individual bytes are treated as if they were individual columns in a compound B-tree index. Locks may cover predefined static ranges or dynamic ranges. In the former case, even ranges as yet unused require locks, i.e., files and pages ranges with no pages yet allocated for the relevant heap. In the latter case, the information about actual ranges covered by each lock would need to be inferred from the data structures used for space allocation. While very preliminary, this might be the first proposal for scalable hierarchical locking in heaps.

6.5 Summary

Hierarchical locking based on key prefixes is promising because locks match query predicates as well as complex objects in master-detail clustering. Insertion and deletion of unique values can benefit from the lock modes of Figure 2. Lock escalation and de-escalation can readily be supported. Adjusting the granularity of locking is not particularly complex if it is applied uniformly to an entire index, which unfortunately seems expensive and thus impractical. Nonetheless, if adjusting the granularity is not required and if lock escalation and de-escalation are sufficient dynamic mechanisms, locking key prefixes seems a viable alternative or complement to other locking strategies.

7 Summary and conclusions

Traditional hierarchical locking is starting to fail for very large indexes – the stepping from a single page to an entire index is too large. The dilemma between locking an entire index and locking millions of individual pages or keys demands a resolution.

The two alternative solutions introduced in this research have their distinct advantages and disadvantages when compared with the traditional design or with each other. Both new designs scale more gracefully than the traditional design due to their intermediate incremental stepping between index lock and page or key lock.

Key range locking in interior B-tree nodes scales with additional B-tree levels and adapts to skewed or unpredictable key distributions. The additional stepping levels between index and key eliminate the dilemma between massive overhead due to many individual locks and massive contention due to a single index lock. Moreover, splitting pages at keys that minimize the size of truncated separator keys [BU 77] matches key ranges in parent and grandparent nodes to clusters of records and complex objects.

Locking prefix values of search keys more obviously matches query predicates and complex objects in master-detail clustering, in many cases comparably with predicate locks and precision locks. However, it suffers in cases of very few or very many distinct
values and in cases of data skew. Moreover, it does not permit efficient local adjustment of the granularity of locking. Thus, key range locking in parent and grandparent nodes is more practical as a general solution than locking prefix values of search keys.

In addition, this research is the first to consider dynamic changes in the lock hierarchy in response to skew in the data or in the workload, and to propose specific algorithms for doing so. In fact, these algorithms are online and incremental such that they minimize disruption to ongoing transaction processing.

These dynamic changes can be exploited in many ways. For example, to maximize performance and scalability of transaction processing, only keys in leaf nodes are locked by default, with overhead, performance, and scalability comparable to traditional key value locking. If an actual transaction would benefit from a larger granularity of locking, e.g., locking the key range of an entire leaf page, key range locking can be introduced specifically in those parents and grandparents that benefit an actual transaction. When no longer advantageous, that granularity of locking can be abandoned to maximize transaction processing performance again.

For some applications and their indexes, the ideal locking strategy might be a combination of these techniques. Multi-granularity locking permits each element to participate in multiple hierarchies. For example, some readers may lock only keys and key ranges based on leaf records and separator keys, while other readers lock key prefixes. This is permissible if writers acquire appropriate locks in all hierarchies. Such locking strategies might be particularly useful in real-time data warehousing with concurrency among large queries and small incremental “trickle” updates, as well as in partitioned B-trees and their use cases [G 03b]. Even other hierarchies not considered in this research might be included, e.g., key ranges based on key ranges with fixed boundary values [L 93] or even multi-dimensional ranges in temporal or spatial data and applications.

Finally, this research simplifies and improves key range locking. The presented proposal simply applies the traditional sound theory for hierarchical locking to keys, gaps between keys, and the combination of key and gap. It avoids unconventional lock modes that violate traditional two-phase locking and traditional hierarchical locking. The new lock modes in Figure 2 are performance techniques with no new semantics. They ensure the same number of lock manager invocations as used in traditional key range locking. For high concurrency during both key insertions and removals, the proposed design employs system transactions. These transactions can use ordinary lock modes and lock retention to transaction commit. In combination, these techniques simplify key range locking without introducing new overhead. Equally important, they correctly admit more concurrency than traditional key range locking and may thus be useful in all database management systems.

Acknowledgments

Jim Gray has been very helpful and encouraging. Phil Bernstein, Barb Peters, and the reviewers contributed helpful suggestions that improved the presentation of the material.
References


Wissenschaftliches Programm
Efficient Time-Travel on Versioned Text Collections

Klaus Berberich, Srikanta Bedathur, Gerhard Weikum

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
{kberberi, bedathur, weikum}@mpi-inf.mpg.de

Abstract: The availability of versioned text collections such as the Internet Archive opens up opportunities for time-aware exploration of their contents. In this paper, we propose time-travel retrieval and ranking that extends traditional keyword queries with a temporal context in which the query should be evaluated. More precisely, the query is evaluated over all states of the collection that existed during the temporal context.

In order to support these queries, we make key contributions in (i) defining extensions to well-known relevance models that take into account the temporal context of the query and the version history of documents, (ii) designing an immortal index over the full versioned text collection that avoids a blowup in index size, and (iii) making the popular NRA algorithm for top-k query processing aware of the temporal context. We present preliminary experimental analysis over the English Wikipedia revision history showing that the proposed techniques are both effective and efficient.

1 Introduction

Numerous versioned text collections are available today, starting from Wikis where textual content is explicitly version-controlled, to the World Wide Web whose dynamic content is only partly archived through efforts such as the Internet Archive [IA]. Similarly, information feeds of Web portals, Weblogs, etc., with their time-stamping mechanism can also be regarded as versioned text collections.

Despite the ubiquity of such versioned text collections, querying and ranking over them is typically only done on the most recent snapshot of the collection. Although this suffices for many simple information needs, certain information needs cannot be adequately answered. Consider the following examples:

1. Finding historical statutes is one of the common information needs of attorneys [Coh]. With almost all public documents of the state now available on the Web, it is natural to use the historical Web to unearth these documents rather than to wade through libraries. Ideally, a query “Indiana court rules” and a time-window of interest should suffice to retrieve relevant documents and thus to speed up legal research.

2. In a case of copyright-infringement a lawyer needs to collect evidence about the illegal copying of copyrighted material (e.g., a piece of writing) on the Web.
suspect has removed the copied material in the meanwhile. Ideally, a query would consist of a snippet of the copyrighted material and the time-window during which the copied material was available.

3. For discovering a population affected by a disease, say HIV, a public health administrator could query a database of electronic health records [ISO] containing longitudinal record of health history of individuals, with keywords relating to symptoms of the disease – e.g., “swollen lymph nodes”, “sore throat” etc., and a time-window of interest – say, in the last 3 years, and expect to be presented a list of candidate records that should be tested in detail.

Similar situations where keyword querying needs to be combined with a time-window of interest also arise for other kinds of business data such as customer-support databases, CRM databases, patent records, etc. Today’s search engines cannot provide adequate results for such temporally contextualized information needs. The relevant (versions of) Web pages have disappeared and are therefore not contained in the current snapshot that is considered. Though Web archives may contain these relevant versions, they provide only limited access functionality such as the Wayback Machine [IA] requiring the knowledge of URLs or extensive browsing within the archived contents.

The idea of querying time-varying relational databases has been well-studied in temporal database research [TCG+93]. However, there has been very little work in translating those ideas into the context of versioned text collections where keyword querying, not SQL, is the query interface. Further, unlike in standard relational databases, the results of textual querying need to be ranked taking into account the relevance to the given keywords.

1.1 Time-travel in Text Search

In this paper, we introduce the notion of time-travel querying and ranking aimed at supporting temporally contextualized information needs such as those mentioned above. Informally, our time-travel query model adds a time-window of interest to the standard keyword-based query model. The given keyword query should be evaluated and ranked over the state(s) of the text collection valid during the given time-window.

A straightforward approach to supporting these time-travel queries would be to periodically generate a new, updated index, and retain the old index. Whenever a query specifies a time-window, consult the indexes that are valid for that time-window and answer the query. Although this approach is easy to implement, it clearly entails a complete replication of the index even when many documents have undergone none or only minor changes. Even if one may reduce the index size with a variety of compression techniques, it is not clear how to score and rank the documents (based on their versions) when the time-window of interest spans more than one index instance. A document could be relevant with high score in one index, while in the subsequent index it could no longer be relevant or could have a different relevance score based on content changes in the meanwhile. The relevance score of a document could vary not only due to local changes within the document, but also due to collection-wide changes that affected the eliteness of a query term. It is
not clear at a glance how to reconcile these variations in score to generate a meaningful document-level result ranking for the time-window of interest.

The key technical challenges in supporting time-travel queries over large versioned text collections are:

1. **Indexing**: As we pointed out above, if each version of the document in the text collection is independently indexed, many entries are replicated unnecessarily. It is essential to develop an indexing strategy that eliminates this redundancy and controls the associated space explosion.

2. **Ranking model**: Commonly used ranking models such as tf-idf and Okapi BM25 are only applicable to static text collections. However, in the case of time-travel queries, the ranking model has to generate a document-level ranking that takes into account the variations in score of individual versions that are valid within the time-window of interest.

3. **Query processing**: Users are typically interested in quickly obtaining few top results to the given time-travel query. Therefore, it is important to develop efficient query processing techniques to identify such top-\(k\) answers, in a spirit similar to the popular family of Threshold Algorithms [FLN03].

### 1.2 Contributions

In this paper, we explore in detail the issues surrounding the explicit incorporation of a temporal context in text retrieval. We introduce and formally define time-travel querying as a mechanism for keyword-based searching over versioned text collections. In order to efficiently answer these queries, we develop an immortal text indexing technique. In contrast to prior work in searching on versioned text [AF92, NN06], our approach ensures fairness in access costs irrespective of the age of relevant documents and the position of the time-window of interest. Finally, we propose a novel time-contextualized ranking framework that goes well beyond the simple boolean querying supported in [NN06].

In summary, the key contributions made in this paper are the following:

1. A relevance model for versioned document collections is introduced. We consider the document-level result granularity, with each document being a sequence of its associated versions. This is in contrast to the version-level granularity treated in earlier research.

2. We present an indexing system called I2T2 (Immortal Index for Time-Travel) that enriches the traditional inverted file index with temporal information and allows for result ranking. We decouple contributions from local (term-document-specific) and global (term-collection-specific) statistics to the relevance score, and utilize this to reduce the index size.
3. In order to further overcome the index-size blowup on account of negligible changes between versions, we introduce approximate temporal coalescing of index entries. We empirically show that this can achieve significant reductions in index size while retaining good accuracy.

4. Addressing the issue of query processing over our extended index, we present TC-NRA (Time Context aware NRA), an adaptation of the NRA algorithm proposed by Fagin et al. [FLN03], to generate top-k answers efficiently for time-travel queries.

5. Finally, we present a comprehensive experimental evaluation of the proposed techniques over the English Wikipedia revision history.

1.3 Organization

The remainder of the paper is organized as follows. Section 2 puts our work in context of related work. We introduce our data and query model, and then present our relevance model in Section 3. Following that, in Section 4, we describe the I2T2 indexing system. The top-k query processing algorithm TC-NRA is presented next in Section 5. The proposed techniques are comprehensively evaluated in a series of experiments that is detailed in Section 6. Finally, in Section 7, we conclude the present work and outline future directions of research.

2 Related Work

There is only scarce work that deals specifically with temporal text indexing – the main problem considered here. Anick and Flynn [AF92], as a first notable exception, describe a help-desk system that enriches inverted file indexes to allow for retrieval of historical document versions. The system supports historical queries both for a given point in time or time span. Access costs are optimized for accesses to the most recent versions and increase the farther one moves into the past, thus making the approach inappropriate for the scenario considered in this work. Nørvåg and Nybø [NN06] concentrate on text-containment queries only and do not consider the ranked retrieval case.

Textual querying is typified by the need for effective ranking of results and has been the focus of research for more than 30 years. The tf-idf scoring model is traditionally used a as measure of a document’s relevance to the given query keywords. This scoring model combines the term frequency (tf) and the logarithmically dampened inverse document frequency (idf) of the term. The individual per-term scores are aggregated to generate the final document score. A more sophisticated Probabilistic IR based model, Okapi BM25 [RW94, RW99] has become prevalent recently as it addresses many of the shortcomings of the simple tf-idf model. Okapi BM25 provides a smoothed non-linear influence of all ranking components in the final score. It normalizes term frequencies taking into account document lengths. It has been shown to be superior to the tf-idf model in
many retrieval tasks.

One form of time-aware querying that is very common is the search for news articles [CGR05]. The main challenge in searching news is to automatically incorporate publication times of news articles in order to generate “interesting” rankings. However, there is no notion of versioning for news article, i.e., each occurrence is considered as an explicit document instance, and the rankings are generated accordingly. In contrast, our work aims at allowing an explicit time-window of interest in queries, and providing document-level rankings for the specified time-window.

Inverted file indexes are an efficient way to index text and are deployed in numerous of today’s systems including major Web search-engines. For a recent comprehensive overview of inverted file indexes we refer to Zobel and Moffat [ZM06]. The main attractions of inverted lists include their excellent compressibility, spatial locality of reference, and adaptability to a wide variety of query interfaces. Maintenance of inverted file indexes against evolving text collections has been studied in the contexts of document additions/deletions, as well as, in-place updates to documents during incremental crawling [BCL06]. However, the goal of all these previous proposals is to keep the index up-to-date with the contents of the text collection – no effort is made in capturing the evolution of the collection in the index. Interestingly, there have been proposals such as [SGM97] that have addressed the issue of capturing the history of database evolution, but their solution is simply to maintain separate, periodically updated indexes for a time window in the past. In contrast, the techniques presented here aim at capturing the history of the collection in a single immortal index, while retaining most of the high-performance features of standard inverted file indexes.

There has been more than two decades of work on incorporating the temporal evolution of databases into index structures (for an excellent survey we refer to [ST99]). However, most of the techniques proposed in this context solve only the boolean aspect of the query, e.g., retrieve the tuple(s) that satisfy the given constraint. Querying versioned text collections as we consider them in this paper necessitates an entirely different querying model (keyword-based querying) and requires a ranking of results. In addition, while the retrieval granularity in the case of temporal databases is at the record-version level, here we focus on aggregated document-level result granularity in order to provide effective results.

Temporal coalescing, a technique that we adapt and extend in the scope of this work, was originally proposed by Böhlen et al. [BSS96] in the context of temporal databases.

3 Data and Query Model

We consider a general model of a dynamic document collection where each document refers to a collection of versions. The textual content is held in each version, and the inter-document references also point to individual versions rather than to the corresponding document.
Formally, a document $d$ is modeled as a sequence of timestamped versions $d^t_i$, i.e.,

$$d = \langle d^t_1, d^t_{t+1}, \ldots \rangle.$$  

The timestamps associated with versions represent either the creation times or observation times of the version, depending on the application. Each version $d^t_i$ can be considered as a vector of terms as in the vector-space model. Document versions are assumed valid until replaced by a newer version of the same document or deleted from the collection. The document deletions are modeled by replacing the document vector by a special zero vector, $\perp$, that denotes an empty and inactive document. A document version $d^t_i$ has the implicit validity time-interval $[t_i, t_{i+1})$ with $t_{i+1}$ being set to now for the current version. The set of timestamps of versions valid for a given time-period $[t_b, t_e]$ is denoted as $V(d, [t_b, t_e])$, i.e.,

$$V(d, [t_b, t_e]) = \{ t_i \mid d^t_i \in d \land d^t_i \neq \perp \land t_i \leq t_e \land t_{i+1} > t_b \}.$$  

We use the notation $d^t_t$ to refer to the version of $d$ valid at time $t$.

We formulate time-varying forms of document-specific and collection-specific statistics. Thus, let $tf(v, d^t)$ denote term frequency (i.e., the number of occurrences) of term $v$ in the document version $d^t$. The length of document version $d^t$ is referred to as $dl(d^t)$. As collection-specific statistical values we define $N(t)$ as the collection size at time $t$ and $avdl(t)$ as the average length of documents, both computed over only the versions that are valid at time $t$. Apart from that, let $df(v, t)$ denote the number of documents in the collection that contain term $v$ at time $t$.

Using the above model, it is possible to represent not only the scenario where documents are explicitly version-managed – such as in Wikis, but also when versions are inferred implicitly like in the case of collections obtained through repeated crawls of the Web. Note that in the latter case, it is possible to have missed a few changes to a document as a result of the crawl strategy, but we consider this to be an issue orthogonal to our setting. Further, the model naturally allows for deletion and subsequent reappearance of documents.

### 3.1 Query Model

A time-travel query $q^{[t_b, t_e]}$ consists of a content part $q$ and a temporal context $[t_b, t_e]$. The content of a query has the same structure as a version in our data model, and can be regarded as a set of keywords. For the evaluation of the query only versions that exist at any point in the time interval of interest $[t_b, t_e]$ are taken into account. If a time-travel query is specified without a temporal context we consider that to be equivalent to the case when the temporal context spans the complete life-time of the archive.

#### 3.1.1 Version-level Relevance Scoring

Next, we adapt two existing relevance models, namely Okapi BM25 and a tf-idf model, to quantify the relevance of a document version $d^t_i$ to a query $q^{[t_b, t_e]}$ as a score $w(q^{[t_b, t_e]}, d^t_i)$. Both models require slight adaption, since statistical values such as document frequencies
and the collection size $avdl$ are fixed in their original formulations and not time-varying as in our scenario.

**TF-IDF Scoring** The tf-idf model considered here combines term frequencies with logarithmically dampened inverse document frequencies. The inverse document frequency of a term $v$ at time $t$ is defined as

$$idf(v, t) = \log \frac{N(t)}{1 + df(v, t)}$$

and, as stated above, is time-varying in our setting. For a temporal context $[t_b, t_e]$ we therefore aggregate inverse document frequencies in the respective time-window and let $\overline{idf}(v, [t_b, t_e])$ denote the aggregated value. In this paper, we consider $\overline{idf}(v, [t_b, t_e])$ obtained by a simple averaging of $idf$ values in the time-window, although other forms of aggregation are also applicable. Altogether we yield the following formulation for the tf-idf model

$$w(q^{[t_b, t_e]}, d^{t*}) = \sum_{v \in q} tf(v, d^{t*}) \cdot \overline{idf}(v, [t_b, t_e])$$

(1)

**Okapi BM25** Okapi BM25 [RW99] employs a different definition of inverse document frequencies as

$$idf(v, t) = \log \frac{N(t) - df(v, t) + 0.5}{df(v, t) + 0.5}$$

and extends term frequencies taking into account document lengths. Again, we aggregate inverse document frequencies over the temporal context as $\overline{idf}(v, [t_b, t_e])$, and obtain

$$w(q^{[t_b, t_e]}, d^{t*}) = \sum_{v \in q} \frac{(k_1 + 1) \cdot tf(v, d^{t*})}{k_1 \cdot ((1 - b) + b \cdot \frac{dl(d^{t*})}{avdl(t_e)}) + tf(v, d^{t*})} \cdot \overline{idf}(v, [t_b, t_e])$$

(2)

as our adaptation of Okapi BM25. The parameters $k_1$ and $b$ are inherited from the original model and are commonly set to values 1.2 and 0.75 respectively.

### 3.2 Document-level Score Aggregation

With the model introduced so far we can assess the relevance of a document version to a given $q^{[t_b, t_e]}$. In order to obtain the document-level relevance, we need to combine the version-level relevance model at a coarser granularity. To this end, we aggregate relevance scores $w(q^{[t_b, t_e]}, d^{t*})$ of document versions $d^{t*}$ that existed at any point in the temporal context $[t_b, t_e]$ to obtain a document relevance score $w(q^{[t_b, t_e]}, d)$. We consider three such document relevance models, which differ in the way version relevance scores are aggregated. Figure 1 illustrates the rankings under these models.

**MIN:** This relevance model conservatively judges the relevance of a document based on the least relevant of its versions. Thus, for a document to score high under this
model, all of its versions must achieve high relevance. The document relevance is defined as
\[ w(q^{[t_b,t_e]}, d) = \min \{ w(q^{[t_b,t_e]}, d_t) | t \in V(d, [t_b,t_e]) \} . \]  
\[ (3) \]

**MAX:** In contrast, the MAX relevance model takes the most relevant version as an indicator of document relevance. Under this model, a document may score high if at least one of its versions achieves high relevance. Formally the document relevance is then defined as
\[ w(q^{[t_b,t_e]}, d) = \max \{ w(q^{[t_b,t_e]}, d_t) | t \in V(d, [t_b,t_e]) \} . \]  
\[ (4) \]

**TAVG:** Finally, the TAVG relevance model considers the relevance of all versions to assess document relevance. Obviously, a simple sum of version relevance scores would favor frequently changing documents. Instead, we use the mean version relevance in \([t_b,t_e]\), which takes into account the lifespan of each version and is defined as
\[
\begin{align*}
  w(q^{[t_b,t_e]}, d) &= \begin{cases} 
    \frac{1}{t_e-t_b} \int_{t_b}^{t_e} w(q^{[t,t_e]}, d_t) \, dt & \text{if } t_b \neq t_e \\
    w(q^{[t,t]}, d_t) & \text{if } t_b = t_e = t 
  \end{cases} 
\end{align*}
\]
\[ (5) \]

Since the function \( w(q^{[t_b,t_e]}, d_t) \) is piecewise-constant in time, the integral in the above equation can be efficiently computed as a weighted summation of these segments. Unlike the MIN and MAX models the TAVG model is more robust to outliers in version relevance scores, considering the entirety of version relevance scores. The MIN (MAX) relevance model, on the other hand, may assign low (high) relevance to a document if one of its versions achieves a (low) high relevance score.

We illustrate the effect of these aggregation models over three documents – A, B, and C, whose relevance scores vary with time as illustrated in Figure 1. Under the MIN model, we see that document B is ranked above the other two documents, since the least relevant version of B is better than the least relevant versions of either A or C – although the other
two documents have at least one version whose relevance score is more than any version of B. On the other hand, under the MAX model, document A comes on top of results, as its most relevant version is clearly better than that of the other two documents. Finally, the TAVG captures the notion of “consistency” of a document (in terms of its relevance score). Thus, the document C is given higher preference since almost all of its versions have consistently good relevance score – except for one single version that existed only during a very small time-window.

Later, in Section 5, we show that all three document relevance models can be efficiently implemented using the index structure I2T2 (Immortal Index for Time-Travel) which we propose in the next section.

4 Temporal Text Indexing

In this section we introduce the design of our temporal text-indexing system I2T2. We first describe our temporal extensions to the inverted file index and, following that, present ways to reduce index size using temporal coalescing.

4.1 Index Design

In order to provide the querying functionality outlined in the previous section, we need to efficiently compute the version-relevance scores \( w(q_{[t_s,t_e]}, d_t) \) and their document-level aggregations.

Both the tf-idf model and the Okapi BM25 model for version relevance considered above are summations of products of a term-document-specific measure and a term-collection-specific measure (i.e., the first and second factor within the summation of Equation 1 and 2 respectively). In our design, the term-document-specific tf-component and term-collection-specific idf-component are maintained separately, which is in contrast to common practice for static document collections where only their product is kept. In our setup, however, this separation adds the flexibility of computing the idf-component depending on the temporal context of the query. Thus the I2T2 system consists of the following three components:

1. **Versions**: the sequence of version timestamps per document
2. **IDF**: the time series of idf-scores per term
3. **TF**: the time series of tf-scores per occurring term per document.

The version index which lists the version timestamps of every document is employed for the computation of validity time-interval for each version of a document. As mentioned earlier, a version is considered to be valid until another version is detected, or, the document has disappeared from the collection. This information is organized as a B-Tree.
As individual documents in the collection undergo changes over time, the overall collection-level term statistics also vary as an aggregated effect. According to the relevance models presented earlier in Section 3.1, given a temporal context \([t_b, t_e]\) we need to efficiently compute the corresponding aggregated idf-score of the term. We maintain for each term the sequence of its idf-scores along with the associated validity time-interval. These idf-scores are computed collection-wide for all terms at regular intervals by taking into account only the active documents. At query time we scan this idf-score sequence for all query terms and retrieve the entries that are valid for the specified query time-window.

Most crucial for the efficiency of query processing, however, is the way per-term per-document information is kept, since, as we detail below, accessing this pool contributes most of the query-processing cost.

The per-term per-document information is typically maintained in an inverted file index. In brief, an inverted file index consists of a vocabulary that maps each term to its inverted list. The vocabulary is commonly organized as a B-Tree to efficiently lookup the inverted list for a given term. The inverted list for a term is a list of postings that are tuples containing a document identifier and additional payload like the frequency of the term in the document. Depending on the query task, these lists are sorted differently – e.g., in document-identifier, or term-frequency order. For a detailed recent survey about inverted file indexes we refer to Zobel and Moffat [ZM06].

Our extension to the inverted file index affects: (i) the structure of postings and (ii) the order of inverted lists. We build our extended inverted file index over all versions of documents, and record the information about the life of versions explicitly in the index. As a result, each posting now includes the validity time-interval of the corresponding document version. In other words, the posting

\[(d, tf, t_i, t_j)\]

in the inverted list belonging to term \(v\) expresses that for term \(v\), document \(d\) had score \(tf\) during the time interval \([t_i, t_j]\). We thus obtain one such posting \((d, tf, t_i, t_i+1)\) for each term \(v\) occurring in a document version \(d^{t_i}\).

Furthermore, for reasons that become clear in the following sections, we keep inverted lists sorted with descending score as the primary sort order, ascending document identifiers as the secondary sort order, and increasing left boundaries of validity time-intervals as tertiary sort order.

4.2 Index-size Reduction through Temporal Coalescing

In our extended inverted file index described above, it is easy to observe that a lot of space is wasted by having one posting per term that occurs in a document version. In the evolution of a typical document over time, the changes are not very drastic, thus the associated tf-values of most terms is unchanged between two adjacent versions. We now introduce temporal coalescing as a means to exploit this feature of document evolution and to counter the waste of space. The idea behind temporal coalescing is to coalesce
temporally adjacent postings belonging to the same document that have identical payloads. We can further reduce the index size by performing approximate temporal coalescing – where two postings are coalesced if their payloads are similar to each other. In the rest of this section, we describe these index reduction strategies.

**Accurate Temporal Coalescing**

**Accurate temporal coalescing** coalesces temporally adjacent postings only if their payloads, i.e. tf values, are identical. Thus, two postings, \((d, t_{f0}, t_0, t_1)\) and \((d, t_{f1}, t_1, t_2)\), are coalesced into \((d, t_{f0}, t_0, t_2)\) only if \(t_{f0} = t_{f1}\). We can implement accurate temporal coalescing efficiently as a single pass algorithm presented in Algorithm 1, by exploiting the sort-order of our index lists.

**Algorithm 1** Accurate Temporal Coalescing

1: \(input = \{(d_i, t_{fi}, t_{bi}, t_{ei})\}\)
2: \(output = \emptyset\)
3: \(b = null\)
4: for \((d_i, t_{fi}, t_{bi}, t_{ei}) \in input\) do
5:  if \(b \neq null \land (b.d = d_i \land b.t_e = t_{ei} \land b.t_f = t_{fi})\) then
6:    \(b.t_e = t_{ei}\)
7:  else
8:    \(b = (d_i, t_{fi}, t_{bi}, t_{ei})\)
9:    \(output.append(b)\)
10: end if
11: end for

Algorithm 1 scans the non-coalesced input inverted list in sorted order. The result of the current round of coalescing is maintained in a posting buffer \(lt\). For each posting read from the inverted list, the algorithm either coalesces the posting with \(lt\), or reinitializes \(lt\) by copying the contents of the posting just read, after appending \(lt\) to the output inverted list.

Accurate temporal coalescing reduces the index size without affecting the resulting rankings. However, the scale of this space reduction clearly depends on the likelihood of having identical payloads in temporally adjacent postings. For the tf-idf relevance model (Equation 1), where the payloads are simply the term frequencies, this likelihood is high as typical changes to a document leave large portions of the document unchanged. In contrast, under the Okapi BM25 relevance model, as can be seen from Equation 2, payloads of temporally adjacent postings are less likely to be identical. There are two reasons for this: (i) a single change in the document that affects the document length, e.g., adding a few words, is likely to make the payloads of all new postings different from those of their temporally preceding postings, and (ii) the collection-level average document length measure could change even when there are no changes to the document, also affecting the payloads of postings.
Approximate Temporal Coalescing

In order to overcome the limitations of accurate temporal coalescing, we introduce approximate temporal coalescing. This is based on our hypothesis that temporally adjacent postings can be coalesced even if their payloads differ to a small degree without inducing much error in the final query result. Thus, temporally adjacent postings are coalesced even if their payloads differ, as long as the error made through the use of coalesced posting is within a tunable bound.

Formally, for an input sequence of temporally adjacent postings $I$ belonging to document $d$ we aim to find a shorter sequence $O$ of coalesced postings so that

$$
\forall (d, t_f, t_i, t_{i+1}) \in I : \forall (d, t_f, t_j, t_{j+1}) \in O :
\quad t_j \leq t_i \land t_{i+1} \leq t_{j+1} \Rightarrow \text{error}(t_f, t_f) \leq \epsilon .
$$

In the formula above, error is a measure of the approximation error made by using the payload $t_f'$ of the coalesced posting, and $\epsilon$ is a tunable threshold value the error must not exceed. For error, we use the relative error measure, defined as

$$
\text{error}_{rel}(t_f', t_f) = \frac{|t_f' - t_f|}{\max\{c, |t_f|\}} ,
$$

where $c$ is a sanity constant, which we fix at $10^{-4}$.

The problem above can be mapped to finding a piecewise-constant representation for the $N$ points $(t_i, t_f)$ retaining our guarantee on the error. Finding piecewise-constant representations is a well-studied problem in time-series segmentation [KCHP01, TT06] and histogram construction [GSW04, IP95, JKM+98]. Using dynamic programming an optimal solution having a minimal number of $B^*$ postings (segments, buckets) can be found in time $O(N^2B^*)$ [JKM+98]. The greedy approximate Algorithm 2 has $O(N)$ time complexity and performs well in practice. It produces solutions that retain the guarantee on the error and have a close-to-optimal number of postings.

We note that the algorithm, in contrast to Algorithm 1, takes as an input a document-specific sequence of temporally adjacent postings sorted in ascending order of the left validity time-interval-boundary. For a sequence of temporally adjacent postings the auxiliary function coalesce produces a single posting minimizing the maximum error. For the relative error measure that we use, this optimal value can be looked up as described in Guha et al. [GSW04], thus computable in constant time. In each step, Algorithm 2 attempts to add the newly read posting to the buffer $b$, which holds a sequence of postings that can be coalesced while retaining the error guarantee. If the attempt fails, i.e., the coalesced posting produced by coalesce does not retain the error guarantee, the coalesced posting corresponding to the contents of the buffer $b$ is added to the output and the buffer $b$ is reinitialized.
Algorithm 2 Approximate Temporal Coalescing

1: \textit{input} = \{(d, tf_i, t_{bi}, t_{ei})\}
2: \textit{output} = \emptyset
3: \textbf{for} \ (d, tf_i, t_{bi}, t_{ei}) \in I \ \textbf{do}
4: \quad \textbf{if} \ b = \emptyset \ \textbf{then}
5: \quad \quad b.\text{append}((d, tf_i, t_{bi}, t_{ei}))
6: \quad \textbf{else}
7: \quad \quad lt = \text{coalesce}(b \cup (d, tf_i, t_{bi}, t_{ei}))
8: \quad \quad \textbf{if} \ \exists (d, tf_j, t_{bj}, t_{ej}) \in (b \cup (d, tf_i, t_{bi}, t_{ei})): \text{error}(tf_j, lt.tf) > \epsilon \ \textbf{then}
9: \quad \quad \quad \textit{output}.\text{append}((\text{coalesce}(l)))
10: \quad \quad \textbf{else}
11: \quad \quad \quad b.\text{append}((d, tf_i, t_{bi}, t_{ei}))
12: \quad \textbf{end if}
13: \textbf{end if}
14: \textbf{end for}
15: \textit{output}.\text{append}((\text{coalesce}(b)))

5 Query Processing for Ranked Retrieval

Having introduced our index design and temporal coalescing as a way to reduce index size, we now discuss how queries $q^{[t_b,t_e]}$ can be efficiently evaluated under the different relevance models introduced in Section 3. Since we are interested in the ranked retrieval of top results for a query, we adapt the popular No Random Accesses (NRA) algorithm from the family of threshold algorithms [FLN03] for query processing. We coin our extended algorithm TC-NRA (Temporal Context aware NRA).

5.1 Preliminaries: NRA Algorithm

Before we describe our TC-NRA algorithm, we briefly recall some details about the original NRA. As the name indicates, NRA determines the top-$k$ result using only cheap sorted accesses to the inverted lists in a round-robin fashion, and avoids expensive random accesses. At each step of the algorithm, the candidate documents are kept in a main-memory pool. For each document seen in at least one of the lists, the set of lists on which the document score has been evaluated is maintained. Based on the scores seen so far in the inverted lists, for each candidate a worstscore and a bestscore value is maintained. These reflect its worst and best possible scores, respectively, including the lists in which the document is yet unseen. The minimal worstscore in the current top-$k$, referred to as $min_k$, acts as a threshold for stopping the index scans. The algorithm terminates when none of the candidates has a bestscore value exceeding $min_k$. When this is the case, none of the remaining candidates can anymore make it into the top-$k$. The algorithm is guaran-
ted to produce the correct top-$k$ result if $\text{worstscore~(bestscores)}$ increase (decrease) monotonically.

### 5.2 TC-NRA Algorithm

The NRA algorithm described above is clearly oblivious to the notion of document versions (and their corresponding validity time-intervals), as well as, to the time-window of interest expressed in the query. In order to adapt the NRA algorithm to our setting, we extend it in the following two key aspects: (i) the bookkeeping associated with candidates during the run of the algorithm, and (ii) the definitions of $\text{worstscore}$ and $\text{bestscore}$ of a candidate. The resulting algorithm, called TC-NRA, is presented in Algorithm 3.

Recall that the required result granularity for time-travel queries is at the level of documents as collections of versions. Referring to equations 3, 4, and 5, we see that the relevance score of a document in the query temporal-context $[t_b, t_e]$ depends on the scores of its versions that are valid during this period. Thus, unlike in the case of original NRA where keeping one $\text{worstscore}$ and $\text{bestscore}$ per candidate is sufficient, we now have to maintain one $\text{worstscore}$ and $\text{bestscore}$ for each document version that existed in the temporal context $[t_b, t_e]$. We maintain these as two time series, denoted as $\text{worstscores}$ and $\text{bestscores}$. Next, the bookkeeping of evaluated terms per candidate is extended – for a dimension (term) $v$ the time intervals for which the dimension has been evaluated already are kept track of in $\text{evaluated}[v]$. In the pseudocode of Algorithm 3 we make use of some notational shortcuts. When advancing the cursor on the current list (line 10), the method $\text{next}([t_b, t_e])$ advances the cursor to the next posting $(d, s, t_k, t_l)$ that exists during the temporal context, i.e., fulfills $t_k < t_e \land t_l \geq t_b$. Further, we use abbreviated notation for updating the $\text{worstscores}$ (line 15) and $\text{bestscores}$ (line 19), thus when writing $\text{worstscores}[T] = \text{worstscores}[T] + s$ the value $s$ is added to $\text{worstscores}[t]$ for all $t \in T$.

We move on to the definition of $\text{worstscore}$ (line 15) and $\text{bestscore}$ (line 34) of a document, as an aggregation of the respective values of its constituent versions. Clearly, the model employed for document-level aggregation also affects the definitions of these values. Accordingly, we present appropriate definitions that satisfy the monotonicity requirements of NRA, under each of the three document-level score-aggregation models we have considered.

**MIN** For the MIN relevance model, $\text{worstscore}$ and $\text{bestscore}$ are defined as

\[
\text{worstscore}(d) = \min \{ d.\text{worstscores}[t] \mid t \in [t_b, t_e] \}
\]
\[
\text{bestscore}(d) = \min \{ d.\text{bestscores}[t] \mid t \in [t_b, t_e] \}.
\]
Algorithm 3 Time Context aware No Random Accesses (TC-NRA) Algorithm

1: \( \text{min}_k = 0; \ candidates = \emptyset; \ topk = \emptyset \)
2: 
3: /* fetch index lists for terms \( v \) in query \( q \) */
4: \( L = \{ v \mid v \in q \} \)
5: 
6: for \( v \in L \) do
7: \( (d, s, t_k, t_l) = v.\text{next}( [t_b, t_e] ) \)
8: \( \text{high}_v = s \)
9: 
10: /* do bookkeeping */
11: \( T = [t_k, t_l) \setminus \text{d.evaluated}[v] \)
12: \( \text{d.worstscores}[T] = \text{d.worstscores}[T] + s \)
13: \( \text{d.evaluated}[v] = \text{d.evaluated}[v] \cup T \)
14: for \( w \in L \) do
15: \( T = [t_k, t_l) \setminus \text{d.evaluated}[w] \)
16: \( \text{d.bestscores}[T] = \text{d.worstscores}[T] + \text{high}_w \)
17: end for
18: 
19: /* manage candidates and topk */
20: if \( \text{worstscore}(d) > \text{min}_k \) then
21: \( \text{topk}.\text{removeMinimum}() \)
22: \( \text{topk}.\text{insert}(d) \)
23: \( \text{candidates}.\text{remove}(d) \)
24: else if \( \text{bestscore}(d) > \text{min}_k \) then
25: \( \text{candidates}.\text{insert}(d) \)
26: else
27: \( \text{candidates}.\text{remove}(d) \)
28: end if
29: \( \text{min}_k = \min\{ \text{worstscores}(d) \mid d \in \text{topk} \} \)
30: 
31: /* check stopping criterion */
32: if \( \text{candidates} = \emptyset \lor \max\{ \text{bestscore}(d') \mid d' \in \text{candidates} \} \leq \text{min}_k \) then
33: return topk
34: end if
35: end for
MAX For the MAX relevance model we define $worstscore$ and $bestscore$ as

$$worstscore(d) = \max \{ d.worstscores[t] \mid t \in [t_b, t_e] \}$$

$$bestscore(d) = \max \{ d.bestscores[t] \mid t \in [t_b, t_e] \} .$$

TAVG For the TAVG relevance model we define $worstscore$ and $bestscore$ as

$$worstscore(d) = \begin{cases} \frac{1}{t_e - t_b} \int_{t_b}^{t_e} d.worstscores[t] \, dt & \text{if } t_b \neq t_e \\ d.worstscores[t] & \text{if } t_b = t_e = t \end{cases}$$

$$bestscore(d) = \begin{cases} \frac{1}{t_e - t_b} \int_{t_b}^{t_e} d.bestscores[t] \, dt & \text{if } t_b \neq t_e \\ d.bestscores[t] & \text{if } t_b = t_e = t \end{cases} .$$

Monotonicity of $worstscore$ and $bestscore$ in all three cases follows directly from the monotonicity of each individual $worstscore[t] (bestscore[t])$ that can only increase (decrease) in the course of Algorithm 3.

6 Experiments

The presented techniques were implemented in a prototype system using Java JDK 1.5. All experiments were run on a SUN V40z machine having four AMD Opteron CPUs, 16 GB RAM, and a large network-attached RAID-5 disk array. All data is kept in an Oracle 10g database that runs on the same machine.

6.1 Setup & Datasets

As a dataset for our experiments we used the English Wikipedia revision history that is available for download as one large XML file. This dataset contains the editing history of the English Wikipedia spanning the time-window from January 2001 to December 2005 (the time of our download). This rich dataset contains much more information than the mere encyclopedia articles, among them, for instance, discussions associated with articles. We refer to Buriol et al. [BDLM05] for a detailed discussion of the dataset and its properties. For our experiments, we randomly picked 20% of versions, excluding those versions marked as minor revisions (e.g., if only typos are corrected). This yielded a total of 892,255 documents and 2,795,383 versions, with average of 3.13 versions per document at sample standard deviation 14.24. We computed idf-scores for each term on this dataset at monthly intervals and maintained it as the idf time-series component of I2T2.

We extracted a set of 45 queries from a large query log that was recently made available by AOL research, however, is no longer publicly available for download. The selected queries are the most frequent queries among those that led to a result click on a Wikipedia article. For each query we generated six temporal contexts. Three temporal contexts were chosen
as the years 2003, 2004, and 2005. Further, for each query and each year we randomly picked one temporal context that coincides with one quarter, thus giving us another three temporal contexts per query.

6.2 Index Sizes

We computed indices both for the tf-idf and the Okapi BM25 model using approximate coalescing varying the threshold $\epsilon$ as 0%, 1%, 5%, 10%, yielding a total of eight different indices. Note that $\epsilon = 0\%$ corresponds to accurate coalescing and is considered as the baseline in our experiments. We computed the top-100 for each keyword query, each temporal context, each index, and each document-level score-aggregation model resulting in a total of 6,480 queries that were processed. The resulting index sizes (measured as the total number of postings) are given in Table 1. The non-coalesced indexes for tf-idf and Okapi BM25, which we computed for comparison, have a total number of 1, 244, 168, 879 postings.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>tf-idf Number of Postings</th>
<th>Ratio</th>
<th>Okapi BM25 Number of Postings</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>1,129,472,027</td>
<td>1.0000</td>
<td>1,186,996,012</td>
<td>1.0000</td>
</tr>
<tr>
<td>1%</td>
<td>180,196,069</td>
<td>0.1595</td>
<td>595,180,256</td>
<td>0.5014</td>
</tr>
<tr>
<td>5%</td>
<td>178,581,509</td>
<td>0.1581</td>
<td>327,687,360</td>
<td>0.2761</td>
</tr>
<tr>
<td>10%</td>
<td>177,003,930</td>
<td>0.1567</td>
<td>245,488,770</td>
<td>0.2068</td>
</tr>
</tbody>
</table>

Table 1: Index Sizes

The figures show that even for a small threshold $\epsilon = 1\%$ the index under the tf-idf model is reduced to about 16% of its original size. Similarly, for the Okapi BM25 model index size steadily decreases as $\epsilon$ is increased. For $\epsilon = 10\%$, as an example, the size of the index resulting from approximate coalescing is about 21% the size of the original index.

6.3 Accuracy

In the first experiment, we analyze how much the top-100 changes as we increase the threshold $\epsilon$. To this end, for every query, we compare the top-100 obtained for a particular value of $\epsilon$ against the top-100 produced by the baseline. Two result lists are compared using the following two measures: The overlap is the fraction of documents that occurs in both result lists and does not take into account the order of documents. Kendall’s $\tau$ is computed for the documents that appear in both results lists and reflects the agreement of the two lists on the order of documents. Here, a value of 1 ($-1$) indicates perfect agreement (disagreement) in order. Figure 2(a) and Figure 2(b) give mean overlap and mean Kendall’s $\tau$ values over all queries and temporal contexts for tf-idf and Okapi BM25 relevance models respectively.
We consistently observe that overlap and Kendall’s τ decrease gracefully as we increase the threshold ϵ for both the tf-idf and the Okapi BM25 relevance model. Further, we observe larger overlap and Kendall’s τ values for the tf-idf relevance model. Combining these results with the index sizes reported in Table 1, we see that for ϵ = 10% and the tf-idf relevance model, for instance, on an index that is only 16% the size of the original index, we yield a top-100 list containing on average 75 documents that are also present in the original result list.

6.4 Performance

Next, we analyze the performance of our approach for different values of the threshold ϵ and across the different document-level score-aggregation models that were introduced in Section 3. For both tf-idf and Okapi BM25 in combination with yearly and quarterly temporal contexts, we run our set of queries as one batch for MAX, MIN, and TAVG. The total number of sorted accesses that is needed to process this batch of queries is used as a measure of performance. The resulting figures for tf-idf and Okapi BM25 models are given in Figure 3(a) and Figure 3(b).

Figure 2: Accuracy@top-100

Figure 3: Performance@top-100
A first observation that can be made is that the MIN model is clearly the most expensive requiring up to one order of magnitude more sorted accesses. The MAX and TAVG models, in contrast, require about the same significantly lower numbers of sorted accesses. For all curves we observe that, as we increase the threshold $\epsilon$ from 0% to 10%, the number of required sorted accesses reduces by nearly an order of magnitude. While we observe a steady decrease in the number of sorted accesses for Okapi BM25, the respective figures for tf-idf exhibit a sharp drop for $\epsilon = 1\%$ that is followed by almost no further decrease. We note that this is in accordance with the index sizes reported in Table 1 that exhibit analogous behavior.

7 Conclusions

In this paper, we have explored in detail the issues in supporting a novel breed of temporally contextualized queries, called time-travel queries, over versioned document collections. We introduced a model for versioned document collections, and developed three relevance scoring models that aggregate version-level scores at a document-level granularity. In order to efficiently evaluate time-travel queries under the proposed relevance models, we designed an immortal indexing system called I2T2 and an efficient top-$k$ processing algorithm called TC-NRA. Our experiments over the English Wikipedia revision history showed that our indexing strategy can be tuned to reduce the index size by close to an order of magnitude, while retaining Kendall’s $\tau$ value close to 0.8 and about 75% content overlap in the top-100. We believe that these preliminary results show significant promise for supporting time-travel querying functionality and also opens avenues for a number of directions of future research. We plan to continue these studies over a wide variety of datasets including Web collections such as WebBase and Internet Archive. We are also investigating a variety of optimizations for the I2T2 system both in terms of its space-consumption and efficient index management. Another interesting direction we plan to explore is to automatically relax the temporal context in time-travel queries, thus treating it rather as a preference parameter of the user than as a hard condition for filtering results. Our final goal is to integrate these features into a popular, open-source text indexing system for web archives such as NutchWAX [Nut].

References


Abstract: Skyline queries have recently received a lot of attention due to their intuitive query formulation: users can state preferences with respect to several attributes. Unlike numerical preferences, preferences over discrete value domains do not show an inherent total order, but have to rely on partial orders as stated by the user. In such orders typically many object values are incomparable, increasing the size of skyline sets significantly, and making their computation expensive. In this paper we explore how to enable interactive tasks like query refinement or relevance feedback by providing ‘prime cuts’. Prime cuts are interesting subsets of the full Pareto skyline, which give users a good overview over the skyline. They have to be small, efficient to compute, suitable for higher numbers of query predicates, and representative. The key to improved performance and reduced result set sizes is the relaxation of Pareto semantics to the concept of weak Pareto dominance. We argue that this relaxation yields intuitive results and show how it opens up the use of efficient and scalable query processing algorithms. Assessing the practical impact, our experiments show that our approach leads to lean result set sizes and outperforms Pareto skyline computations by up to two orders of magnitude.

1. Introduction

Due to the ever growing volume of database content and the personalization needs in information searches, human preferences already play an essential part in today’s information systems. This is because mere SQL-style queries only too often produce empty or too numerous results. First approaches at cooperative databases as those by [LL87, Mo88], handled user queries that retrieved empty results with respect to a database instance by automatic relaxation of query predicates. Using score values to express the utility of database objects with respect to a query, cooperative queries come in various flavors:

- **Top-k queries** (see e.g. [GBK00, FLN01]) have shifted retrieval models from exact matching of attribute values to the notion of best matching database objects. Top-k models rely on basic scorings of objects for each query predicate and a utility function to aggregate the objects’ total scores.
Skyline queries extend this principle to cases where still score-based preferences exist for each query predicate, but no utility function is a-priori known to compromise between predicates (see e.g. [BKS01,TEO01,PTF03,BGZ04]). Skyline approaches adopt the principle of Pareto optimality, i.e. only those objects are returned, where no object exists in the database having better or equal predicate values.

Multi-objective retrieval [BG04] finally allows for the interleaved evaluation of arbitrary compositions of skyline and top-k queries with proven instance-optimal complexity.

Especially the skyline paradigm has proven its usefulness in a variety of applications (e.g., digital item adaptation [KB06] or location-based services [HJ04]), since users generally cannot be expected to provide sensible weightings for a utility function. But while score-based approaches generally allow for efficient query evaluation, their expressiveness in terms of human user preferences remains rather limited, cf. [Fi99]. With the use of preferences modelled as strict partial orders with intuitive “I like A better than B” semantics ([Ch02, Ki02]), this lack of expressiveness was remedied at the price of more expensive query evaluation. A first evaluation algorithm of such partial order preference queries was given only recently by [CET05]. Also here the Pareto principle was used for evaluating queries involving several partial order preferences:

- In [Ki02] and [CET05] a strong Pareto dominance principle called Pareto accumulation is used: an object has to be better or identical in all attribute values for the query predicates, and strictly better in at least one to dominate another object.
- In contrast [Ch03] and [BG05] propose a weak Pareto dominance principle called Pareto composition, where an object’s attribute values has to be better, identical or incomparable in all predicates, and strictly better in at least one to dominate another object.

The Pareto principle extends querying capabilities and the result set contains all possible best database objects with respect to arbitrary utility functions. On the other hand Pareto sets grow exponentially in size with increasing numbers of preferences [Be78]. Thus, typical tasks during the query process (like query refinement) rather need a good (and efficiently computed) overview over skylines.

For instance, [KRR02] presents an online algorithm where users can influence the order in which skyline objects are produced. Eventually the entire skyline is calculated, but at every stage of the computation users can provide a direction where most relevant objects might be expected. The work in [BZG05] also relies on user interaction, by presenting the user with a representative sample of the expected skyline set, and then exploiting user feedback to elicit an appropriate utility function for the final result ranking. [KP05] proposes to cover the skyline set with $\varepsilon$-spheres where each center of a sphere is a representative for all skyline objects within a distance of at most $\varepsilon$. This set of representatives is subsequently returned to the user. However, the computation of an $\varepsilon$-sphere cover was shown to be NP-hard for more than 2 independent predicates. Moreover, the calculation of such approximations always needs expensive computations of the entire skyline. These approaches only focus on total-order preferences: all objects
can be compared in each predicate, which makes combinations of different predicates simple. Due to the indifference property in partial order preferences the Pareto combination leads to even bigger result sets: if an object is incomparable to other objects with respect to just a single preference, it still is Pareto-optimal and thus part of the skyline, even if it is the least preferred object with respect to all other preferences. In practical applications such incomparability often occurs: users can be indifferent between items and very rarely model preference relations between all possible attribute values for a query predicate anyway.

Recent research in [Ki05] has started to combat such indifference in partial order preferences by means of ‘substitute values’. The substitute values (SV) semantics assigns equal usefulness to some incomparable values. Still, this semantics only remedies a small number of cases and is comparable in size and evaluation time to the complete skyline.

Our goal is somewhat more ambitious. We want to efficiently provide ‘prime cuts’ of the skyline that can be used in an interactive query process. These prime cuts have to be both manageable in size and representative of the Pareto skyline. In this paper we present an innovative algorithm for the efficient computation of such prime cuts which relies on weak Pareto dominance, as defined in [BG05]. Weak Pareto dominance changes the preference semantics to an even higher degree than SV semantics. The resulting ‘restricted skyline’, i.e., all objects not weakly Pareto dominated, contains intuitively appealing objects, can be derived surprisingly efficient, and thus will deliver our prime cuts. The contribution of our approach is therefore twofold:

- Restricted skylines derive manageable subsets of the partial order skylines (useful e.g. as a preview, or for query refinement) by taming the effects of incomparability. Our evaluation shows that sizes of restricted skylines are usually lean.
- Our approach allows to efficiently approximate these restricted skylines without having to compute the entire Pareto set first. Query processing relies on progressive iteration of ranked result lists for each predicate and allows for pruning.

In the following we will give a motivating scenario for partial order skylines and explain the semantics of the restricted skyline set. We will present the efficient evaluation algorithm for restricted skylines and perform extensive experiments to prove the practical applicability.

2. Weak Pareto Dominance and Restricted Skylines

The following example will illustrate Pareto skylines and lead to the basic notion of weak Pareto dominance.

Example 1: Given preferences P1 on car types and P2 on colors in Figure 1 and the following database instance: a green roadster, a black coupé, a blue SUV, a yellow truck and a pink limousine. None of them are dominated. The green roadster is maximal in P1. It does not dominate the black coupé, because black color is preferred over green. Furthermore, the user is indifferent between black and blue cars, thus the blue SUV is
not dominated by the black coupe, nor dominated by the green roadster because of $P_2$. Though the yellow truck has the worst car type, the user has not given any judgment on its color, thus making it incomparable. Finally, the pink limousine is completely incomparable to all other objects. Thus, the Pareto skyline contains all five elements.

Using the normal definition of Pareto sets, in Example 1 the entire database would have to be retrieved and returned to the user. Since a user usually is interested in refining queries according to the most promising result objects, retrieving a sophisticated selection from the skyline is a far more cooperative behavior. Our restricted skyline is such a selection. But on what grounds can we select ‘better’ objects from the full Pareto skyline?

Generally speaking, skyline queries are only sensible if no ordering or weightings between individual predicates are provided. Otherwise utility-based ranking schemes such as top-$k$ queries would be far more efficient to use. Pareto sets are designed to consist of all optimal objects with respect to all possible utility functions. Therefore, selecting a subset of the skyline will always ignore objects that are nevertheless optimal for some utility function. In other words, any selection will consider some utility functions as being more probable than others. Such an assessment has to be based on heuristics. We rely on the heuristic that all user preferences should be relaxed evenly and as little as possible, i.e. the relaxation scheme should be fair. In any case, a selection doesn’t have to be the final result set. If it can be computed reasonably fast and yields manageable result sets, it can also be used as a good starting point for focused searches such as the online algorithm in [KRR02] or the feedback algorithm in [BZG05]. Since our selection relies on weak Pareto dominance, we will formalize its semantics in the following definition (cf. Pareto composition in [Ch03]):

**Definition 1:** (weak Pareto dominance)
Let $O$ be a set of database objects and $x, y \in O$. An object $x$ is said to weakly dominate object $y$ with respect to partial order preferences $P_1, \ldots, P_n$, if and only if there is an index $i$ ($1 \leq i \leq n$) such that $x$ dominates $y$ with respect to $P_i$ and there is no index $j$ ($1 \leq j \leq n$) such that $y$ dominates $x$ with respect to $P_j$. That means, with $>_P$ denoting the domination with respect to partial order $P$:

$$x \text{ weakly dominates } y \iff \exists i (1 \leq i \leq n): x >_P y \land \neg \exists j (1 \leq j \leq n): y >_P x$$

We call the set of all non-weakly-dominated objects the ‘restricted’ skyline. Please note that for total order preferences, weak and strong Pareto dominance coincide, because
there are no incomparable objects. Let us reconsider our example and see what changes, if we restrict the skyline set using weak Pareto dominance.

Example 1 (cont.): Consider the objects from above under the notion of weak Pareto dominance (Figure 2). There is still no weak dominance relation between the green roadster, and the black coupé, because black color is preferred to green, but a roadster is deemed better than a coupé. However, both of them now weakly dominate the yellow truck and it can be removed in the restricted skyline. Removing the yellow truck seems indeed a very intuitive thing to do, because $P_1$ tells us that everything is better than a truck and the user, although voicing explicit color preferences, did not express his/her opinions on yellow cars. Moreover, we have to take a closer look at the relation between the black coupé and the blue SUV. The user is indifferent between both colors. But the black coupé fits his/her car type wishes to a higher degree, hence is probably more desirable. The weak dominance relation reflects this semantics: the blue SUV is weakly dominated by the black coupé and can be removed. Please note that the pink limousine with incomparable predicate values only is still not dominated by anything and will thus also be part of the restricted skyline. This reflects the notion that an item may be desirable, even if a user was not aware of it when formulating the query.

In the end, the result size in our small example is almost halved and only less intuitive candidates have been pruned. Our work in [BG05] shows that restricted skylines are a proper subset of the normal skyline, i.e. the strong Pareto set. The same applies to the substitute values skyline, as shown in [Ki05]. Finally, it can be shown that the restricted skyline is always a subset of the SV-skyline.

3. Efficiently Computing Restricted Skylines

Unlike numerical skylines, any partial order algorithm needs to handle object incomparability. This makes algorithms on total orders (such as NN [KRR02] and BBS [PTF03]) unsuitable. In contrast we rely on a scheme using topologically ordered lists: for each query predicate a list of all database objects sorted according to the respective user preference is created. Incomparability can be resolved by exploiting the level order of the preference. The algorithm’s main challenge is to determine, if all relevant objects have already been seen. In each query evaluation our algorithm therefore first computes possible value combinations (so-called $l$-cuts), which guarantee safe pruning: if a set of
objects instantiate any \( l \)-cut no relevant object can exist in the tails (higher than level \( l \)) of the sorted lists. The creation of the sorted lists and the calculation of the pruning thresholds are only dependent on preference size, not on database size, and therefore fast to compute.

3.1 Level Order for Partial Order Preferences

For pruning, we have to arrange for sorted access to objects for each query predicate: possibly relevant objects should be returned earlier than rather irrelevant objects. To create a proper sorting from the given partial preference orders, we use a simple breadth first topological ordering defining ‘levels’:

**Definition 2:** (level order)
Let \( P \) be a partial order preference. A value \( v \) is said to belong to level \( l \) or \( \text{level}(v) = l \) with respect to \( P \), if and only if the longest path from any maximum attribute value in \( P \) to \( v \) consists of \((l - 1)\) edges. Values not explicitly expressed in \( P \) belong to level 1. We denote the set of all values in level \( l \) as \( \text{level}(l) := \{ v \mid \text{level}(v) = l \} \).

Analogously, a database object \( x \) is said to be in level \( l \) with respect to \( P \), if its attribute value is in level \( l \).

This notion of levels imposes an intuitive sorting: all maximum (i.e. non-dominated) objects of \( P \) are on level 1, all objects that are only dominated in \( P \) by maximum objects are on level 2, and so on. We call this order level order. In the special case of numerical or total order preferences the level corresponds to each object’s rank, if objects with identical scores/attribute values are considered to have equal rank. But for partial orders this level order has another nice property:

**Lemma 1:** (level order domination)
Let \( O \) be a set of database objects and \( x, y \in O \). Then object \( x \) can only dominate object \( y \) with respect to a partial order preference \( P \), if \( \text{level}(x) < \text{level}(y) \) with respect to \( P \).

**Proof:** If \( x \) dominates \( y \) there is a path of length \( q > 0 \) from \( x \) to \( y \) in \( P \). Thus it directly follows from the definition of levels by longest paths in Definition 2, that:

\[
\text{level}(x) < \text{level}(x) + q \leq \text{level}(y).
\]

---

**Figure 3. Level order example**
Though objects can only be dominated by objects in smaller levels, due to the partial order semantics they do not have to be dominated by all objects in these levels, but can also be incomparable. For example, blue cars are in a smaller level than grey cars for our preference P2, although both are incomparable (see Figure 3). In the following we will assume all database objects to be accessed in level order for each preference.

Note that it is not necessary to compute a complete object index based on the level order for each incoming query. Instead, the database maintains object sets clustered by value, i.e., the sets of objects sharing the same value for a predicate. Then, creating a list in level order just means to sort references to these sets, not to sort all database objects. Since user preferences are typically rather small, producing level orders is fast even for large databases.

3.2 Identifying the Pruning Thresholds

In the last section we have defined a sorted list of objects for each predicate. For pruning we introduce the concept of l-cuts. While iterating over the lists, we have to check whether all relevant (i.e. not weakly dominated) objects have been accessed already.

**Definition 3: (l-cut of preference orders)**

For a partial order preference P and natural number l, a subset of values $C \subseteq P$ is called $l$-cut, if

(a) $\forall v \in C : \text{level}(v) \leq l$

(b) $\forall (w \in P \setminus C) \exists v \in C : v >_P w$

A set of database objects $D$ forms an instance of an $l$-cut $C$ if for each $v \in C \exists o \in D: o$ has attribute value $v$. An $l$-cut $C$ is minimal, if no subset $C' \subset C$ is an $l$-cut.

The intuitive meaning of $l$-cuts is to form sets of attribute values that if instantiated by database objects, dominate all object values beyond the $l$-th level. Every completely instantiated level of values forms a trivial $l$-cut. But generally $l$-cuts will be much smaller, and in the following we only need to consider minimum $l$-cuts.

**Example 1 (cont.):** Every single red car is instance of a 1-cut with respect to $P_2$. A 2-cut is instantiated by any pair of a blue and a black car. Regarding preference $P_1$, every roadster is instance of the 1-cut, every coupé instantiates a 2-cut, and so on.

For efficient pruning in our skyline evaluation we have to allow for quick tests whether a set of objects instantiating an $l$-cut has already been accessed. Hence, our first step in query evaluation is to compute all minimal $l$-cuts for each preference dimension. If we later find some object set instantiating any such cut we have found a pruning threshold. We now present a simple way to calculate minimal $l$-cuts. We first split the preference graph into levels, according to Definition 2:
Algorithm 1 (calculating attribute value levels)

0. Select \( \text{level}_0 \) as the set of all maximum attribute values in a preference graph \( P \), i.e. all attribute values that are not dominated by any other attribute value. \( l := 1 \)
1. \( \text{level}_{l+1} := \emptyset \)
2. While there are attribute values in \( \text{level}_l \) do
   2.1. Consider the next attribute value \( x \) in \( \text{level}_l \)
   2.2. For each attribute value \( y \) directly dominated by attribute value \( x \) with respect to \( P \) do
      2.2.1. If \( y \not\in \text{level}_0 \cup \ldots \cup \text{level}_{l+1} \), then \( \text{level}_{l+1} := \text{level}_{l+1} \cup \{ y \} \)
      2.2.2. If \( y \in \text{level}_j \) for some \( j \leq l \), then remove \( y \) from \( \text{level}_j \) and set \( \text{level}_{l+1} := \text{level}_{l+1} \cup \{ y \} \)
3. If \( \text{level}_{l+1} \) is not empty, set \( l := l+1 \) and proceed with step 1.

From these level sets, we can now determine minimal \( l \)-cuts. Obviously, each complete set \( \text{level}_l \) is a cut candidate, because all objects having attribute values in \( \text{level}_l \) with \( l < j \) are dominated by some object having an attribute value from set \( \text{level}_j \). Moreover, if we replace some cut element by any object dominating that cut element, the resulting set still forms a cut. Thus, to find all possible cut candidate value sets, we have to systematically enumerate all possible replacements. For this purpose, we first build a cut candidate value set from each complete \( \text{level}_l \) and then exhaustively replace attribute values by dominating values. Finally we remove redundant values to identify minimal cuts.

Algorithm 2 (calculating minimal cut value sets)

0. Given \( n \) sets of attribute values \( \text{level}_1 \), \ldots, \( \text{level}_n \) as output by algorithm 1 and initialize \( \text{candidates}_1 \), \ldots, \( \text{candidates}_n := \emptyset \), \( \text{replace}_1 \), \ldots, \( \text{replace}_n := \emptyset \) and \( \text{mincuts}_1 \), \ldots, \( \text{mincuts}_n := \emptyset \)
1. For \( l := 1 \) to \( n \) do
   1.1. If \( \text{level}_l \not\in \text{candidates}_l \) then \( \text{candidates}_l := \text{candidates}_l \cup \{ \text{level}_l \} \)
   1.2. For \( j := 1 \) to \( \text{level}_l \) do
      1.2.1. Consider the \( j \)-th attribute value \( a_i \) in an enumeration of \( \text{level}_l \) and initialize \( \text{replace}_i := a_i \)
      1.2.2. For each \( y \) with \( a_i \prec y \) and \( y \not\in \text{replace}_i \) do \( \text{replace}_i := \text{replace}_i \cup \{ y \} \)
   1.3. Generate all possible combinations \( \{ x_1, \ldots, x_{\text{level}_l} \} \) with \( x_l \in \text{replace}_l \) and in each combination remove redundant attribute values, i.e. duplicates and values dominated by another value in the set. \( \text{candidates}_l := \text{candidates}_l \cup \{ x_1, \ldots, x_{\text{level}_l} \} \)
   1.4. Consider all candidate sets \( \text{cand} \) in \( \text{candidates}_l \) and if no subset of \( \text{cand} \) is in \( \text{candidates}_n \), \( \text{mincuts}_l := \text{mincuts}_l \cup \{ \text{cand} \} \)

71
Algorithm 2 is exponential in the size of the partial order preference. However, this size is typically rather small, and $l$-cut computation is always independent of the actual database size. Therefore, query processing efficiency is dominated by the actual skyline computation described in the next subsection. Please note that we nevertheless include the cost of the minimal $l$-cut computations in the query processing times in all our experiments.

3.3 Correctly Pruning Database Objects

In each preference we identified all minimal $l$-cuts. Now we are ready to present a way for pruning irrelevant parts of the database without missing elements of the restricted skyline. This is the major component needed to build an efficient evaluation algorithm for partial order preference queries under the weak Pareto dominance paradigm. The following theorem will show a sufficient condition to correctly prune database objects:

**Theorem 1:** (absence of false negatives)

Let $O$ be a set of database objects and $S_1, \ldots, S_n$ be level-ordered lists of $O$ with respect to partial order preferences $P_1, \ldots, P_n$. Let $o_1, \ldots, o_k \in O$ and assume that $o_1, \ldots, o_k$ have already been accessed in all level ordered lists and $\{o_1, \ldots, o_k\}$ form an $l$-cut with respect to $P_i$ for some numbers $i$ and $l$. Then no object that for all $1 \leq j \leq n$ occurs on a higher level than $l$ in $S_j$ can be part of the restricted skyline.

**Proof:** Let $\{o_1, \ldots, o_k\}$ be as defined above and $u \in O$ be an object that has not yet been accessed in any $P_j (1 \leq j \leq n)$. For the sake of contradiction we will assume that object $u$ belongs to the restricted skyline set, i.e. it is not weakly dominated by any other object. Since $\{o_1, \ldots, o_k\}$ form an $l$-cut in $P_i$, $u$ has to be dominated by some object $o_m (1 \leq m \leq k)$ with respect to $P_i$. Because we have assumed $u$ to be not weakly dominated by any other object, there has to be at least one preference where $u$ dominates $o_m$. Since $o_m$ has already been accessed in all $P_j$ and $u$ has not yet been accessed with respect to any $P_j$, its level $\text{level}(u) \geq \text{level}(o_m)$. Now according to Lemma 1 $u$ cannot dominate $o_m$ in any preference and thus must be weakly dominated. This contradicts the assumption of $u$ being part of the restricted skyline. ■

![Figure 4. False positives due to pruning](image-url)
Now we know that unseen objects can never be part of the restricted skyline and can be
correctly pruned after we have a completely known set of objects that instantiates an l-cut.
Unfortunately, due to the intransitivity of weak Pareto dominance, in some rare
cases an unseen object could still weakly dominate a member of the restricted skyline
candidate set, thus resulting in a false positive.

**Example 3:** Given preferences $P_1$ on car types and $P_2$ on colors in Figure 4 and the
following database instance: a blue coupé, a blue SUV, a green SUV and a grey
limousine. Let us assume that we have iterated over the sorted lists up to level 2 in each
preference, i.e. we have seen all roadsters, coupés and SUVs and all red, blue and black
cars. Given the database instance, we have accessed the blue coupé, blue SUV and green
SUV. Moreover the first two items have been accessed in both preferences and form a 2-
cut with respect to $P_1$. Following theorem 1 we can now prune all remaining objects, i.e.
the grey limousine. This pruning is indeed correct, since the grey limousine is weakly
dominated by the blue coupé. But whereas neither the blue coupé, nor the blue SUV
dominate the green SUV, it is dominated by the pruned grey limousine and thus a false
positive in the restricted skyline set.

As we can see from Example 2, preference graphs where such false positives can occur
have to consist of long isolated branches and the database instance should be rather
sparse on top objects. In fact, finding these conditions in all preferences is very unlikely
(cf. Section 4.7).

### 3.4 Efficiently Approximating the Restricted Skyline

For computing the correct restricted skyline we have to
- derive the Pareto skyline,
- test all elements against all other database objects for weak Pareto dominance, and
- finally remove all weakly dominated objects.

However, this is very inefficient since for Pareto skyline computation with partial order
preferences usually all database objects have to be accessed (for example on a database
with only 500,000 tuples and 5 partial order preferences calculating the Pareto skyline
takes about 22 minutes). Exploiting sorted lists and the pruning condition defined in
section 3, in the following algorithm we will take a few false positives into account.
However, in return we may prune large parts of the database and thus get an efficient
query processing, while still always correctly deriving all objects of the restricted skyline
(for example calculating the approximate restricted skyline in the same scenario and
setting as above takes only 35 seconds).
Algorithm 3: (approx. restricted skyline computation)

0. Given a set of database objects $O$ and a query containing $n$ partial order preferences $P_1, \ldots, P_n$; given a set of $n$ sorted lists $S_1, \ldots, S_n$ of $O$ with respect to $P_1, \ldots, P_n$ in level order. Initialize a set for all accessed objects $accessed := \emptyset$, sets for all objects accessed in the $n$ lists $accessed_1, \ldots, accessed_n := \emptyset$, a set for all objects already accessed with respect to all preferences $complete := \emptyset$, and a set for all objects currently under consideration $current := \emptyset$. Compute all sets of minimal cuts $\text{mincuts}_{i,l}$ for the $1 \leq i \leq n$ preferences and $1 \leq l \leq \text{maxlevel}$, levels, using Algorithms 1 and 2. Initialize a counter for the levels $l := 1$, for the current preference $i := 1$.

1. If none of the preferences $P_1, \ldots, P_n$ has an $l$-th level, then return $\emptyset$ as the restricted skyline and terminate. If preference $P_i$ has no $l$-th level, proceed with step 4.

2. Get all attribute values of level $l$ for the $i$-th preference $P_i$ and iterate over list $S_i$ retrieving all objects having any of these attribute values into the set $current$.

3. If $current \neq \emptyset$ then
   3.1. $accessed_i := accessed_i \cup current$ and $accessed := accessed \cup current$
   3.2. $complete := accessed_1 \cap \ldots \cap accessed_n$
   3.3. If there exists some set of objects $C \subseteq complete$ such that the respective set of $i$-th attribute values of the objects in $C$ is equal to some element of $\text{mincuts}_{i,l}$, i.e. the objects in $C$ instantiate an $l$-cut with respect to $P_i$ and have already been accessed in all lists $S_1, \ldots, S_n$, do
      3.3.1. For $j := i + 1$ to $n$ do get all attribute values of level $l$ for the $j$-th preference $P_j$ (if level $l$ exists in $P_j$) and iterate over list $S_j$. Union all objects having any of these attribute values with the sets $accessed_i$ and $accessed$ like in step 3.1.
      3.3.2. $complete := accessed_1 \cap \ldots \cap accessed_n$
      3.3.3. Compare all objects from set $accessed$ pairwise for weak domination and subsequently remove all weakly dominated objects from set $accessed$.
      3.3.4. Return the set $accessed$ as the restricted skyline and terminate.

4. If $i < n$, then set $i := i + 1$, else set $i := 1$ and $l := l + 1$. Set $current := \emptyset$ and proceed with step 1.

Basically the algorithm iterates over the preference information in a round robin fashion. It considers all objects that form a level in a preference. Of course instead of using sorted lists, all objects with a certain attribute value could also be retrieved using a database index (step 2). The algorithm then checks if an $l$-cut has been instantiated by completely known objects on the current level, and – if not – proceeds to process the next preference. Whenever a round is complete, it proceeds to the next level. If an $l$-cut has been instantiated by completely known objects, the current level is completed in all preferences and all higher levels are pruned (which is correct according to Theorem 1). The algorithm then checks for weak dominations and removes all dominated objects.
4. Evaluation

To evaluate the performance of our algorithm and compare skyline sizes, we conducted extensive experiments with various parameter settings. To avoid bias, both data and preferences are synthesized randomly, and we show averages over multiple runs in our evaluation. The database content is generated according to several different distributions. Preferences are generated based on several parameters:

- **Preference size.** The number of attribute values in a preference.
- **Preference depth.** The number of levels in the preference graph (cf. Definition 2).
- **Edge ratio.** The ratio between nodes and edges in the preference graph, i.e. the average node degree.

We evaluated different scenarios to study the influence of these parameters. In all scenarios, we measured the time required to compute the skylines (runtimes) and the skyline sizes for the restricted skyline, substitute values (SV) skyline and Pareto skyline. For restricted skyline computation, we use the algorithm described in Section 3. For all other skylines, we need to do a pair-wise object comparison for all object pairs.

Table 1 shows our default configuration used as baseline setting. In all experiments, parameters not explicitly mentioned are set to these default values. We ran all experiments on a 2.4 GHz AMD Opteron64 Dual-processor Linux machine, equipped with 20GB main memory. The algorithm is not (yet) parallelized, therefore only one processor was actually used. Memory consumption was not regularly captured. We only measured it for the largest database size (1 million objects), where the computation of restricted skylines required 811MB.

In the next sections, we describe each experiment and its outcome in detail. Please note that we always use a logarithmic scale for both time and size to suit the large differences between skyline types.

---

1 The BBS+ or SDC+ algorithms described in [CET05] may yield better runtime results for the Pareto skyline case. However, the experiments in [CET05] show results only for queries including 1 or 2 partially ordered preferences, and due to the underlying R-Tree indexing structure performance is bound to suffer for higher-dimensional skyline queries.
4.1 Influence of database size

To determine the influence of the database size in terms of our algorithm’s scalability, we varied the number of database objects between 50,000 and 1,000,000. Restricted skylines are in all cases computed by about two orders of magnitude faster than SV and Pareto skylines (see Figure 5a). In absolute figures, runtimes for Pareto skylines of more than 15 minutes on a powerful server can hardly be considered practical. Our algorithm can compute the skyline about two orders of magnitude faster. The dominant operations are pair-wise object comparisons which proceed for each object until a) a dominating object is found or b) the object has been compared to all others. Due to the weakened domination definition case a) occurs much more frequently in our approach. Additionally, on average far fewer comparisons are required until a dominating object is found. Figure 5b shows that the restricted skyline size starts very small (32 for 50,000 objects) and stays manageable even for large databases (297 for 1 million objects). In contrast, SV and Pareto skylines always comprise several thousand objects, already an unacceptable size for practical usage, e.g., in query refinement or relevance feedback. In summary, our proposed skyline algorithm scales well in terms of computation and skyline size.

4.2 Influence of query dimensionality

The goal in this scenario was to see how the number of preferences specified in a query affects skylines. After a small decrease in skyline sizes for 2-3 dimensions (where domination relationships are not yet outweighed by incomparability between the growing number of possible pairs of dimensions), the SV and Pareto skylines are touched by the curse of dimensionality. Like comparable work shows: their sizes quickly increase significantly up to nearly the whole database. On the other hand, the restricted skyline size only increases slightly (see Figure 6b). But what is more, even for large numbers of preferences it is still computed about an order of magnitude faster than SV and Pareto skylines, as shown in Figure 6a. We can state that the restricted skyline approach makes interactive refinement or feedback in high-dimensional skyline querying practical.
Influence of preference size and shape

In this experiment, we varied the preference size between 5 and 30 attribute values. Figure 7a shows that between 5 to 20 attribute values, SV and Pareto skyline sizes grow up to 20%, r.s. 25% of the database. Further increase of preference sizes doesn’t show a significant impact on skyline sizes. In contrast, the restricted skyline shrinks to a minimum at preference sizes of 20, and then stays fairly constant. For preference depth, we see a different picture (Figure 7b). Pareto and SV skyline shrink notably when increasing depth from 2 to 15. This happens due to the reduction of incomparable attribute values. For depth 15, we already get a linear dominance order, without any incomparable value pairs left. For this case, Pareto, SV, and restricted skylines becomes identical, since weak and strong dominance coincide.
4.4 Influence of Skewed data distribution

For our next set of experiments, we changed the distribution of our data collection. Using a Zipf distribution, we varied skews from uniform (skew parameter -1.0) to highly skewed (0.0). In the latter case, the most preferred attribute values in each preference is already assumed by 14% of all database objects. As we can see in Figure 8a, with growing skew the different skyline types coincide more and more. With so many objects having top attribute values, the chance for incomparability gets lower, and a set of rather similar top objects is bound to dominate the whole rest of the database. Similar effects can be observed when shifting the head of the Zipf distribution to the least preferred objects, thus creating a multitude of overall bad objects. In both cases, the restricted of skyline is computed an order of magnitude faster than the Pareto skyline.

4.5 Influence of Gaussian data distribution

Finally, we investigated the influence of Gaussian data distribution on skylines. Varying the standard deviation, we measured sizes and runtimes. This distribution encourages the creation of objects with medium preferred attribute values in all preferences. As Figure 8b shows, the restricted skyline size constantly stays about two orders of magnitude lower than in the Pareto and SV semantics case, independently of the standard deviation. Also here, the restricted skyline is computed about a magnitude faster than Pareto and SV skylines.

4.6 Coverage of Pareto by Restricted Skyline

To investigate how good the Pareto skyline is covered by the restricted skyline, i.e., how representative our selection from the original skyline set is, we performed a separate evaluation. We compared the coverage of the restricted skyline over the full Pareto skyline with the coverage of a random sample of the Pareto skyline. As measure for

---

Figure 8. a) Zipf skew and b) Gaussian distribution effect on skyline size
coverage, we use the average minimum distance of all Pareto skyline points to the objects in the restricted skyline. Small average minimum distances show a good approximation of the original set. To calculate this measure, we select for each object in the Pareto skyline the nearest object of the restricted skyline, and compute their Euclidean distance. As we have no natural numeric distances, we use again the level order to translate preference differences to numeric difference: for each preference $P$, the object value $v$ is replaced with the numeric value $v' = \text{level}(v) / \text{maxlevel}$. The same measure is used to compute the coverage of an equally large random sample of the Pareto skyline. Such a random sample can be seen as optimal regarding representativeness, with respect to its size. As shown in Figure 9, the restricted skyline exhibits nearly the same coverage as the random sample. This shows that the restricted skyline does not bias toward a specific area of the Pareto skyline.

4.7 Occurrence of False Positives

In all described settings, besides computing the restricted skyline according to Algorithm 3, we also computed it by exhaustive comparison of all database objects, to identify false positives. Even with our small edge ratio of 1.2, we did not encounter a single false positive. A closer look shows that it is indeed highly improbable to create preference graphs with long isolated branches, while at the same time having a database instance where the values at dominating positions are not occupied by some object.

5. Summary and Conclusions

Although skylines on partial order domains gain importance in practical applications due to their intuitive query capabilities, their evaluation times and especially their large result set sizes are still hampering their usefulness in typical interactive tasks, such as query refinement or for providing relevance feedback. Therefore the concept of weak Pareto dominance has recently been introduced, allowing to derive the restricted skyline. Restricted skylines generally allow to retrieve only the best matching objects with
respect to the user’s preferences. Moreover, the relaxed semantics of restricted skylines usually lead to intuitive results.

For fast query processing, we designed an efficient evaluation algorithm to approximate restricted skylines. It iterates over object lists for each preference, topologically sorted according to the level order of the respective preference. Hence, our algorithm allows for the pruning of possibly large irrelevant chunks of the database with proven correctness. While the complete restricted skyline is retrieved, some false positives can theoretically occur in the approximation. However, our evaluation indicates that these cases are very rare, and the amount of false positives is negligible in practice.

To quantify the practical impact of our approach, we performed extensive experiments. Varying preference characteristics and data distributions, our experiments show that restricted skylines are efficient to compute, as well as lean in size. Restricted skylines can be computed generally up to two orders of magnitude less expensive than Pareto skylines and stay lean even in the face of growing database sizes. They are also significantly less prone to the curse of dimensionality in face of larger numbers of user-provided preferences. Moreover, compared to similar-sized, representative random samples of the original Pareto skyline, restricted skylines do not exhibit a significant bias.

In summary, restricted skylines together with the proposed evaluation algorithm do indeed provide useful ‘prime cuts’ of the original Pareto skyline to the user: efficient to compute, suitable for higher dimensions, and representative.

Our future work will focus on reconciling skyline computations with utility-based ranking schemes, at least up to a certain point. In that respect, our level-ordering and sorted object lists can be seen as a first step towards mappings from purely qualitative rankings to approximate utilities for characteristic attribute combinations.

References


Ranking von Produktempfehlungen mit präferenz-annotiertem SQL

Matthias Beck, Sven Radde und Burkhard Freitag
{Matthias.Beck, Sven.Radde, Burkhard.Freitag}@uni-passau.de


1 Einleitung

Der Bedarf an Beratungssystemen steigt durch die zunehmende Kontextualisierung ständig. Eines der prominentesten Beispiele sind die personalisierten Buchempfehlungen von Amazon [LSY03]. Wichtig ist, die Präferenzen des Nutzers zu berücksichtigen und für ein Ranking der Treffermenge zu verwenden. Es ist daher wünschenswert, dem Nutzer die Möglichkeit zu geben, seine Wünsche und Präferenzen auf einfache und verständliche Weise zu spezifizieren.

Der Beitrag dieser Arbeit ist eine einheitliche Methode zur Annotation von SQL-Anfragen mit Gewichten, die zusätzlich die Angabe von Softconstraints erlaubt. Basierend auf den angegebenen Gewichten wird ein Ranking der Resultatrelation definiert. Dabei bleibt im Gegensatz zu anderen Verfahren (s. Abschnitt 7) die Antwortsemantik der Anfrage erhalten. Dies und die Tatsache, dass nur die SQL-Anfrage annotiert wird, macht die Benutzung beliebiger Datenbanksysteme möglich. Das schließt aber nicht aus, durch Anpassung des Anfrageoptimierers die Ranking-Information bereits bei der Anfrageauswertung nutzbringend einzusetzen.

2 Anwendungsfall

Der heutige Markt für Mobilfunkprodukte ist im wesentlichen durch kurze Produktzyklen und die hohe Innovationsfolge der technischen Entwicklung gekennzeichnet. Dabei wird besonders deutlich, dass Kunden nur durch qualifizierte Beratung eine optimale Kaufentscheidung treffen können, was einen dauerhaft hohen Schulungsaufwand für das Personal bedeutet. Datenbankgestützte Beratungssysteme können aufgrund ihrer zentralen Wartbarkeit zeitnah auf Änderungen der Produktlandschaft reagieren. Sie haben somit den Vorteil, das Beratungspersonal stets mit aktuellen Daten und Informationen zum Produktkatalog versorgen zu können. Dadurch unterstützen sie den Beratungsverlauf, was den Schulungsbedarf des Personals bei gleichbleibend hoher Beratungsfähigkeit reduziert.

Darüber hinaus stellen die heute im Web verfügbaren “Beratungssysteme” meist eher einfache “Konfiguratoren” dar, welche vom Kunden ein relativ großes Domänenwissen fordern, um gute Resultate zu liefern. Hochwertige Onlineberatung kann hier einen großen Beitrag zur Kundenzufriedenheit leisten und auch dazu dienen, neue Käuferschichten zu erschließen, welche von den bisherigen Möglichkeiten des Online-Shoppings nicht angeprochen werden.


Die Sortierung der Empfehlungen kann verfeinert werden, sobald detailliertere Informationen über die Präferenzen des Kunden verfügbar sind. Beispielsweise ergeben sich innerhalb der Klasse „Multimediahandy“ sicherlich Qualitätsunterschiede z.B. hinsichtlich der Auflösung der Digitalkamera oder bei den vom Medienplayer abspielbaren Formaten, so dass die Resultatmenge innerhalb dieser Klasse weiter sortiert werden kann. Die Präferenzen des Benutzers bestimmen wieder die Gewichtung zwischen diesen beiden Faktoren (also ob ggf. eine bessere Digitalkamera oder ein besserer Medienplayer bevorzugt wird). Zu beachten ist, dass diese feinere Sortierung innerhalb der vorher deﬁnierten größeren Klassen erfolgt.

Neben der Aufgabe, diese Präferenzen vom Kunden zu erfragen, muss einem Beratungssystem also eine geeignete Technik zur Verfügung stehen, an den Produktkatalog Anfragen entsprechend den oben beispielhaft erläuterten Anforderungen zu stellen. Aufgrund der weiten Verbreitung relationaler Datenbanken bietet es sich an, hierfür herkömmliche SQL-Anfragen in geeigneter Weise zu annotieren, so dass die Präferenzen des Kunden als Gewichte in die Auswertung einfließen können.
3 Gewichtsannotiertes SQL

Das gewichtsannotierte SQL soll zwei Anforderungen erfüllen. Erstens ist es notwendig, die Zugehörigkeit zu unterschiedlichen Klassen unterschiedlich gewichten zu können, und zweitens sollen diese Gewichte ein Ranking und damit eine Sortierung der Resultatmenge ermöglichen. Dies erhält dem Kunden seine freie Wahlmöglichkeit, während die empfohlenen Produkte bevorzugt dargestellt werden können (in der Form von „Top-10-Empfehlungen“ o.ä.).

In einem Beispielszenario soll nun die Zugehörigkeit zur Klasse MultimediaHandy mit einem Gewicht von 2 und zur Klasse BusinessHandy mit einem Gewicht von 1 annotiert werden. Nimmt man der Einfachheit halber an, dass diese beiden Klassen jeweils durch ein boolesches Attribut in einer Tabelle handy repräsentiert werden, so kann die Anfrage folgendermaßen formuliert werden:

**Beispiel 1**

```sql
SELECT * FROM handy
WHERE (MultimediaHandy = 1) [2] OR (BusinessHandy = 1) [1]
```

Die intuitive Bedeutung dieser Anfrage ist dabei wie folgt: In der sortierten Resultatmenge sollen die Mobiltelefone, die zu beiden Klassen gehören, an der Spitze stehen, gefolgt von den reinen Multimedia-Handys, zuletzt die reinen Business-Handys. Besteht die Selektionsbedingung wie im Beispiel aus einem n-stelligen OR, so steigt die Anzahl der Klassen-Kombination und damit, bei geeignet gewählten Gewichten, die Anzahl der Werte, die der Rang eines Tupels annehmen kann, exponentiell in der Anzahl der Operanden n. Fügt man in die OR-Bedingung aus Beispiel 1 nur einen weiteren Operanden ein, so erhöht sich die Anzahl der Klassen bereits auf acht. Somit lässt sich bereits mit einer relativ kleinen Anzahl von gewichteten Ausdrücken ein differenziertes Ranking erreichen. Dabei müssen für jedes Tupel aber nur die spezifizierten n Bedingungen ausgewertet werden.

Die Realisierung einer solchen Sortierung muss folgende Anforderungen erfüllen: Ein Gewicht 0 darf keinen Einfluss auf die Sortierung nehmen. Je höher andererseits die Gewichte der Klassen sind, denen ein Element der Resultatmenge angehört, umso besser soll der Rang des Elementes sein.

Allgemein gilt die Form \((expression)[weight]\) für einen gewichtsannotierten Ausdruck. *expression* steht dabei für einen beliebigen booleschen Ausdruck, der in der Where-Klausel einer Anfrage gestattet ist. Erlaubt sind insbesondere Ausdrücke der Form \((table.column <= const)\), geschachtelte Ausdrücke wie \((E and F)\) und Ausdrücke, die Subqueries enthalten, wie \((exists select ... )\). Ebenfalls möglich ist eine Schachtelung von Gewichten: \(((A) [1] and (B) [4]) [2] or (C) [3])\). Falls für einen Ausdruck kein Gewicht angegeben ist, wird als Standard ein Gewicht von 0 verwendet.

Eine natürliche Einschränkung ist, dass gewichtsannotierte Ausdrücke nur als (direkte) Bestandteile einer Disjunktion oder Konjunktion möglich sind. Ein allein stehender Ausdruck der Form \((A or B) [1]\) ist daher nicht sinnvoll, im Gegensatz zu \(((A) [1] or (B) [2])\). Diese Restriktion bedeutet keine Einschränkung der Ausdrucksmächtigkeit, denn nur innerhalb von Disjunktionen bzw. Konjunktionen gibt es mehrere Klassen, die relativ zueinander gewichtet werden können. Dies ist beim ersten Ausdruck nicht der Fall. Wäre
dieser Ausdruck das Selektionskriterium einer Anfrage, dann würden alle Elemente des Resultats mit 1 gewichtet werden, da sie alle die Bedingung \((A \lor B)\) erfüllen.

4 Ranking


4.1 Rank Mapping

Für eine gegebene gewichtsannotierte Query \(Q\) wird zunächst die Ergebnisrelation\(^1\) \(R\) berechnet. Anschließend wird auf jedes Tupel \(\mu \in R\) das Rank Mapping angewendet. Ergebnis dieses Mappings ist dann ein Rank Mapping Tree für jedes Ergebnistupel.

**Definition 1 (Rank Mapping)** Sei \(DB\) eine Datenbankinstanz, \(Q\) eine Selektionsbedingung mit Gewichtsannotationen und \(\mu\) ein Resultattupel bzgl. \(Q\). Sei weiter \(\odot \in \{\text{AND}, \text{OR}\}\). Dann definieren wir das Rank Mapping \(r_{DB}^{\text{start}}(Q, \mu)\) durch

\[
\begin{align*}
 r_{DB}^{\text{start}}(Q, \mu) &:= (0, 1, r_{DB}(Q, \mu)) \\% (1) \\
r_{DB}(C_1^{\odot} \circ \ldots \circ C_n^{\odot}, \mu) &:= ((r_1, d_1, r_{DB}(C_1, \mu)), \ldots, (r_n, d_n, r_{DB}(C_n, \mu)))% (2) \\
\text{wobei} & \quad d_i = \begin{cases} 1, & \text{falls } C_i[\mu] \text{ wahr in } DB \\ 0, & \text{sonst} \end{cases}
\end{align*}
\]

\(^1\)Zur Berechnung der Gewichte ist die Ergebnisrelation vor einer Projektion auf die Elemente der Select-Liste erforderlich.

85
\[
\begin{align*}
 r^{\neg}_{DB}(C_1 \land \ldots \land C_n, \mu) & := ((r_1, d_1, r^{\neg}_{DB}(C_1, \mu)), \ldots, (r_n, d_n, r^{\neg}_{DB}(C_n, \mu))) \quad (3) \\
 & \text{wobei} \quad d_i = \begin{cases} 
 1, & \text{falls } (\neg C_i)[\mu] \text{ wahr in } DB \\
 0, & \text{sonst}
\end{cases}
\end{align*}
\]


Durch die Duplikateliminierung bei einer möglichen anschließenden Projektion auf die Attribute der Select-Liste kann der Fall auftreten, dass zu einem Resultattupel \( \mu \) mehrere Ranking Trees \( T_\mu \) entstehen. In einem solchen Fall wird ein ‘guter’ Ranking Tree \( t \in T_\mu \) gewählt, d. h. es muss gelten: \( \forall t' \in T_\mu : t \not< t' \).

Abbildung 1a-b zeigt an einem Beispiel den Aufbau des Rank Mapping Trees. Dieser speichert die grundlegenden Informationen des Rankings. Der Wurzelknoten \((0,1)\) ist vordefiniert. Alle weiteren Knoten ergeben sich durch Auswertung von Teilbedingungen der annotierten Query \( (A \land B) \lor C \). So ergibt sich der Knoten \((3,1)\) durch Auswertung des Ausdrucks \( C \) bezüglich des betrachteten Tupels \( \mu_1 \). Die erste Komponente ist das Gewicht des Teilausdrucks \(3\). Die zweite Komponente \((1)\) zeigt, dass für \( \mu_1 \) der Teilausdruck \( C \) erfüllt ist. Zur besseren Übersicht enthält der Baum in Abbildung 1b keine (leeren) Blätter (vgl. Gleichungen 6 und 7 in Definition 1).

Für eine sinnvolle Auswertung ist es erforderlich, diese Informationen unter Verwendung einer Aggregationsfunktion zu verdichten. Resultat dieser Aggregation ist dann der Ranking Tree.

**Definition 2 (Level Aggregation Function)** Eine Funktion \( \bigcup_{\mathbb{N}_0} (\mathbb{N} \times \{0, 1\})^1 \to \mathbb{R} \) ist eine Level Aggregation Function.

**Definition 3 (Weighted Average Level Aggregation Function)**
\[
\text{agg}_{wa}((n_1, b_1), \ldots, (n_m, b_m)) := \begin{cases} 
 0, & \text{falls } \forall 1 \leq i \leq m : n_i = 0 \\
 \frac{\sum_{i=1}^{m} n_i b_i}{\sum_{i=1}^{m} n_i}, & \text{sonst}
\end{cases}
\]

Nach den bisher gesammelten Erfahrungen ist der gewichtete Durchschnitt als Level Aggregation Function für die vorgestellten Anforderungen gut geeignet. Alternativen wären
etwa Maximum, gewichtete Summe etc. Der Ranking Tree entsteht nun durch rekursive Anwendung einer Level Aggregation Function auf jeden Knoten des ursprünglichen Rank Mapping Trees.

**Definition 4 (Tree-based Level Aggregation)** Sei \( \text{agg} \) eine Level Aggregation Function, \( t \) ein Rank Mapping Tree. Dann wird die Tree-based Level Aggregation \( t_{\text{tl}} \) induktiv folgendermaßen definiert:

1. Falls \( t = (r, b, ((r_1, b_1, \text{sub}_1), \ldots, (r_n, b_n, \text{sub}_n))) \) mit \( n > 0 \), so sei \( t_{\text{tl}}(t) := (\text{agg}((r_1, b_1), \ldots, (r_n, b_n)), t_{\text{tl}}(\text{sub}_1), \ldots, t_{\text{tl}}(\text{sub}_n)) \).
2. sonst \( t_{\text{tl}}(t) := (0) \)

In Abbildung 1b-c wird exemplarisch der Übergang vom Rank Mapping Tree zum Ranking Tree unter Verwendung des gewichteten Durchschnitts gezeigt. Der Wurzelknoten \( \frac{3}{4} \) beispielsweise ergibt sich durch Berechnung des gewichteten Durchschnitts aus den Tupeln \((4,0)\) und \((3,1)\), d. h. \( \frac{4 \cdot 0 + 3 \cdot 1}{4 + 3} \).

### 4.2 Ranking Tree

Jedem Tupel der Ergebnisrelation wird durch das Rank Mapping ein Ranking Tree zugeordnet.

**Definition 5 (Ranking Tree)**

1. \((0)\) ist ein Ranking Tree (Blattknoten).
2. Sei \( r \in \mathbb{R} \) und seien \( t_1, \ldots, t_n \) Ranking Trees mit \( n \geq 1 \). Dann ist \((r, t_1, \ldots, t_n)\) ein Ranking Tree.

Um das Ziel eines Rankings der Ergebnisrelation auf Grundlage der erhaltenen Bäume zu erreichen, ist es erforderlich, eine Ordnung auf Ranking Trees zu definieren. Die Definition der Ordnung erfolgt auf Grundlage der Ebenen des Baumes. Ein solches Level wird dabei wie folgt definiert:
Definition 6 (Ranking Tree Level) Sei $a = (r, a_1, \ldots, a_n)$ ein Ranking Tree. Dann definieren wir induktiv die Funktion $level$. Das resultierende $n$-Tupel $(r_1, \ldots, r_n)$ ist ein Ranking Tree Level.

\[
level_0(a) := (r)
\]
\[
level_m(a) := \begin{cases} 
level_{m-1}(a_1) \circ \cdots \circ level_{m-1}(a_n), & \text{falls } n > 0 \\
(), & \text{sonst}
\end{cases}
\]

Abbildung 2 veranschaulicht den Grundgedanken der Ordnung. Die drei abgebildeten Ranking Trees entstehen aus den Tupeln $\mu_1$ bis $\mu_3$ von Abbildung 1 bei Verwendung des gewichteten Durchschnitts. Wenn die Wurzeln der Bäume bereits eine Unterscheidung ermöglichen, müssen die Kindknoten nicht weiter betrachtet werden (2b-c). Bei gleicher Wurzel entscheidet die Ebene der Kindknoten (2a-b). Sollte auch diese Ebene übereinstimmen, wird die nächste Ebene einbezogen usw.

Definition 7 (Strict Order on Ranking Trees) Es seien zwei Ranking Tree Level $l = (l_1, \ldots, l_n)$ und $k = (k_1, \ldots, k_n)$ gegeben. Dann gilt $l < k$ g.d.w. $\forall 1 \leq i \leq n : l_i < k_i$. Seien jetzt $a$ und $b$ Ranking Trees mit gleicher Struktur. Dann gilt $a < b$ g.d.w.

\[\exists m : level_m(a) < level_m(b) \quad \text{und} \quad \forall 0 \leq i < m : level_i(a) = level_i(b)\]

Die entstehende Ordnung ist partiell. Sie lässt sich jedoch zu einer totalen Ordnung vervollständigen, indem unvergleichbare Elemente auf konsistente Weise in die Ordnung integriert werden. Dies ist im einfachsten Fall z. B. unter Rückgriff auf den Tupelidentifikator o. ä. möglich.

Die Ranking Trees haben eine weitere nützliche Eigenschaft, bieten sie doch durch ihre Struktur direkt eine Erklärungskomponente. Jede Gewichtung, die der Nutzer vergeben hat, findet sich im Ranking Tree repräsentiert und kann so mit minimalen Nachbearbeitungsaufwand direkt verwendet werden, um dem Nutzer Hinweise zu geben, wie er andererseits durch Veränderung der Gewichte ein verändertes Ranking bewirken kann.

5 Erweiterte Möglichkeiten

Oft sind bestimmte Eigenschaften eines Mobiltelefons nicht zwingend erforderlich, wären aber wünschenswert. Die Erfüllung einer solchen Bedingung, eines Softconstraints, sollte sich in einem höheren Rang in der Ergebnisrelation widerspiegeln.


88
SELECT * FROM handy
WHERE (MultimediaHandy = 1) [2] OR (BusinessHandy = 1) [1]
  OR ((Sonderangebot = 1) [1] AND (0=1))

Abbildung 3 zeigt die Ranking Trees für zwei reine Businesshandys, von denen nur das linke ein Sonderangebot ist. Durch die unerfüllbare Teilbedingung (0=1) wird der gesamte Teil (Sonderangebot = 1) AND (0=1) unerfüllbar. Die eigentliche Selektionsbedingung bleibt also im Vergleich zur ursprünglichen Anfrage aus Beispiel 1 unverändert. Andererseits unterscheiden sich, wie Abbildung 3 zeigt, die Ranking Trees, je nachdem ob ein Sonderangebot vorliegt oder nicht. Diese Information kommt dann beim Ranking der Resultatmenge zum Einsatz. Die etwas umständliche Formulierung der Form (... or (Softconstr [n] and false)) lässt sich durch Einführung eines neuen Schlüsselwortes SOFT leicht zu (... or (SOFT Softconstr [n])) umformen, um die Lesbarkeit zu erhöhen.

Abbildung 3 zeigt die Ranking Trees für zwei reine Businesshandys, von denen nur das linke ein Sonderangebot ist. Durch die unerfüllbare Teilbedingung (0=1) wird der gesamte Teil (Sonderangebot = 1) AND (0=1) unerfüllbar. Die eigentliche Selektionsbedingung bleibt also im Vergleich zur ursprünglichen Anfrage aus Beispiel 1 unverändert. Andererseits unterscheiden sich, wie Abbildung 3 zeigt, die Ranking Trees, je nachdem ob ein Sonderangebot vorliegt oder nicht. Diese Information kommt dann beim Ranking der Resultatmenge zum Einsatz. Die etwas umständliche Formulierung der Form (... or (Softconstr [n] and false)) lässt sich durch Einführung eines neuen Schlüsselwortes SOFT leicht zu (... or (SOFT Softconstr [n])) umformen, um die Lesbarkeit zu erhöhen.

Abbildung 3 zeigt die Ranking Trees für zwei reine Businesshandys, von denen nur das linke ein Sonderangebot ist. Durch die unerfüllbare Teilbedingung (0=1) wird der gesamte Teil (Sonderangebot = 1) AND (0=1) unerfüllbar. Die eigentliche Selektionsbedingung bleibt also im Vergleich zur ursprünglichen Anfrage aus Beispiel 1 unverändert. Andererseits unterscheiden sich, wie Abbildung 3 zeigt, die Ranking Trees, je nachdem ob ein Sonderangebot vorliegt oder nicht. Diese Information kommt dann beim Ranking der Resultatmenge zum Einsatz. Die etwas umständliche Formulierung der Form (... or (Softconstr [n] and false)) lässt sich durch Einführung eines neuen Schlüsselwortes SOFT leicht zu (... or (SOFT Softconstr [n])) umformen, um die Lesbarkeit zu erhöhen.

Das Beispiel zeigt Softconstraints auf der OR-Ebene. Analog dazu gibt es auch die Möglichkeit, Softconstraints auf der AND-Ebene auszudrücken. Das entsprechende Muster ist (Bed_1[r_1] and ... and Bed_n[r_n] and (Softconstr [r] or true)).


SELECT ... FROM ...
WHERE ((Bluetooth=1) [1] OR (Infrarot=1) [1]
  OR (USB=1) [1] OR ...)

6 Implementierung

Abbildung 4: Auswertungszeit in Abhängigkeit von der Ergebnisgröße

Die Laufzeit des Verfahrens liegt für die im Anwendungsfall zu erwartenden Problemgrößen von maximal wenigen tausend Tupel in einem guten Bereich. So benötigt die Anfrage aus Beispiel 1 bei einer Ergebnisgröße von 1000 Tupeln 1278 ms, während die gleiche Anfrage als reines SQL ohne Ranking 1258 ms benötigt. Selbst bei einer Ergebnisgröße von 10.000 Tupeln ist der durch das Ranking hinzukommende Overhead im Vergleich zur Gesamlaufzeit gering (mit Ranking 1666 ms, ohne Ranking 1390 ms).

6.1 SQL

Für den Aufbau der Ranking-Trees muss für jedes Tupel der Ergebnisrelation die Information vorliegen, welche Bedingungen und Teilbedingungen der Where-Klausel für dieses Tupel erfüllt sind. Übertragen auf Beispiel 1 muss für jedes Resultattupel bekannt sein, ob es die Bedingungen (MultimediaHandy = 1) und/oder (BusinessHandy = 1) erfüllt. Dies lässt sich durch Case-Klauseln in der Select-Liste erreichen. Folgendes Statement wird auf diese Weise generiert:

```sql
SELECT h.*,
CASE WHEN (MultimediaHandy=1) THEN 1 ELSE 0 END AS RANK_EXPR_0,
CASE WHEN (BusinessHandy=1) THEN 1 ELSE 0 END AS RANK_EXPR_1
FROM Handy h WHERE (MultimediaHandy = 1) AND (BusinessHandy = 1)
```

\(^2\)Gemessen auf einem Pentium mit 1,5 GHz und 1 GB RAM mit einer PostgreSQL 8.0 Datenbank.
In diesem einfachen Fall könnten die Werte MultimediaHandy und BusinessHandy auch direkt selektiert werden, da ihr Wertebereich ebenfalls \{0, 1\} ist. Dies ist für allgemeine Bedingungen nicht möglich, so dass dann Case-Klauseln zwingend erforderlich sind.

6.2 JDBC-Treiber


```java
// Laden des Standard-Datenbank-Treibers
Class.forName("com.ibm.db2.jcc.DB2Driver");
// Laden des Ranking-Datenbank-Treibers
Class.forName("de.uni_passau.im.pref.sql.ranking.jdbc.RankedDriver");

DriverManager.getConnection("jdbc:rank:db2:...", user, pwd);
```


7 Zusammenfassung und Diskussion

Zur Realisierung von Beratungssystemen weit verbreitet ist der Einsatz der Collaborative-Filtering-Methode [CSP03, MS02], der u. a. in [LSY03] genutzt wird. Unser Ansatz konzentriert sich auf die explizite Ermittlung der Kundenpräferenzen, anstatt diese implizit über die schrittweise Bewertung von Beispielprodukten aus dem Katalog zu lernen. Durchaus möglich wäre es aber, die Gewichte aus bisher getroffenen (Kauf-)Entscheidungen des Nutzers zu lernen.


Eine Übertragung des PageRank-Algorithmus (Google) auf relationale Datenbanken präsentieren Hwan und andere in [HHP06].


8 Danksagung

Wir danken den anonymen Gutachtern für ihre hilfreichen Kommentare.

References


Hierarchy-driven Visual Exploration
of Multidimensional Data Cubes

Svetlana Mansmann  Florian Mansmann  Marc H. Scholl
Daniel A. Keim
University of Konstanz (Germany)
Department of Computer & Information Science
Fach D188, 78457 Konstanz

Firstname.Lastname@uni-konstanz.de

Abstract: Analysts interact with OLAP data in a predominantly “drill-down” fashion, i.e. gradually descending from a coarsely grained overview towards the desired level of detail. Analysis tools enable visual exploration as a sequence of navigation steps in the data cubes and their dimensional hierarchies. However, most state-of-the-art solutions are limited either in their capacity to handle complex multidimensional data or in the ability of their visual metaphors to provide an overview+details context.

This work proposes an explorative framework for OLAP data based on a simple but powerful approach to analyzing data cubes of virtually arbitrary complexity. The data is queried using an intuitive navigation in which each dimension is represented by its hierarchy schema. Any granularity level can be dragged into the visualization to serve as an disaggregation axis. The results of the iterative exploration are mapped to a specified visualization technique. We favor hierarchical layouts for their natural ability to show step-wise decomposition of aggregate values. The power of the tool to support various application scenarios is demonstrated by presenting use cases from different domains and the visualization techniques suitable for solving specific analysis tasks.

1 Introduction

The last decade has witnessed an explosion of visual interfaces for OLAP (OnLine Analytical Processing) - dashboards, charts, maps and scatterplots - which have impacted the business intelligence modern analytics. Interactive features, such as zooming, slicing, brushing and filtering, are becoming a commonplace in analysis software. With ever-growing volumes of accumulated data visualization becomes indispensable for extracting useful knowledge from data by a human expert. Adequate visual presentation helps to rapidly reveal patterns, recognize trends or anomalies. Especially the ad hoc queries, when an expert is guided by a mere guess or a hypothesis about the knowledge hidden in the "raw" data, benefit from the ability to visually specify the data set of interest and interact with it.

Analysis tools that abound the market offer a multitude of features and functions that
require training and skill to understand and use. Feature overload and usability deficiencies often lead to loss of orientation and discourage users from using novel techniques. The work reported in this paper is an attempt to design an OLAP interface advanced in terms of its exploration and visualization capabilities, but simple and intuitive in its usability with minimum training requirements. The applicability of a particular visualization technique depends on multiple criteria, such as the type of the analytical task, data volume and complexity, user preferences and skills. Our approach is to account for a variety of tasks by designing a comprehensive framework, in which users can experiment with various layouts and techniques to find satisfactory solutions to specific problems.

OLAP systems employ multidimensional data model to structure “raw” data into multidimensional cubes in which each data entry is shaped into a fact with associated numerical measure(s) and descriptive dimensions that characterize the fact [PJ01]. The values within a dimension can be further organized in a containment type hierarchy to support multiple granularities. The table containing the facts is referred to as fact table; other tables, called dimensional tables, store the dimensional values and the hierarchical relationship among them. Analytical queries aggregate measure values over a range of dimension values to provide the view of the desired dimensionality and granularity. The most common OLAP operations are slice-and-dice to define a sub-cube, drill-down and roll-up to perform aggregation and disaggregation, respectively, along a hierarchical dimension, drill-across to combine multiple cubes, ranking to find the outlier values, and rotating to see the data grouped by other dimensions.

While analytical queries aggregate over detailed data, visual exploration evolves in the inverse direction, i.e. descending from coarsely grained aggregates to more detailed views via stepwise decomposition along selected dimensions. The prevailing “from-overview-to-details” query direction is reflected in the structure of a typical OLAP data browser: each data cube is presented as a hierarchy of its dimensions where each dimension is a recursive nesting of its granularity levels, starting from the coarsest granularity and continuing with each finer level as a child of its predecessor. Users proceed by selecting the measure(s) and the aggregation function to invoke, the dimensions to use as decomposition axes, by filtering the selected data set and manipulating the visual representation of the result. These query steps are performed irrespective of the type of query and the chosen visualization technique. Therefore, we see a great usability improvement potential in designing a common uniform navigation framework for satisfying any type of analytical query.

The rest of the paper is organized as follows. Some outstanding related work on OLAP visualization is discussed in Section 2. In Section 3 we describe the data navigation as the core component of our proposed visual OLAP interface for visual exploration of data. Section 4 identifies the visualization techniques which best fit the needs of explorative analysis. Section 5 introduces the explorative framework as an abstraction of the mapping between user interaction and visual data representations. Finally, the concluding remarks are found in Section 6.
2 Related Work

An attempt to address the related work on designing visual interfaces for exploring multidimensional data in its entirety would explode the scope of this work. We limit ourselves to naming a few solutions which offer distinguished features relevant for our work.

Eick et al. presented a survey of common visual metaphors and associated interaction techniques with improved visual scalability, i.e. the capability to effectively display large volumes of multidimensional data [EK02]. The proposed techniques were implemented in the ADVIZOR system, in which each visual metaphor has a single zoom path. Rather than displaying multiple granularities within the same visualization, multiple perspectives, i.e. linked views displayed on the same screen, present various projections of data.

Stolte et al. proposed a class of multiscale, i.e. supporting multiple zoom paths, visualization techniques for data cubes in [STH03]. Implementation of these techniques is a part of an advanced system called Polaris [STH02], as well as its commercial successor Tableau Software [T06], which extend the Pivot Table interface by offering a combination of displays and tools for visual specification of analysis tasks.

Techapichetvanich et al. proposed a visualization framework for OLAP based on the Hierarchical Dimensional Visualization (HDDV) technique [TD05] which uses colored stacked bars. Each bar shows the results of decomposing the root aggregate along a chosen dimension, whereas the color is used to mark the portion of values satisfying the specified range condition. Bars are not explicitly linked to each other, allowing to split the same aggregate along multiple dimensions.

A work by Hellerstein et al. [HAC+99] elaborates on the mechanisms to improve human-computer interaction and user control when analyzing massive data sets. Their proposed framework combines visualization and data mining techniques to guide the user through the data discovery process.

ProClarity was the first to introduce a hierarchical drill-down visualization called Decomposition Tree [P06], which places each aggregate into a node and its constituent sub-aggregates, obtained by drilling down into any dimension, to its child nodes. Child nodes display the next level of detail from left to right by value and percentage of the total. ProClarity’s technique is designed to expand node by node for the "speed-of-thought" analysis, is limited to decomposing a single measure and has no visual formatting of the values. Besides, it is rather wasteful in terms of display utilization and is thus infeasible for exploring large data volumes.

Further enhancements to the hierarchical disaggregation are found in the OLAP web client Report Portal 2.1 released by XMLA Consulting [R06]. Report Portal offers graphical decomposition trees, such as BarChart and PieChart Trees, which arrange the entire drill-down view into a chart hierarchy. Moreover, drill-down steps are not aligned into dimensional levels, thus allowing to expand “sibling” values along different dimensions.

In a previous work we introduced an explorative framework based on enhanced decomposition trees [VM06] which implement various tree layouts and allow to choose between different visual presentations within the nodes. Schema-based navigation approach en-
ables efficient browsing and fast generation of visual hierarchies, support of complex di-

censional hierarchies and parallel exploration on multiple cubes along shared dimensions.

Classification of supported dimensional patterns can be found in [MS06]. In this work we

concentrate on the database aspects of transforming the cube schema into frontend naviga-

tion, translating user interactions into OLAP queries and arranging the results of successive

query steps into a specified visual format.

Whenever a data cube contains spatio-temporal characteristics, the analysis may benefit

from specialized exploration techniques for space-time patterns. A synopsis of techniques

for spatio-temporal exploration arranged according to the data and task types is produced

in [AAG03]. Kuchar et al. [KHHP06] point out that time dimension is not an ordinary data

attribute and that, therefore, to ensure satisfactory analysis, interaction and visualization

techniques have to incorporate explicit awareness of temporal characteristics.

3 Datacube as a Navigational Hierarchy

The entire explorative framework can be considered as composed of an input and an output

area for specifying queries and presenting query results, respectively. The input element

is the navigation which displays the structure and the contents of the data in form of a

browser for visual querying. The main area of the graphical interface serves for outputting

the results of user interaction in a selected visual format.

To be visually navigable, the facts are modeled according to the multidimensional data

type. Thereby, a straightforward spreadsheet approach with no distinction between struc-

ture and contents is inadequate for managing complex data where explicit knowledge of

the structure is needed to ensure the validity of generated OLAP operations.

3.1 Conceptual and Logical Design

Conceptual design is the initial phase of building a data cube. At this stage, the developer

investigates the user requirements as well as the available data and defines the structure

of the fact tables and of the associated dimensional hierarchies using modeling techniques

such as ER or Multidimensional ER model, UML or Dimensional Fact model.

The challenges of designing a data cube are demonstrated at an example drawn from an

existing university data warehouse. Decision-makers expect the data from decentralized

procurement systems to be extracted into a single fact table Expenditures with measures

amount and items for the amount paid and the number of ordered items, respectively. The

facts are determined by dimensions category for cost type, project as the order’s

destination, funding as the funding source, institution as the origin of the order, and period

for tracing the date of the purchase the expenditure was registered. Figure 1 shows the

expected structure of the cube as a dimensional fact schema. The schema is designed using

the Dimensional Fact model, introduced by Golfarelli et al. in [GMR98] and extended by

super-/subclass relationships to account for heterogeneity in dimension institution.
With the exception of funding, dimensional values are further arranged into hierarchies. Dimensions are presented as directed graphs with aggregation levels as nodes and “rolls-up-to” relationships between them as edges. A dimension’s graph is rooted at an abstract node with a single value $\top$, interpreted as all. The bottom-level node of each dimension is linked directly to the fact table. Temporal dimension period is a classical example of a multiple hierarchy: days can be aggregated along three distinct aggregation paths.

A dimension worthwhile special treatment is institution. It is typical for institutional hierarchies to be unbalanced and consist of multiple sub-classes, each with its own attributes and hierarchical levels. However, an analysis task may require a heterogeneous hierarchy to be “fetched” into a single dimension by means of generalization, i.e. superclass/subclass relationships. Standard OLAP tools, incapable of supporting generalization hierarchies, would only work on separate data marts, one for each homogeneous sub-class. Our approach admits heterogeneity within a dimension but enforces the hierarchy’s subdivision into homogeneous disjunct subclasses to ensure correct aggregation (further details can be found in [MS06]). Back to the example at hand, all institutional instances are subdivided into sub-classes of administrative and teaching institutions, each with its own hierarchy.

Relational OLAP employs star or snowflake schema, both based on arranging the data in fact and dimensional tables. Fact table is constrained to have one column per measure, as well as one column per dimension for storing the references to the values in the respective dimensional table. Star schema places the entire dimension, however complex it might be, into a single relation by pre-joining all aggregation levels, similarly to a spreadsheet. Snowflake schema normalizes hierarchical dimensions by placing each granularity into a separate table, thus eliminating redundant storage and facilitating consistency of updates.

Snowflake schema becomes the only option when dimensional hierarchies are prone to irregularities, such as heterogeneity, non-strictness, missing values, mixed granularity etc. Figure 2 shows the resulting snowflake schema institution. An auxiliary relation subclass stores the IDs and the names of all defined subclass categories. The constraint of being referenced by just a single attribute in the fact table is satisfied by the “trick” of merging the primary keys from service and chair into a single relation institution.
Upon the completion of the normalized snowflake schema, a star-schema view of the each homogeneous (sub-)dimension is generated. The view is defined as a join between all dimensional levels and its purpose is simply to facilitate the generation of the OLAP query from the visual navigation events (further explanation is given in the next subsection).

### 3.2 Navigation Hierarchy

Users access the data via a file-browser-like navigation hierarchy. The top-level navigation objects are the data cubes, consisting of three types of objects, namely the dimensional hierarchies, the measures, and the aggregation functions applicable to the measures, as shown in Figure 3(a). Dimensions are represented by their hierarchy schemata in a top-down fashion: the top level is the abstract root node $\top$ (with the exception of non-hierarchical dimensions which do not need an extra root node); each further hierarchy level is a sub-folder nested in the folder of its parent level. Non-hierarchical dimension levels, i.e. those with the finest granularity, are presented by non-folder icons.
Figure 3(b) shows the browser hierarchy of institution. Notice, that unlike the dominating OLAP navigation approaches, which directly display the data hierarchy of a dimension, our schema-based approach explicitly presents the hierarchy schema. Each schema node has a dimensional data table related to it; the values at each level are accessed via a “lens” icon, as illustrated in Figure 3(b) for the node department. Dimensional data can be further explored in the context of its hierarchy: data entities are supplied with “back” and “forward” arrows to navigate to that entry’s parent and child values, respectively.

Once the user has chosen the measure \( m \) and the aggregation function \( a \) from fact table \( f \), the query can be initialized as \( \text{SELECT } a(m) \text{ FROM } f \). Selection of any sub-dimension \( d_i \) in dimension \( d \) is interpreted as a drill-down along \( d \):

\[
\text{SELECT } d_i \text{.id, } a(m) \text{ FROM } f \text{ JOIN } \ldots \text{ JOIN } d_i \text{ GROUP BY } d_i \text{.id.}
\]

Filtering along any dimension is done by setting/unsetting values displayed in the dimension’s popup data view and is reflected in the WHERE clause.

Since dimensions in the snowflake schema are linked to the fact table only via the bottom granularity level, all higher levels referenced by a query need to be joined all the way down in order to reach the fact data. Figure 4 shows the correspondence between the navigation nodes and the data tables for the subtree of administrative institution. Black arrows show the corresponding join attributes of all tables. Notice the beneficial effect of the joined view adm_inst: any dimensional node irrespective of its depth can be joined with the fact table directly via this view, without having to traverse the navigation tree. Materialization of the joined views may be considered for improving the query execution time due to spared intra-dimensional joins.

Information about the structure of the data cubes, necessary for generating the data browser and mapping navigation events to database queries, is stored as metadata. The metadata has been modeled in accordance with the specification of the Common Warehouse Metamodel (CWM) [Obj06]. In the recent years, CWM has evolved into a widely adopted standard for metadata interchange in the data warehouse environment. OLAP package of this model provides all necessary concepts (e.g., multiple hierarchy, generalization relationship, etc.) for describing data cubes as navigation objects in an explorative interface.
4 Visualization of OLAP Queries

Confronted with evaluation of thousands of tuples returned by an OLAP query, users tend to focus on a few outstanding values and discard the remaining data, or run another query to retrieve the next-level aggregates (drill-down or roll-up) w.r.t. the ones obtained previously. While this strategy can be successfully applied for reporting tasks, the exploratory analysis is driven by finding information hidden in the data via ad hoc queries. In such scenarios, abstract visual representations (i.e., overviews) facilitate the comprehension. As data volumes for analysis are huge and a lot of insight is hidden at lower granularity levels, coarsely grained aggregates play rather an auxiliary role in explorative queries. The analyst needs to assess the data at different aggregation levels and compare the aggregates at the same level as well as across levels. As an example, one might want to compare expenditures of departments with each other, but comparison of a small department with a project team of approximately the same size would also make sense.

Multiscale visualization techniques are effective for facilitating the exploration process because they change the visual representation to show the data at different levels of abstraction: at a high level, there are just few coarsely grained values; as the user zooms, the data density decreases, allowing to show more detailed representations of individual data points [STH03]. Preservation of the overview throughout the course of interaction (zooming and panning) becomes crucial as otherwise the user might easily lose the context.

Eventually, scalability of the visualization techniques, i.e. the ability to display ever growing number of items in a limited screen while avoiding clutter and overlap, becomes a challenge [EK02]. This challenge turned display space into a scarce resource, which in turn lead to the development of space-filling visualization techniques that attempt to utilize the whole available display space.

4.1 Hierarchical Visualization Techniques for OLAP

There are many factors to evaluate when identifying visualization techniques adequate for a specific application. Visualization encompasses anything from simple charts to highly unique representations, from static diagrams to complex interactive views. In the analysis tools of the current state of the art, traditional visualization techniques such as charts, plots, or pivot tables, prevail. Their main advantages are simplicity, straightforward interpretability, and familiarity. However, the applicability of these techniques is rather limited. For example, the charts only allow uniform granularity, which makes it impossible to compare aggregates across granularity levels within the same view. Furthermore, previously visualized values get lost each time a drill-down or a roll-up step is performed.

The standard interface for a series of successive disaggregation steps is a pivot table, which nests multiple granularities along its two axes. Evidently, pivot tables are not effective for exploratory analysis because of textual list-based representation of the data. For some query types, the resulting pivot table can be replaced by a visual spreadsheet technique. Consider applying a non-cumulative aggregation function, such as MIN, MAX, or AVG, in
Figure 5: Hierarchical HeatMaps for exploring non-cumulative aggregates

Figure 6: Multiscale Recursive Pattern for (dis-)aggregating along time dimension

which case the aggregates at all levels remain within comparable ranges. This common value range opens up an opportunity to map the values to attribute color and transform the pivot table into a hierarchical HeatMap. Figure 5 contains an example of such HeatMap: bottom-level nodes are represented as cells in an array; higher level nodes are shaped as rectangles spanning the width of their subtrees, similarly to the cells in a pivot table. The visualization is obtained by a 3-step drill-down along the subtree of teaching institution for the measure amount’s maximum value. A logarithmic scale of the colormap cope with the skewness of the resulting distribution; sliders at the colormap’s poles are useful for dynamically adjusting the visualization’s sensitivity to outliers.

Due to the hierarchical characteristics found in many dimensions of data cubes, hierarchical visualization techniques lend themselves to be utilized in the analytical process. In general, the analyst follows a top-down approach starting at the higher aggregation levels, because the limited amount of aggregates makes it easier to retain an overview. Step-wise, he either decides to apply slice-dice or drill-down operations to reduce the subset or to retrieve more details.

Traditionally, node link diagrams are used to represent hierarchical data in a compact yet intuitive way. Given a good spatial embedding of the nodes of such a tree, it is easy to determine connectivity characteristics of individual nodes. In the context of visualizing OLAP data, a technique, called Decomposition Tree, proposed by ProClarity [P06], and their visually enhanced variants [VM06] bridge the gap from spreadsheet or chart views.
Figure 7: Decomposition Tree with space-filling bars

Figure 7: Decomposition Tree with space-filling bars. The connection layouts, based on node-link diagram supports immediate perception of child-parent relationships, however, it is not efficient in terms of display utilization because most of the pixels are wasted as background. Moreover, the nodes at lower levels quickly run out of space for displaying the labeling information. Here, the enclosure, also known as “value-by-area”, visualization techniques offer a space-optimized solution. The most commonly used enclosure techniques are TreeMaps [Shn92] and their variations, employing rectangular node shape, and radial layouts, such as the SolarPlot [Chu98] and InterRing [YWRP03], in which the aggregates are laid out radially, with the most aggregated values at the center and finer granularity levels farther away from the center.

Figure 8 shows the results of querying a data cube containing student enrollments from a university data warehouse, visualized using a SolarPlot. The aggregated measure (number of enrollments) values are mapped to the size of the circle segments in each ring. For examining the distribution of foreign students over departments, the analyst filtered out Germany in the dimension country (German: Land), retrieved the sub-aggregates at subcontinent (Ger.: Subkontinent) and, subsequently, at country level, and, in the final step, added the dimension department (Ger.: Institut) as the outer ring. SolarPlot technique overcomes the problem of nested rectangle layouts (TreeMaps) that allocate progressively less space to the nodes at lower tree levels. This is due to the fact that outer rings intrinsically occupy more space than the inner ones. One drawback is that large proportions of the usually rectangular screen stay unused.
4.2 Temporal and Spatial Visualization Techniques

In the multidimensional data model, spatial and temporal components of the data are treated just like any other dimensions. However, to fully exploit the potential and the richness of spatio-temporal data attributes, the awareness of their specific characteristics has to be incorporated into the data warehouse and the analysis tools on top of it.

The outstanding role of the temporal dimension in OLAP is reflected in the data warehouse definition: “...a subject-oriented, integrated, time-variant, non-volatile collection of data in support of management’s decision making process” [Inm02]. Rivest et al. [RBM01] point out that space is the other of the two analytical components, needed to take full advantage of a data warehouse, and that spatial dimensions, just like temporal ones, should be considered standard for any data warehouse implementation. Indeed, there is an estimation that about 80% of all data stored in corporate databases has a spatial component [Fra92].

The requirements towards a visual interface in handling spatio-temporal data is twofold: 1) to communicate spatial or temporal patterns hidden within the data, and 2) to facilitate query specification by building up a mental image of the data distribution which helps in restricting too generally formulated queries to data regions of interest.

**Example 1.** To visualize the temporal evolution of a certain measure within a data cube, a recursive pattern visualization [AKK95] can be adapted to fit into a calendar view, similar to the calendar-based visualization [VWVS99]. Figure 6 demonstrates the results of analyzing the volumes of email communication along time dimension using the above approach. As the information can be displayed at several granularities without changing position of the underlying data items, this technique has proven to be very scalable. Recursive pattern calendar view can be used as an exploration technique in its own right,
or as a node-level view in a decomposition tree, to combine temporal exploration with dis-aggregation along other dimensions.

Example 2. Spatial attributes, such as address, location coordinates, position, orientation, or size, are a frequently encountered dimension in many data cubes. Geography is often used as an exploration axis due to the fact that other data characteristics (i.e., consumer behavior) may strongly vary in space. However, in straightforward techniques, such as highlighting a map or a cartogram according to the measure’s value, large countries - which do not necessarily represent more data elements than small countries - dominate the visual impression and a lot of display space remains unused.

Distortion approaches are employed to overcome the disadvantages of classical maps. The measure under analysis is used as a distortion parameter as demonstrated in [KMP+05]. In case of a gap-free rectangular distortion, the resulting map degrades to a TreeMap, as in the Hierarchical Network Map technique [MV06]. Figure 9 reveals the geographical distribution of the network traffic at a network gateway. The approach simultaneously displays two measures: one value is mapped to the size of the rectangular areas whereas the other is shown by color. In this example, the size of the country’s or the internet backbone system’s node is proportional to the number of IP addresses they represent, and color shows for the actual traffic volumes registered at each node.

5 Exploratory Framework

Interactive visualization is defined by Colin Ware in [War04] as “a process made up of a number of interlocking feedback loops that fall into three broad classes. At the lowest level is the data manipulation loop, through which objects are selected and moved using the basic skills of eye-hand coordination. ... At an intermediate level is an exploration and navigation loop, through which an analyst finds his or her way in a large visual data space... At the highest level is a problem-solving loop through which the analyst forms hypotheses about the data and refines them through an augmented visualization process.”

The exploratory framework introduced in this work supports the interactive visualization process, described above, by defining an appropriate interface for 1) data navigation, i.e. translating visual navigation steps into database queries, 2) visualization, i.e. mapping the retrieved data into a selected visual format, and 3) interaction, i.e. stepwise manipulation of the generated view to solve the defined analytical task. In section 3 we already described our approach to data navigation. Therefore, the focus in this section is on the last two steps that address the generation and iterative refinement of the visualization.

The previous section contains just a limited selection of visualization techniques from a multitude of potentially applicable ones. The challenge of integrating various layouts into a unified framework is to find a common abstraction interface for them all in order to enable their generation by invoking the same navigation events.

As all supported visualization techniques are based on a hierarchical layout, each visualization is a tree, i.e. a directed acyclic graph with all edges oriented away from the root. Any two nodes of the tree connected by an edge are characterized by parent-child relation-
ship. In the context of disaggregation the tree abstraction can be narrowed to the notion of a decomposition tree: the nodes contain the aggregate values and the edges are labeled by the respective dimensional characteristics of their nodes. Depending on the layout (e.g., in “edge-less” space-filling approaches), the labels can be placed inside the nodes.

Child level aggregates are obtained by decomposing a node or the entire tree level along a dimensional category. At this point, the visualizations can be classified according to their node properties. In plain-value-node techniques, as found in Figures 8 and 5, each tree node refers to a single aggregate value, mapped to node’s area and/or color. Chart-node approaches, as in Figures 7 and 6, allow decomposition inside the node, i.e. a node is composed of all child values of an aggregate, obtained by decomposing the latter along a specified dimension and arranged into a chart or other 2-dimensional view.

Further common elements of the presentation layer can be defined as follows:

**Dimensional axis** corresponds to a dimension category, represented by a node in the navigation hierarchy. Every distinct value of the dimension’s values is interpreted as a coordinate (e.g. axis month, coordinate March). Each level of the tree, and – in case of a chart-node layout – also each node has its own dimensional axis.

**Decomposition axis** is the dimensional axis chosen for the disaggregation step, resulting in populating the chart-nodes’ contents or adding a new level to the tree. **Decomposition schema** is the lattice of all decomposition axes in the visual hierarchy.

**Core axis** is defined for the class of space- and/or time-aware visualization techniques. These approaches treat the respective spatial or temporal dimension differently from any other dimension, for example, in the recursive pattern, the only valid decomposition operation is that along the time axis whereas other dimensions may be only used for filtering.

**Tree split** generates a new level of the tree by dis-aggregating the values in the bottom-
level nodes along the specified decomposition axis. In terms of OLAP operations, dis-
aggregation corresponds either to a drill-down, i.e. descending in the granularity of the
same dimension, or to a nesting, i.e. decomposing along a category from a different dimen-
sion. This separation is important for chart-node techniques, which distinguish between a
primary, an inner, and an outer split:

- **Primary split** is the first decomposition step which generates the subset of aggregates
to populate the chart view of the root node.

- **Inner split** is a drill-down into a category of the primary split. The inner split refines
the granularity within the node’s chart.

- **Outer split** is a decomposition step along any axis that is not a part of the primary
split axis’ dimensional hierarchy. Outer split leaves the dimensional axis of the
nodes’ inner charts unchanged.

The above defined abstractions of the presentation are fundamental for implementing a
uniform approach to generating a visualization and interacting with it. Once a visualiza-
tion technique has been chosen, the view is instantiated with the measure(s) and the ag-
gregation function selected from the cube’s navigation hierarchy. Decomposition steps are
performed by dragging the dimensional nodes from the navigation into the visualization
area. Filtering can be performed in the background by setting the values to hide/display in
the data browser, or directly, by deleting nodes or entire sub-trees from the visualization.

Further interaction techniques, such as zooming, rearranging the layout, swapping axes,
sorting, brushing, re-scaling etc. are supported. However, these features are not specific to
our framework and therefore, we omit further details.

6 Conclusion

In this work, we introduced a framework for interactive hierarchy-driven exploration of
OLAP data, in which the data navigation as well as the visualization approaches are based
on the hierarchical view of the data. Hierarchical structures enjoy growing popularity in visual analysis. Multidimensional data cubes do not only contain hierarchically structured
data but are also explored in a predominantly hierarchical fashion by iteratively refining
the subset of interest and its dimensionality and granularity. We identify two key issues in
supporting powerful visual analysis, namely, (1) data navigation for specifying iterative
queries and (2) minimization of cognitive process, or appropriate presentation.

As the first building-block of the framework, we proposed a schema-based data naviga-
tion which facilitates visual specification of OLAP queries. Unlike traditional direct data
tree navigation, found in many OLAP interfaces, our approach strictly distinguishes be-
tween the schema of a hierarchical dimension and the data that populates it. Representing
dimensions by their schema results in a highly compact navigation, admitting multiple
and generalization hierarchies. The ability to stay at the schema level allows the user to
generate a visualization of a large data set with a small number of drag&drop operations.
In the next step, we identified scalable visualization techniques that are able to cope with the challenge of representing potentially large result sets of OLAP queries. Hierarchical layouts are appreciated for their natural ability to show the data of different granularity in the same view. An explorative query in a data cube can be viewed as a series of disaggregation and filtering steps, therefore the sub-aggregates computed at each step can be displayed in form of a decomposition tree. Special characteristics of time and space dimensions were taken into account as those two dimensions are prominent in the context of data warehouses and full account of their specific requirements is likely to reveal valuable patterns in the course of exploration.

By defining a common abstraction level for mapping the results of user interaction into a visual presentation, we could integrate manifold visualization approaches for solving a variety of exploration and conventional reporting tasks in the data warehouse context.

References


Algorithms for merged indexes

Goetz Graefe

HP Labs ¹
Goetz.Graefe@HP.com

Merged indexes are B-trees that contain multiple traditional indexes and interleave their records based on a common sort order. In relational databases, merged indexes implement “master-detail clustering” of related records, e.g., orders and order details. Thus, merged indexes shift de-normalization from the logical level of tables and rows to the physical level of indexes and records, which is a much more appropriate place for it. For object-oriented applications, clustering can reduce the I/O cost for joining rows in related tables to a fraction compared to traditional indexes, with additional beneficial effects on buffer pool requirements.

Prior research has covered merged indexes without providing much guidance for their implementation. Enabling the design proposed here is a strict separation of B-tree and index into two layers of abstraction. In addition, this paper provides algorithms for (i) concurrency control and recovery including locking of individual keys and of complex objects, for (ii) data definition operations including adding and removing individual indexes in a merged index, for (iii) update operations including bulk insertions and bulk deletions, for (iv) enforcement of relational integrity constraints from uniqueness constraints to foreign key constraints, and for (v) query processing including caching in query execution plans dominated by nested iteration and index navigation.

Within such a merged index, the set of tables, views, and indexes can evolve without restriction. The set of clustering columns can also evolve freely. A relational query processor can search and update index records just as in traditional indexes. With these abilities, the proposed design may finally bring general master-detail clustering and its performance advantages to traditional databases.

1 Introduction

Some database management systems provide master-detail clustering in their on-disk data structures, with the aim of saving I/O operations in online transaction processing, data warehousing, and data mining. Not only can a single I/O operation fetch multiple related records from multiple tables or indexes, but the number of buffer frames required to hold an entire query’s working set also decreases due to master-detail clustering.

1.1 The value of master-detail clustering

In general, master-detail clustering should apply not only to pairs of tables or indexes but to any number of them, such that entire complex objects can be stored, fetched, buffered, and saved together. For example, it is beneficial if orders and their order details can be clustered, but it is even better if an entire customer object including orders, shipments, invoices, payments, etc. can be clustered together. In every query that assembles a com-

¹ Palo Alto, CA. Much of this research was performed while employed at Microsoft.
plex object by joining these tables and indexes, I/O cost is reduced not by percentages but by a factor. As others have observed, “optimization techniques that reduce the number of physical I/Os are generally more effective than those that improve the efficiency in performing the I/Os” [HS 04].

As disk performance improves, access latency improves much more slowly than transfer bandwidth. For example, the I/O size for which access time equals transfer time has grown from 10-30 KB in early relational databases to 200-500 KB today. Thus, today’s databases ought to employ large I/Os [GG 97], and techniques are needed that complement large pages, e.g., interpolation search [G 06] and master-detail clustering.

Due to historical trends of disk access times and disk bandwidth, high-end servers often employ “short-stroking,” which is “the practice of formatting a disk drive such that data is written only to the outer sectors of the disk's platters. In I/O-intensive environments, this increases performance, as it reduces the time spent by the drive actuator seeking sectors on a platter. However, short stroking also wastes a substantial portion of the disk drive's capacity” [S 05]. Master-detail clustering can cut the number of I/O operations, because related records from multiple indexes and tables can be fetched with a single I/O operation. For example, if many applications access orders and order details almost always together, master-detail clustering can cut I/O in half and only half as many disk arms are needed. Thus, each disk can be loaded with twice as much data and costs for disks, disk controllers, etc. are cut in half. Such cost savings make merged indexes attractive for high-end applications. Cost savings of merged indexes may rival those of compression in data warehouses [PP 03].

1.2 Master-detail clustering in B-trees

In a database management system that relies on B-tree indexes as the main indexing mechanism, as most commercial systems do, an appropriate implementation strategy for master-detail clustering is to map multiple indexes to a single B-tree. If, in the earlier example, each such index has the customer identifier as the leading key column, all records contributing to a customer object are naturally clustered together within the B-tree. As orders and their order details are often accessed by the order number, it seems beneficial to merge the index on orders.orderkey and the index on orderdetails.orderkey into a single B-tree with orderkey as the leading column.

Figure 1 shows the sort of records within such a B-tree. The sort order alone keeps related records co-located; no additional pointers or references between records are needed. In addition to performance, B-trees provide effective and efficient space management. If all data about multiple customers fit on a single page, they will share a leaf page. If a
A single customer object requires multiple pages, neighboring B-tree leaves will be used. Most B-tree implementations attempt to allocate neighboring leaves contiguously on disk such that all data about a customer can be fetched with a single disk access.

In its most limited form, master-detail clustering combines two non-clustered indexes, e.g., associating two lists of row identifiers with each key value. Alternatively, master-detail clustering may merge two clustered indexes but not admit any non-clustered indexes. The design pursued here accommodates any combination of clustered and non-clustered indexes in a single B-tree, thus enabling clustering of entire complex objects.

In merged indexes, search keys and clustering keys may differ among the participating individual indexes, i.e., participating indexes may differ not only in their overall record structure and contents but also in their keys. The proposed design permits adding and removing individual indexes from a merged B-tree index at any time, with no change in search efficiency. While the performance effects of merged indexes are similar to those of traditional master-detail clustering, this flexibility and thus power for the database administrator distinguishes merged indexes from prior designs for master-detail clustering.

For operations on individual keys and records, including both queries and updates, concurrency control and recovery are no more complex than in traditional indexes, and the different keys do not affect search performance. Even bulk operations such as data load are as fast as in databases and tables using traditional indexes.

1.3 Purpose and scope

The purpose of this research effort is to describe a very flexible yet simple design for merged indexes. The goals outlined above can be achieved with techniques known today, primarily by combining existing data structures and algorithms in new ways. Thus, this research points more often to new combinations of old techniques than to new fundamental techniques, which reduces the implementation effort for merged indexes.

It may be worth pointing out that the purpose here is not to recommend that any and all indexes should participate in merged indexes. If it is important to scan a specific index as fast as possible, interleaving its records with those of another index is counter-productive. Merged indexes are an additional capability available to database administrators during physical database design. Careful index tuning remains a necessity for high performance and high scalability, whether based on manual tuning or a software tool.

The remainder of this paper, after reviewing related work, focuses on data structures, concurrency control and recovery, index creation and maintenance, and query processing techniques, followed by a summary and some conclusions of this research.

2 Related work

Hierarchical and network database management systems have long clustered records of different types by “set membership,” and their storage engines have supported such clustering similarly for a long time. Merged indexes are different as they cluster records by key values rather than record-to-record pointers, and thus merged indexes fit more readily into the architectures of modern database management systems. They are a versatile yet simple way of bringing master-detail clustering to relational database management systems and data warehouses.
Härder’s notion of a “combined image” permits non-clustered indexes with equal search key to share a B-tree [H 78]. Figure 2 illustrates a non-clustered index for three tables. The search key is fixed for the B-tree, as is the set of indexes it contains.

<table>
<thead>
<tr>
<th>Key</th>
<th>1</th>
<th>3</th>
<th>2</th>
<th>RID</th>
<th>RID</th>
<th>RID</th>
<th>RID</th>
<th>RID</th>
<th>RID</th>
</tr>
</thead>
</table>

Counters

Table 1  Table 2  Table 3

Figure 2. Combined non-clustered index.

In comparison, a merged B-tree permits both clustered and non-clustered indexes. Merged indexes permits the set of indexes to change dynamically, and it permits indexes that agree on only some of their key columns. If they agree on none of their key columns, the merged index degenerates into a “concatenated index.”

Härder and Reinert explored the advantages of such indexes for verification of foreign key constraints in relational databases [HR 96]. These advantages included efficient verification of foreign key constraints as well as efficient join operations, both merge joins for large results and index nested loops joins for small results. In index nested loops join, locality benefits affect disk pages loaded in the buffer pool and, in the ideal case, even the appropriate cache lines in the CPU’s data cache.

Valduriez renewed this idea in join indices, which are “prejoined relations … stored separately from the operand relations”. It “is a binary relation” that “only contains pairs of surrogates”. Valduriez also proposed a “multi-relation clustering scheme used in combination with join indices” without much elaboration (all quotes from [V 87]).

In comparison, merged indexes support joins and foreign key constraints more generally than combined images and join indices because the set of key columns might vary among the individual indexes. For example, a single merged index may support joins and foreign key verification between customers and orders as well as between orders and order details.

Oracle’s main database product permits multiple tables clustered on the same set of columns, up to 32 tables and up to 16 columns. Records are co-located using a B-tree key or a hash function. Each value of the cluster key is stored only once. Tables can be added and removed, but the cluster key is fixed for all tables and indexes. For example, a single cluster cannot support efficient search of orders by customer identifier and of order details by customer identifier and order number. Thus, Oracle’s design will rarely permit clustering more than two levels of components within a complex object.

In comparison, merged indexes conceptually store each key column in each record, relying on prefix truncation [BU 77] and other compression techniques to save storage space. However, merged indexes support much more flexibility with respect to clustering keys, search keys, complex objects, and future additions to an existing merged index.

There are three dimensions useful to classify merged indexes or equivalent storage structures. First, the contents may be pointers only as in Figure 2, they may be full rows only as in Oracle’s design, or they may be a mixture of clustered and non-clustered indexes. Second, the set of individual indexes within a merged index may be fixed for the entire lifetime of the merged index, it may permit removal only, or it might permit free addition and removal. Third, clustering may require equal keys as in Härder’s and Oracle’s designs, it may support arbitrary keys as long as those are declared when the merged index is first created, or it may permit declaration of additional keys while the merged
The proposed design for merged indexes supports the most general choices with minimal impact on efficiency.

Note that merged indexes are very different from "star indexes" in IBM’s Red Brick product [F 94]. A star index is a B-tree index on a single fact table within a relational data warehouse, with uniform record format throughout the index and with some of the foreign key values replaced by record identifiers of the dimension tables.

3 Data structures

In order to simplify design and implementation of merged indexes, a crucial first step is to separate implementation of the B-tree structure from its contents. One technique is to employ normalized keys, such that the B-tree structure manages only binary records and binary keys. Thus, all comparison operations compare keys using binary comparisons using a simple method such as “memcmp()” or even an equivalent hardware instruction. This implementation of B-trees also simplifies implementation of B-tree optimizations such as prefix and suffix truncation [BU 77], order-preserving compression, and many optimizations for effective use of CPU caches [GL 01, L 01].

The remaining crucial issue, therefore, is to map index keys with multiple columns, collation sequences, etc. to binary B-tree keys. For traditional indexes with only a single logical index in each B-tree, fairly simple techniques suffice: unsigned integers are immediately usable (perhaps after reversing the byte order [L 01]), signed integers require toggling the sign bit, a NULL indicator bit must precede each value, variable-length strings require a termination symbol, multiple columns can be simply concatenated, etc.

3.1 Key format

Mapping multi-column keys to binary strings is a bit more complex in merged indexes than in traditional indexes, however, in particular if adding and removing any index at any time is desired and if individual indexes may have different key columns.

<table>
<thead>
<tr>
<th>FIELD VALUE</th>
<th>FIELD TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Customer identifier”</td>
<td>Domain tag</td>
</tr>
<tr>
<td>123</td>
<td>Data value</td>
</tr>
<tr>
<td>“Order number”</td>
<td>Domain tag</td>
</tr>
<tr>
<td>4711</td>
<td>Data value</td>
</tr>
<tr>
<td>“Index identifier”</td>
<td>Domain tag</td>
</tr>
<tr>
<td>“Orders.orderkey”</td>
<td>Identifier value</td>
</tr>
<tr>
<td>“2006/12/20”</td>
<td>Data value</td>
</tr>
<tr>
<td>“Urgent”</td>
<td>Data value</td>
</tr>
<tr>
<td>…</td>
<td>Data values</td>
</tr>
</tbody>
</table>

Figure 3. B-tree record in a merged index.

Merged indexes permit growing and shrinking the set of individual indexes merged into one B-tree. Thus, it is essential to design a flexible mapping from keys in the index to byte strings in the B-tree. A tag that indicates a key column’s domain and precedes the actual key fields, as shown in Figure 3, can easily achieve this.

In practice, different than illustrated in Figure 3, a domain tag will be a small number, not a string. It is possible to coalesce the domain tag with the Null indicator (omitted in
Figure 3) such that the desired sort order is achieved yet actual values are stored on byte boundaries. Similarly, the index identifier will be a number rather than a string.

While it is possible that a tag indicates merely the key’s type, e.g., “32-bit unsigned integer,” it is desirable to indicate a domain or a distinct type, e.g., “order number” or “invoice number.” If tags merely indicate types, orders and invoices may be interleaved in the B-tree pages for this example. If domain tags are used, orders and invoices naturally separate into two groups, yet order details naturally cluster with the appropriate order records based on equal domain tag and order numbers.

Domain tags are not required for all fields in a B-tree record. They are needed only for key columns, and more specifically only for those leading key columns needed for clustering within the merged index. Following these leading key columns is a special tag and the identifier of the individual index to which the record belongs. For example, in Figure 3, there are only 2 domain tags for key values plus the index identifier. If there never will be any need to cluster on the line numbers in order details, only leading key fields up to order number require the domain tag. Thus, the per-record storage overhead for merged indexes is minimal and may indeed be hidden in the alignment of fields to word boundaries for fast in-memory processing.

All fields following the index identifier store only the field but no domain tag. In a clustered index, one may expect that a small minority of fields require a domain tag. The columns following the index identifier may define sort order and thus contribute to efficient searches; they just do not permit finer clustering. In Figure 3, for example, the date value might be part of the sort order and the search key, but it does not require a domain tag because the index identifier precedes it.

Incidentally, if the index identifier is made the first column, records belonging to that individual index will be clustered within the merged index, with no interleaving records from other indexes. The effect is that this index is concatenated with other indexes within the merged index. With respect to storage space, since the index identifier is common to all records in many pages, prefix truncation eliminates it from all records in most pages, both in leaves and in upper B-tree nodes.

3.2 Page format

In general, a traditional page format using an indirection vector to manage variable-length records [L01]. In order to balance the increased key size due to domain tags, key prefixes common across an entire page are truncated and stored only once per page, and that suffixes are truncated from separator keys when they are posted in a parent during a leaf split [BU 77].

For maximal prefix and suffix truncation, split operations do not necessarily split at the center and instead split only near that point, e.g., such that each resulting leaf will be between 40% and 60% full. During index creation and bulk insertion, each leaf might be between 75% and 95% full if the desired fill factor is 85%. Within the range of possible split points, the shortest possible separator is chosen [BU 77].

For some operations, notably initial B-tree creation and bulk load operations, partitioned B-trees offer substantial advantages, as described in detail elsewhere [G 03]. Partitioned B-trees require an artificial leading key column, which indicates the partition in a B-tree index or the run number if a B-tree index is used to store runs in an external merge
sort. The partition number is likely to be constant within all nodes other than the B-tree root and prefix truncation will therefore hide it in practically all B-tree leaves and nodes.

### 3.3 Unique indexes

In order to preempt some possible misconceptions about merged indexes and uniqueness constraints including primary key constraints, it may be useful to point out that these concepts are in fact orthogonal. While it is true that two successive records from the orders table might not be next to each other in a merged index even if order number is the leading key column, insertion of a duplicate key into a unique index will still fail because it will find its intended insertion point already occupied.

During index construction when a uniqueness constraint is first declared, it is desirable to shift verification of the uniqueness constraint from the insertion logic to the sort operation. If no sort operation is required because an appropriate sorted source exists, an operation very similar to stream aggregation, inserted between the sorted source and the index insertion logic, can verify that the future index is indeed free of duplicate keys.

### 3.4 Indexes on materialized views

For the storage layer in a database management system, there is little difference between indexes on standard tables and indexes on materialized views. Thus, it is perfectly possible that a merged index combines individual indexes from multiple tables and multiple views. For example, in order to keep with each order its total value, i.e., the product of quantity and price summed up over all order details, a materialized view with a “group by” clause on order number can be indexed on its primary key and merged with other indexes on order number, notably indexes on the orders and order details tables.

### 3.5 Computed columns

A small difficulty arises if all components of a complex object are to be clustered with the object’s root yet some sub-components are “strong entities” with their own globally unique identifiers. A typical example is a customer object with orders and order details. The B-tree search key that permits clustering all these row and record types is the customer identifier. An appropriate column exists in the orders table as a foreign key, but it does not exist in the table of order details because orders have a globally unique identifier. Thus, it may seem impossible to cluster order details with customers and orders.

<table>
<thead>
<tr>
<th>COLUMN</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrderKey</td>
<td>Int foreign key references Orders</td>
</tr>
<tr>
<td>LineNo</td>
<td>Int</td>
</tr>
<tr>
<td>CustKey</td>
<td>Select CustKey from Orders where ...</td>
</tr>
<tr>
<td>Quantity</td>
<td>Numeric</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>Primary key (OrderKey, LineNo)</td>
</tr>
</tbody>
</table>

Figure 4. Table definition for Order Details.

The solution, illustrated in Figure 4, is to propagate the customer identifier to the table of order details using techniques for computed columns and materialized views. A computed column is declared in the table of order details, e.g., as “select custkey from orders
where orders.orderkey = orderkey.” This column is indeed stored in indexes for the table. Like other computed columns, this column cannot be updated directly by user commands. Instead, it is recomputed automatically whenever one of the columns orders.orderkey, orders.custkey, or order details.orderkey is modified, quite similar to cascading actions for foreign key constraints. In a sense, the table of order details has become a materialized view joining orders and the traditional table for order details.

This solution hints at further considerations for master-detail clustering and merged indexes. Relational databases would generally benefit from a separation of table definition and storage definition, because redundancy among indexes for tables and for views could be reduced. Among possible approaches, concepts explored for GMAP [TSI 96] could prove useful, in particular if they are expressed using a SQL dialect.

3.6 Hash indexes and hash clustering

A simpler case is a column computed from values within the same row such as a hash function applied to one or more columns. A B-tree index on the result of a hash function is practically equivalent to a hash index, in particular if interior nodes of the B-tree can be as large as the directory of a hash index and if interpolation search is used within each such large node.

Master-detail clustering based on hash functions can be supported more easily within such a B-tree than in traditional hash indexes. For example, index growth or a non-uniform distribution of hash values are accommodated by splitting B-tree leaves.

In such a B-tree, entries with equal hash value can be organized using additional sort columns. For example, customer and order rows may be hashed on customerkey, with the result serving as the leading B-tree key. The columns customerkey, orderkey, etc. may be additional keys in the B-tree. This index is an unsorted hash index as well as a sorted B-tree index, with an overall sort order that the query optimizer can exploit in grouping, merge join and index nested loops join.

Using this design, very little code beyond standard B-tree code is required to augment a database engine with hash indexes as well as master-detail clustering on hash values. For example, existing code can be used for concurrency control, index creation, data load, defragmentation, consistency checking, etc. I/O performance in this design is similar to traditional hash indexes if non-leaf layers can be retained in the buffer pool. CPU performance is similar if B-tree nodes can be very large, if the hash value distribution is approximately uniform, and if interpolation search is used within each node.

3.7 Summary of data structures

In summary, the separation of B-tree implementation and contents permits exploiting performance enhancements such as prefix and suffix truncation as well as adaptation and generalization of master-detail clustering, which previously has been used in pre-relational database management systems and only in limited forms in modern relational database management systems. Relatively simple data structures and a fairly straightforward mapping permit powerful clustering of entire complex objects without violating relational theory, relational languages, or their design philosophy. Expected performance advantages, in particular for online transaction processing applications on high-end servers, can translate into substantial cost advantages.
4 Concurrency control and recovery

Concurrency control and recovery have traditionally been concerns managed in the lower levels of the storage engine, and merged indexes do not require a change of that basic design.

4.1 Key range locking

Traditional locking of key values and key ranges [GLP 75, L 93, M 90, ML 92] applies to merged indexes in the same way as to traditional B-tree indexes. The same trade-offs and optimizations apply. Of course, the key values being locked must be the keys as outlined in Figure 3, i.e., they must include domain tags and the index identifier. If locking is implemented in the software layer that interprets B-tree records as binary strings, this requirement is readily met.

In merged indexes, key range locking might lock a record from one individual index in order to insert a record from another individual index. For example, if orders and order details are ordered in the most obvious way, adding a detail record might lock the next order, assuming that “next key locking” is used for range locking [L 93, M 90]. In a design that makes details follow the header, e.g., orders and invoices follow customers and order details follow individual orders, there might be fewer surprising concurrency control conflicts if the prior key is locked for a range rather than the next key.

Maybe more importantly, lock modes should be designed such that a key and a range together can be intention-locked, in the sense of multi-granularity locking [GLP 75], and the gap between two keys and an individual key value each can be locked separately [G 07]. For performance reasons, it makes sense to implement this not as two levels of lockable resources (comparable to the standard example with pages and files) but as a single resource with many possible lock modes [G 07, L 93].

Another interesting case is locking individual keys when dropping an individual index within a merged index, i.e., when bulk-deleting many keys from the B-tree. It is quite possible that many neighboring keys belong to other individual indexes and might be locked by concurrent transactions. If a deletion only updates the record’s ghost bit, marking the record invalid and non-existent for subsequent queries, the deletion operation only requires key value locks that do not interfere with these other transactions. Asynchronous ghost cleanup will eventually erase those records when their space is needed.

4.2 Locking complex objects

If locking and concurrency control are left to the lowest levels of the storage engine, i.e., the level that manages B-tree keys and records as binary strings without regard to individual indexes within a merged index, it might seem impossible to lock entire complex objects with a single lock or with only a few locks. Strictly speaking, this is true, but a combination of other design choices effectively results in locks on entire complex objects.

Splitting B-tree leaves not at the center but at a point near the center chosen to maximize the effects of prefix truncation and suffix truncation also promotes leaf boundaries that coincide with boundaries of complex objects. For example, if a leaf split occurs between two records that both belong to an index on order details, the separator key in the
parent must include not only an order number but also a line number. If, on the other hand, the split point is chosen to fall between two orders, the separator key can be shorter. Moreover, if the size of an order and its details roughly coincides with the size of a leaf page, this order like will be stored in a dedicated leaf page, with the obvious beneficial effect on prefix truncation. Optimization of suffix truncation automatically achieves this effect on locking.

In order to achieve object-level locking, the locking strategy should require locks not only in B-tree leaves but also on separator keys in the leaves’ parents [G07]. Some commercial database management systems lock both leaf pages and individual keys within; the proposed method acquires the same number of locks but instead of leaf pages it locks key values and key ranges in the leaves’ parent nodes.

If objects are small such that multiple objects fit in a leaf, this design locks multiple objects at a time. If an object is very large, it might require multiple locks at the parent level. While this is not strictly equal to locking complex objects, it achieves its main effects, namely locking related component records with very few locks rather than locking them one at a time, and it achieves this effect with fairly traditional locking mechanisms.

The locking discipline uses standard hierarchical locking with standard intention locks. If most complex objects are very large, locking might start in the leaves’ grandparent nodes rather than in the parents. Note also the standard techniques for lock escalation and de-escalation apply [J91, LC89], such that a transaction may lock an entire leaf’s key range by locking a key in the parent but reducing that lock to an intention lock after acquiring a lock on a key in the leaf in the case of contention. Interestingly, initial acquisition of a large lock and subsequent lock de-escalation can improve performance because the smaller locks can be acquired without concern about conflicts and thus they can be inserted into the lock manager’s hash table without preceding search [GL92].

### 4.3 Locking summary records

Summary records, e.g., records representing or indexing rows in a materialized view that summarize the total value of each order by summing up order details, can be concurrency bottlenecks and might therefore warrant special lock types. Escrow locks [O86] can be employed and can be augmented with intent-to-escrow lock modes for use in upper B-tree levels [GZ04]. Traditional shared and exclusive locks can be combined with escrow locks within the same B-tree, such that rows from tables and materialized and indexed views can be co-located in the same merged index and each locked appropriately for its use. A typical example is a merged index with an index from the table orders and an index on a view that summarizes order details for each order.

If it makes sense for the data in the merged index, locking in escrow mode even at the parent level is possible, in particular if lock de-escalation is available. All intention locks are compatible with all other intention locks [K83]; therefore, two transactions might concurrently hold, for example, intent-to-share and intent-to-escrow locks for the same complex object.

### 4.4 Recovery considerations

The concurrency control mechanisms above are very standard with respect to their needs for write-ahead logging and recovery. Escrow locks for summary records are aug-
mented with commit-time-only exclusive locks that follow the normal logging and recovery regimes for traditional updates under traditional exclusive locks [GZ 04]. Thus, advanced optimizations are possible such as lock acquisition during the log analysis phase of crash recovery with transaction processing during redo and undo phases. Other possible optimizations include database mirroring with automatic failover.

The essence of the techniques proposed here is to employ entirely ordinary B-tree techniques for B-trees managing strictly binary keys. The meaning of these binary keys is immaterial, including the mapping from multiple tables, views, indexes, column types, column sequence, sort order, collation sequence, etc. The advanced locking proposals above, e.g., locking key ranges in the parent and grandparent level of a B-tree, only affect the concurrency among multiple transactions; they do not affect their logging or recovery including database mirroring.

One particular logging optimization may be worth calling out. A standard use of system transactions is to remove ghost records left behind by user transactions during deletions. Separation of logical deletion (turning a valid record into a ghost) and physical removal (reclaiming the record’s space) serves three purposes, namely simplified rollback for the user transaction if required, increased concurrency during the user transaction (locking a single key value rather than a range), and reduced overall log volume. If the system transaction performing the ghost clean-up can capture transaction start, record deletion, and transaction commit in a single log record, there is never any need to log undo information, i.e., the deleted record’s contents.

For clustered indexes, this may represent a substantial reduction in log volume. In merged indexes, this also applies to records left behind while dropping an individual index. Thus, it is possible to drop an individual index without logging its contents, albeit with many subsequent system transactions and one log record for each of them.

4.5 Summary of concurrency control and recovery

The value of the chosen approach to concurrency control and recovery for merged indexes and complex objects is that it relies entirely on well-understood mechanisms, many of them already implemented in most commercial database systems. The additional facilities, e.g., range locking at the parent level within a B-tree or escrow locks for summary records, are not strictly required to make merged B-trees work. If implemented, they benefit merged indexes and traditional indexes alike. The reliance on traditional B-trees for master-detail clustering and complex-object clustering reduces the variety of on-disk data structures and thus the required effort for development and testing of improvements and of new capabilities.

5 Index creation and maintenance

Similar to concurrency control and recovery for merged indexes, which are characterized primarily by using existing techniques and benefiting from their fortuitous interactions, index creation for merged indexes requires combining mostly traditional techniques with a few recent innovations. This section covers index creation, update operation including bulk updates, and finally algorithms for adding and removing individual indexes in existing, populated merged indexes.
5.1 Index creation

Initial creation of a merged index is fairly simple if all participating indexes are still empty – it is merely a metadata operation to define the appropriate catalog entries. If some or all of the participating tables and views are non-empty, there are multiple algorithms for creating the initial merged index and its B-tree.

The obvious strategy employs a full outer join of all individual indexes. Specifically, after the data for each individual index is scanned and sorted as appropriate for a traditional single-index B-tree, a full outer join using a merge join algorithm combines them such that the new B-tree can be loaded with the desired fill factor, on-disk layout, etc.

A less obvious strategy employs techniques for bulk insertion and incremental index operations, to be discussed shortly.

5.2 Insertion, update, and deletion

Ordinary update operations work on merged indexes just as they do for traditional, single-table indexes. The only difference, of course, is that the search key must be augmented with domain tags as discussed earlier. Duplicate detection in unique indexes relies on finding a key already present.

Large updates can be optimized using index-by-index updates with sorted streams of change items instead of row-by-row updates with random search operations in each non-clustered index of the affected table or view [ABC 01]. Split-sort-collapse sequences for updates of keys in unique non-clustered indexes with the possibility of false uniqueness violations also apply to unique indexes within merged indexes. The only difference, of course, is that the cost function for alternative update strategies must reflect the size of the merged index rather than the individual single-table index.

More interesting is a complex update plan with verification of foreign key constraints and perhaps even cascading. Given that merged indexes are ideal precisely for data with foreign key constraints, optimal performance and locality require that verification of constraints be deeply integrated with index maintenance. In that case, the traditional phase separation between index maintenance and constraint verification is not guaranteed, and interactions are possible similar to the Halloween problem [GLS 93, M 97].

For the execution of such integrated operations, it is an open question whether pipelining change items between update operations is sufficient or a new integrated update operation is required. In either case, if update operations employ read-ahead or prefetch in the indexes they maintain, redundant prefetch requests should be avoided while updating of merged indexes.

5.3 Bulk insertion and bulk deletion

Traditionally, bulk operations such as data load created a dilemma for database administrators. The choice was either to update indexes slowly and incrementally or to drop indexes and recreate them after completion of the load operation. This dilemma applies to merged indexes in an even worse way because dropping an individual index means removing its many records from the merged index, and re-creating an individual index means inserting its many records into the merged index.
There is, however, a third alternative, based on partitioned B-trees [G 03]. Note that traditional partitioning is a metadata operation, such that changes in the partitioning scheme force expensive recompilation of cached query execution plans. Partitioned B-trees define partitions within the B-tree using an artificial leading key column. Partitions appear and disappear by insertion and deletion of records or by updates of values in the artificial leading key column.

A bulk insertion exploiting partitioned B-trees proceeds in two steps. First, new records are inserted into a new partition, i.e., old and new data do not interleave. Second, partitions are merged such that only a single partition remains.

After the first step, data is immediately searchable, as it is correctly present in the B-tree. The B-tree becomes optimized for search performance only during the second step. The algorithm logic employed is very similar to a merge step in an external merge sort. The second step is not a single large transaction, however; instead, many small system transactions ensure high availability of the index data by frequent transaction commits.

Since a merge step is practically equally efficient whether there is only two merge input or many, it is not required that the initial insertion in the first step inserts into only a single partition. Instead, it can use multiple partitions, e.g., using a partition per memory load such that the first step only generates initial sort runs but does not merge any runs. If so, bulk insertion can very efficiently insert new data into multiple indexes, using only in-memory sort for each appropriate index sort order.

The main concern about this procedure is that both steps require logging. If log bandwidth and space are available, the dilemma of choosing between inserting very slowly and wastefully dropping existing indexes is resolved. Very preliminary design work indicates, however, that non-logged merge operations are possible, based on “force” buffer pool management during transaction commit [HR 83] or “careful replacement” of pages in the buffer pool and on disk [GR 93, LT 95].

Bulk deletion proceeds in the opposite order. First, records to be deleted are moved to a separate partition, a few records at a time using a system transaction that can commit without flushing the log. All data remain searchable during this period, even if search performance is not optimal. Second, this entire partition is deleted in a single large and efficient user transaction.

The separation of insertion into a B-tree and optimization of the B-tree structure also applies to bulk insertion into merged indexes. For the B-tree structure and its search efficiency, it does not matter whether a partition contains B-tree entries from one or from multiple individual indexes.

### 5.4 Adding and removing an index

Adding a new individual index to an existing, populated merged index is very similar to a bulk insertion and can employ the same data movement. After the required catalog updates, the new data is inserted into one or more new partitions. Once the new data is in the B-tree, they are available for queries and updates. Optimization of the B-tree structure by merging the partitions such that complex objects are properly co-located is a second step that can proceed as system load permits using many small system transactions.

The desirable effect of this two-step procedure is that the actual data insertion does not create contention with other indexes and their data. Instead, the existing partitions serve all concurrent user transactions such that user transactions and data definition do
not affect one another. Only the small system transactions that merge data can interfere
with user transactions, but the system transactions can commit and release locks at any
time because they do not modify the B-tree’s contents, only its representation.

Removing an index can use the opposite sequence, very similar to bulk deletion. First,
all records belonging to the index are moved to one or more separate partitions; second,
this partition is deleted using a large but efficient operation.

It is even possible to add multiple indexes to a merged index in a single operation, or
to merge two merged indexes into one. Similarly, it is possible to split an existing merged
index into two or to remove multiple individual indexes from a merged index in a single
operation. In each case, the generality of the record format is pivotal to making these op-
erations possible, and partitioned B-trees are pivotal to making them efficient.

5.5 Online index operations

An index operation is online if it permits concurrent insertions, deletions, updates in
the indexed table or view. Online index creation can be implemented using two alterna-
tive approaches [MN 92]. First, the index is created without concern for concurrent u-
pdates, which are applied only in a final catch-up step based on a “side file,” e.g., the re-
covery log filtered for updates of the relevant table. Some complexity arises if the index
creation operation is part of a large transaction, possibly with multiple online index op-
erations, because all final catch-up must occur at transaction commit or each index must
be taken offline from its last catch-up operation until transaction commit.

Second, updates can be applied immediately by the concurrent transactions to the fu-
ture index; the principal difficulty is dealing with deletions in the relevant table while the
index is not yet complete and may not contain the index entries for the affected rows. In
this approach, “anti-matter” index entries can be inserted into the future index such that
the index build operation recognizes when to suppress index entries. If the new index is
for a summary view with “group by” or “distinct” operations, anti-matter takes the form
of negative counters rather than a single flag bit [GZ 04].

Merged indexes do not require special considerations for online index operations. Both
the side-file and the no-side-file approach are possible, as is online creation of in-
xes on summary views. The individual indexes within a merged index are not much
different from traditional indexes; the main difference is that their keys and information
fields are mapped to a binary string in a particular way and that they are then stored not in
dedicated B-tree but in a merged index.

5.6 Incremental index operations

Similarly, incremental index creation does not depend on each individual index being
stored in a dedicated B-tree. In incremental index creation, a table of contents captures
the key ranges in the source, e.g., a table’s clustered index, that have already been
scanned and appropriately reflected in a new index. This table of contents starts empty,
leading to a notion of “instant index creation,” and ends up indicating that the entire table
is fully indexed. Frequent commit operations by the index builder permit high concur-
rency with user transactions, which are even permitted to employ the new index if useful
while still incomplete [G 03].
In fact, a user transaction can even invoke system transactions that copy appropriate data from the source to the new index, in order to ensure that the entire user transaction can be satisfied by the new index [G 03a]. This variety of incremental index creation could lead to a new, totally demand-driven paradigm for online index tuning.

Merged indexes can participate in incremental index creation as well as in shrinking an index incrementally if desired. A table of contents is required to capture the current state of the partial index. The table of contents can apply to a single one among the individual indexes, to a group of them (e.g., orders and order details but not invoices and invoice details), or to all individual indexes within the merged index. The mechanisms for these operations are fairly straightforward, whereas appropriate tuning policies and software tools remain as research opportunities.

Related to incremental index operations is the concept of in-memory-only indexes, which act effectively as caches. Imagine a buffer manager that, rather than evicting an index page from the buffer pool, invokes appropriate mechanisms that remove the page from the index and ensure appropriate modifications in the index’s table of contents. While complex to realize due to the multiple software layers involved, a mechanism of this type might be particularly useful when applied to merged indexes.

5.7 Summary of index maintenance

In summary, index maintenance for merged indexes is not very different from index maintenance for traditional indexes. However, bulk operations take on additional importance because they support not only traditional bulk operations but also changes in the definition of the merged index, i.e., adding and removing individual indexes. Partitioned B-trees may be an important technique, because they not only speed up these operations but also permit changing the set of indexes with minimal concurrency control contention.

6 Query processing

Merged indexes create unique opportunities for efficient query processing, because they store data both sorted and clustered. The strongest effect is, of course, on joins of individual indexes in the same merged index.

What is suffering, however, is scanning just one of the individual indexes. Thus, merged indexes may not work well with some data mining algorithms or with hash join. More generally, if a merged index contains more individual indexes than needed for a given query, scan and join performance suffer. However, there are some reasonably efficient techniques to obtain just the records of one individual index.

Merged indexes can be very useful for nested query execution plans, in particular for caching results of previous invocations. These, too, will be discussed below.

6.1 Relational joins

Merged indexes can improve the performance of both merge join and index nested loops join. In fact, it has been observed previously that merge join and index nested loops join are very similar algorithms in many ways [GLS 93]. Beneficial effects accrue both for disk I/O and for cache faults.

Merge join performance should improve for multiple reasons, including less random I/O and fewer required pages in the buffer pool.
The performance of index nested loops join should improve perhaps even more due to improved locality and the resulting buffer effects. For any one value in the join, e.g., one orderkey in the join of orders and orderdetails, the number of required I/O operations, buffer pool frames, etc. is cut in half. If the merged index combines or replaces more than two traditional indexes, the savings can be even higher.

The greatest difference between merge join and index nested loops join is that the former joins entire streams or indexes whereas the latter is most appropriate when data about a small set of individual objects is needed, e.g., a single invoice or a single customer. Thus, online transaction processing for line-of-business applications will likely experience the most immediate and the most dramatic benefits of merged indexes.

6.2 Single-index scan

Scanning an individual index within a merged index is rather like scanning a traditional multi-column B-tree with a predicate on the second column but none on the first column. The required techniques have been described previously as multi-dimensional access method [LJB 95].

Specifically, such a scan interleaves what could be understood as two scans. The first among those enumerates the distinct values of the leading column. If, for example, a traditional B-tree’s search columns are named x, y, and z, the first scan answers the query “select distinct x from … order by x.” The first probe into the B-tree simply follows the B-tree edge from the root to the left-most leaf, equivalent to “select min (x) from …” Each subsequent probe attempts to find the next-largest value of x, equivalent to adding the clause “where x > x_0” with the most recent result value for column x replacing x_0. The number of B-tree probes is equal to the number of distinct values in column x plus 1 at the right edge of the B-tree.

The second among these logical scans combines the user’s predicate on column y with the term “where x = x_0.” Thus, the user’s predicate is employed and the B-tree is probed once again for each distinct value of column x.

If the leading column x is an integer, an optimization uses “where x ≥ x_0+1” rather than “x > x_0.” In fact, this predicate can be combined directly with the user predicate on the next column, cutting the total number of B-tree probes to the number of distinct values of column x plus the number of gaps in the series of distinct value in column x.

If there is a user predicate only on the third key column, say column z, distinct pairs of the first two columns must be enumerated using basically the same technique. Alternatively, one can think of it as one scan enumerating distinct values of column x, one scan enumerating distinct values of column y for each value of x, and a third scan evaluating the predicate on column z. The I/O pattern and cost are equal for these two ways of designing and implementing the algorithm.

In a partitioned B-tree, this technique can be used to search all existing partitions. The artificial leading key column that serves as partition identifier takes on the role of column x in the example above, and the number of B-tree probes is multiplied by the number of partitions plus the number of gaps in the sequence of partition identifiers. If the leading user-defined index key is not restricted, the scan must enumerate pairs of values for the partition identifier and leading user-defined column.

When scanning an individual index within a merged index, these techniques apply both to the partition identifier in a partitioned B-tree and to all other columns preceding
the index identifier in the record format. For example, if a merged index contains indexes on the customer, orders, and order details tables, scanning the B-tree for just the index on the orders table needs to enumerate partition identifiers, customer identifiers, and order numbers. Thus, merged indexes prove their advantage in complex object assembly but not for purposes of data mining and statistics or in combination with hash aggregation and hash join, as mentioned in the introduction.

6.3 Caching in nested iteration

Temporary indexes are a powerful capability in database query processing. After all, materialized and indexed views are optional and thus often temporary. Similarly, in-memory hash tables in hash join and in hash aggregation are temporary indexes, albeit in thread-private memory. Sort operations, e.g., for merge join or stream aggregation, are very similar to B-tree index creation [G 03].

Temporary B-tree indexes and merged indexes are particularly interesting in nested iteration and correlated sub-queries. In those query execution plans, an inner plan has formal parameters, also known as correlation columns. The inner plan is invoked repeatedly with actual parameter values taken from the current outer row. If two outer rows provide the same correlation values, execution of the inner query execution plan can be avoided if the result of the first execution has been cached. Thus, an index is needed that maps correlation columns to result columns.

An inner query’s result may contain any number of rows, including zero rows. An empty result is important to cache, too, because that information permits avoiding re-execution as much as a non-empty result. In fact, even if the actual result is not cached, its size and cost of re-computation can be quite useful, e.g., to order the outer rows in queries for only a few output rows are desired such as “exists” and “top” queries. The cost of re-computation may vary by correlation value for many reasons, e.g., if there is a predicate in the inner query using a less-than comparison with the correlation value.

In order to represent empty results in the cache, it is expedient to create a “control” index that maps outer correlation columns to information about the result, e.g., its cardinality and size, its cost of re-computation if that depends on specific correlation values, and its usage frequency and most recent usage. In other words, if the main temporary index for the result of the inner query is a cache of fetched data somewhat like a traditional buffer pool, this control index can take the role of the buffer descriptors used to manage the buffer pool contents. The disadvantage of two indexes is, of course, that both indexes may incur I/O during query processing.

Merged indexes can eliminate the overhead of using two separate indexes by ensuring that a lookup in the control index loads the query results into the buffer pool if they are already available. Thus, a merged index provides all the power of representing a query result and its metadata without the need for inappropriate record formats.

6.4 Query optimization

Query optimization considers both logical aspects (tables, rows, columns, cardinality estimation, equivalence transformations) and physical aspects (indexes, records, fields, cost calculation, algorithm choices). Merged indexes offer an additional choice in physical database design and therefore do not affect any of the issues on the logical level.
For cost calculation, merged indexes affect data volume in scans and buffer pool effectiveness in joins. The buffer pool effects are quite similar to those of nested iteration for sub-queries and of index nested loops join, with similar cost formulas.

6.5 Summary of query processing

To recap this section on query processing, merged indexes have their place in join processing, in particular in the context of complex object assembly. They also permit single-index scans but only with limited efficiency. Thus, merged indexes are not a cure-all for performance problems but they are a valuable option during physical database design. In fact, their efficiency during retrieval of complex objects ought to reshape discussions about de-normalization for database performance. Merged indexes also add new power to nested iteration, in particular caching results of inner queries for duplicate values in the outer correlation columns.

7 Summary and conclusions

In summary, merged indexes, an implementation technique at the physical level of indexes and records, achieves the performance advantages that often are pursued with de-normalization at the logical level of tables and rows. These performance advantages are achieved without exposing data and applications to the well-known dangers of non-normalized database schemas.

Many of the beneficial effects in the proposed design are based on four techniques. First, the core B-tree implementation manages only binary records, leaving it to the next layer in the software stack within a database management system to map individual indexes and their key columns to appropriate binary strings. Second, each record includes an index identifier as one of its columns such that records from multiple indexes can be interleaved within a single B-tree yet separated when needed. Third, domain tags precede the leading key columns up to and including the index identifier, which permits adding a new individual index to a merged index at any time with immediate and automatic clustering. Fourth, partitioned B-trees permit efficient bulk operations in any B-tree, which in the case of merged indexes includes not only traditional bulk operations but also changes in the set of individual indexes interleaved to form the merged index.

These techniques enable many benefits. Most importantly, clustering of master-detail records and even of complex objects is almost natural. This is true even in traditional relational databases, with tremendous improvements in join costs and buffer requirements. Moreover, locking of entire complex objects is quite readily possible by replacing locks on leaf pages with locks on pages or on key ranges in the leaves’ parent pages.

In relational database management systems, clustering has been neglected to-date due to the perceived difficulties involved. The presented design may change both this perception and the reality. The advantages of merged indexes for performance in online transaction processing and in data warehousing are very compelling. Given that most applications are designed and implemented using object-oriented design, object-oriented programming languages, and object-oriented tools, clusters of related records in merged indexes can match the access behavior of the applications. Each disk read in a merged indexes and each page in the buffer pool can carry more useful information than is possible
with traditional, individual indexes. If the present paper encourages and contributes to the design and to implementations of merged indexes, it has served its purpose.

Acknowledgments

Wey Guy gave very helpful feedback on an earlier draft of this paper.

References

Effective and Efficient Indexing for Large Video Databases

Christian Böhm Peter Kunath Alexey Pryakhin Matthias Schubert
Institute for Informatics
University of Munich
D-80538 Munich, Germany
{boehm,kunath,pryakhin,schubert}@dbs.ifi.lmu.de

Abstract: Content based multimedia retrieval is an important topic in database systems. An emerging and challenging topic in this area is the content based search in video data. A video clip can be considered as a sequence of images or frames. Since this representation is too complex to facilitate efficient video retrieval, a video clip is often summarized by a more concise feature representation. In this paper, we transform a video clip into a set of probabilistic feature vectors (pfvs). In our case, a pfv corresponds to a Gaussian in the feature space of frames. We demonstrate that this representation is well suited for accurate video retrieval. The use of pfvs allows us to calculate confidence values for frames or sets of frames for being contained within a given video in the database. These confidence values can be employed to specify two types of queries. The first type of query retrieves the videos stored in the database which contain a given set of frames with a probability that is larger than a given threshold value. Furthermore, we introduce a probabilistic ranking query retrieving the k database videos which contain the given query set with the highest probabilities. To efficiently process these queries, we introduce query algorithms on set-valued objects. Our solution is based on the Gauss-tree, an index structure for efficiently managing Gaussians in arbitrary vector spaces. Our experimental evaluation demonstrates that sets of probabilistic feature vectors yield a compact and descriptive representation of video clips. Additionally, we show that our new query algorithms outperform competitive approaches when answering the given types of queries on a database of over 900 real world video clips.

1 Introduction

Video clips are an important type of multimedia data. Due to recent technical advances, the amount of video data that is available in digital formats as well as the possibility to access and display such video files has increased enormously. Nowadays, it is possible to view complete movies on mobile phones and MP3 players. Another important aspect is that broadcasting videos over the WWW (e.g. in video podcasts) allows to distribute video data to a large number of people while spending minimum effort and budget.

The enormous amount of video clips and movies that is currently available causes a need for database techniques to manage, store and retrieve video data for various applications. In this paper, we focus on the following scenario: Given a database of movies or video clips, we want to retrieve all movies from the database that are likely to match a given set
of query images. The query images might consist of a continuous image sequence of a scene or might be sampled from the complete movie. For this type of scenario, there are various applications. For example, a company wants to determine if a given video podcast or shared video file contains scenes from any copyright protected movie or video clip. In this scenario, the company would store all of its movies in the database and automatically check if the scenes in the video podcast match any scenes in the database.

Another example is a database of news programs recorded on various days from various tv stations. A user can retrieve all news programs that are likely to contain a given video clip featuring a particular event. Since most news programs use videos which are provided by video news agencies, it is very likely that the news programs dealing with similar topics contain similar news clips. Another application is the detection of commercials in video data recorded from television. In this case, the commercial is the query and the programs are stored in the database. Thus, there are varying applications for this scenario varying from the detection of single scenes to similarity search on complete movies.

From a technical point of view video data consists of a sequence of images (so-called frames) that might be accompanied with some soundtrack. In our approach, we focus on the image part only. To allow similarity search on video clips, each frame is usually represented by a feature vector corresponding to some content based image representation such as color histograms or texture vectors. So-called summarization [ZRHM98, GGM02, CSL99] techniques are used to reduce the enormous number of frames. For summarization, a video is decomposed into shots, i.e. a sequence of frames within a movie showing the same scenario recorded from the same camera position. The images within a shot are usually very similar and thus, the images are usually associated to very similar feature vectors. Therefore, each shot can be summarized by some representative object and only the representative objects are stored in the database. To represent a shot, it is often sufficient
to simply take the centroid or mean vector of all feature vectors within the shot. Newer approaches like [IBW+04] represent shots as Gaussian probability density functions (pdf) where each component $\mu_i$ of the mean vector is complemented by a variance $\sigma_i^2$. We call such feature vectors where each vector component is associated to a variance value probabilistic feature vector (pfv). This type of summarization is usually more accurate because the method additionally considers the variance among the summarized feature values. In our new approach, we condense the given video data even more, by representing all similar frames by one Gaussian regardless of the shot they belong to. To conclude, each movie in the database is represented by a set of probabilistic feature vectors (pfvs) where each Gaussian represents a set of similar frames.

Our work is focused on similarity search and scene detection in movie databases. To pose a query, a user has to provide a video clip that might comprise a scene in the movie or even the complete movie. The query clip can be transformed into a set of frames, corresponding to a set of traditional feature vectors or a set of probabilistic feature vectors. To use probabilistic (rather than traditional) feature vectors for the queries yields advantages as well as disadvantages: extracting a set of frames and determining traditional feature vectors without further summarization might be computationally simpler and less expensive. In contrast, probabilistic feature vectors might represent the information contained in the query in a more concise way. Therefore, we will examine both possibilities.

Furthermore, we develop a method for comparing both types of query representations to objects stored in the database which is based on the likelihood that the query matches the database object. Based on this method, we describe two types of probabilistic queries. The first type is the set-valued probabilistic threshold query retrieving all movies matching the given query frames with a likelihood which is higher than a specified threshold value. The second query type is the set-valued probabilistic ranking query retrieving the top $k$ movies from the database which are most likely query hits.

Although summarization considerably decreases the size of the representation of each database object, query processing still requires to examine every movie description in the database. Therefore, we will introduce algorithms for query processing that are facilitated by the Gauss-tree [BPS06b], an index structure for probabilistic feature vectors. Let us note that our previous work on the Gauss-tree was focused on querying single objects. In this paper, we introduce techniques for querying set-valued objects which is a more complex problem.

Our main contributions in this paper are:

- A compact representation of a video as sets of probabilistic feature vectors and a method for similarity and partial similarity search based on statistics.
- The specification of two new types of probabilistic queries on sets of probabilistic feature vectors.
- Efficient algorithms for processing these new types of queries on sets of probabilistic feature vectors which are based on the Gauss-tree.

The rest of the paper is organized as follows. Section 2 surveys related topics like content
based video retrieval and similarity search using point sets and probabilistic feature vectors. Additionally, the Gauss-tree is introduced as the index structure the query algorithms are based on. In section 3, we will formalize our model and the new types of queries. Section 4 describes the new algorithms for query processing. To demonstrate the quality of our approach to video retrieval and show the superior efficiency of our query algorithms, we provide several experiments on a database of over 900 video clips in section 5. The paper is concluded by section 6 containing a short summary.

2 Related Work

2.1 Video Summarization Techniques.

Since video data consists of large sequences of images or frames, a straightforward feature representation of a movie might contain thousands or even millions of feature vectors. In order to handle such data efficiently, summarization techniques are usually applied to the original data, i.e., the original feature vectors are grouped together and each group is represented by a summarization vector or summarization representative. Then similarity is defined based on these summarizations. Summarizations are usually generated by applying optimization algorithms on feature vectors. They describe a video as a mix of statistical distributions or cluster representatives. The authors of [CSL99] propose an approach for obtaining a compact representation of videos that computes the optimal representatives by minimizing the Hausdorff distance between the original video and its representation. There also exist approaches which apply $k$-medoid or $k$-means clustering for the summarization of video clip content [ZRHM98]. In [GGM02], a summarization technique is presented which describes spatial-temporal areas in a sequence of a few dozen frames by mixtures of Gaussian distributions. The authors of [IBW+04] demonstrated that Gaussian mixture models computed from video shots yield higher retrieval precision compared to keyframe-based models. However, to the best of our knowledge, none of these techniques uses an index structure for the pfvs to accelerate query processing.

2.2 Similarity Search Based on Set-Valued Objects

Set-valued objects are usually compared by complex distance measures like [EM97, RB01] allowing similarity queries. However, selecting a suitable distance measure for a particular application is often quite difficult because there exist many different notions of similarity between two sets of feature vectors. Another problem is the understandability of the derived distances. For complex distance measures and large set-valued objects containing hundreds of instances, it is very difficult to understand why the set-valued objects are similar. Finally, employing the proposed distance measures often yields efficiency problems. Since most of the distance measures for set-valued objects are non-metric, employing index structures is not always possible. Additionally, useful filter steps avoiding
time consuming distance calculations like in [BKK+03] were introduced for a minority of multi-instance distance measures only. To the best of our knowledge there is so far no query algorithm handling sets of probabilistic feature vectors, instead of ordinary set-valued objects.

2.3 Similarity Search Based on Probabilistic Feature Vectors

In [CKP03] a new uncertainty model is introduced and several new types of queries are described that allow the handling of inexact data. [CXP+04] describes two methods for efficiently answering probabilistic threshold queries that are based on the R-Tree [Gut84]. A probabilistic threshold query returns all data objects that are placed in a given query interval with a probability exceeding a specified threshold value. [TCX+05] introduced the U-Tree for indexing uncertain 2D objects. All these approaches do not handle sets of probabilistic feature vectors and do not apply a Bayesian setting. Thus, the mentioned approaches are rather dealing with data objects having an uncertain location. Besides the mentioned methods for indexing spatially uncertain objects, [DYM+05] introduces existential uncertainty. The idea of this approach is that the existence of each data object is uncertain.

2.4 The Gauss-tree

In [BPS06b], the Gauss-tree is introduced which is an index structure for managing large amounts of Gaussian distribution functions. Additionally, [BPS06b] proposed probabilistic identification queries which are based on a Bayesian setting, i.e. the paper deals with the retrieval of the database objects that explain a given query observation with the highest probability. This setting is more similar to the queries described in this paper. However, the queries in [BPS06b] are based on the assumption that there is exactly one individual object explaining the query object. In our setting a very important aspect is that one query video clip might be contained in several database movies. Another major difference to the approach described in this paper is that [BPS06b] strictly deals with single-valued probabilistic feature vectors. In [BPS06a] the Gauss-tree was extended to handle objects having an uncertain location as proposed in [CXP+04].

Since the Gauss-tree is the index structure our new method is based on, we will now survey the main characteristics of this approach and the processing of single-valued queries. For the Gauss-tree, a single pfv is defined as follows:

**Definition 1** A probabilistic feature vector \( v \) is a vector consisting of \( d \) pairs of feature values \( \mu_i \) and standard deviations \( \sigma_i \). Each pair defines a univariate Gaussian distribution of the true feature value \( x_i \), defined by the following probability density function:

\[
N_{\mu_i, \sigma_i}(x_i) = \frac{1}{\sqrt{2\pi \sigma_i}} \cdot e^{-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}}
\]
The probability density of a probabilistic feature vector $v$ for a given vector of actual values $x$ can be calculated in the following way:

$$p(x|v) = \prod_{i=1}^{d} N_{\mu_i, \sigma_i}(x_i)$$

Let the dimensionality of the data space be $d$, i.e. our pdf are $d$-variate Gaussian functions each of which is defined by $d$ pairs of means and standard deviation $(\mu_i, \sigma_i, 1 \leq i \leq d)$. According to this definition our method is based on independent features which is often not given in a given application. However, as in naive Bayes classification, neglecting the dependencies between the dimensions does not necessarily cause a bad retrieval performance. Furthermore, in image data the correlations between the features are more or less an inherent characteristic of the transformation method and not to a given database. Thus, it is possible to use feature transformation techniques like principal component analysis (PCA) to find orthogonal dimensions. The idea of the Gauss-tree is to regard the parameters of each Gaussian as vectors (points) of a $(2 \cdot d)$-dimensional space. The structure of the index is then inherited from the R-tree [Gut84] family, as formalized in the following definition:

**Definition 2 (Gauss-tree)**

A Gauss-tree of degree $M$ is a search tree where the following properties hold:

- The root has between 1 and $M$ entries unless it is a leaf. All other inner nodes have between $M/2$ and $M$ entries each. A leaf node has between $M$ and $2M$ entries. An inner node with $k$ entries has $k$ child nodes.

- Each entry of a leaf node is a probabilistic vector consisting of $d$ probabilistic features $(\mu_i, \sigma_i), 1 \leq i < d$.

- An entry of a non-leaf node is a minimum bounding rectangle of dimensionality $2 \cdot d$ defining upper and lower bounds for every feature value $[\bar{\mu}_i, \hat{\mu}_i]$ and every uncertainty value $[\bar{\sigma}_i, \hat{\sigma}_i]$ as well as the address of the child node.

- All leaf nodes are at the same level.

In Figure 2, we see an example of a Gauss-tree consisting of 3 levels. In the middle, we have depicted the minimum bounding rectangle of a leaf node for one of the probabilistic features. This minimum bounding rectangle allows to store feature values between $\bar{\mu} = 3.0$ and $\hat{\mu} = 4.0$ and uncertainty values between $\bar{\sigma} = 0.6$ and $\hat{\sigma} = 0.9$. A few sample pfv which are stored in this data page are also depicted. The Gaussian functions (probability density functions, pdf) which correspond to these pfv are also shown on the right side of Figure 2 in gray lines.

For query processing, we need a conservative approximation of the probability density functions which are stored on a page or in a certain subtree. Intuitively, the conservative approximation is always the maximum among all (possible) pdfs in a subtree. This maximum can be efficiently derived from the minimum bounding rectangle. In Figure 2, the
maximum function which has been derived from the depicted minimum bounding rectangle is shown on the right side using a solid black line. As a formula, the approximating pdf \( \hat{N}_{\bar{\mu}, \bar{\mu}, \bar{\sigma}, \bar{\sigma}}(x) \) is given as:

\[
\hat{N}_{\bar{\mu}, \bar{\mu}, \bar{\sigma}, \bar{\sigma}}(x) = \max_{\mu \in [\bar{\mu}, \bar{\mu}], \sigma \in [\bar{\sigma}, \bar{\sigma}]} \{N_{\mu, \sigma}(x)\}
\]

With a case distinction involving seven different cases, \( \hat{N}_.(x) \) can be efficiently and analytically computed:

\[
\hat{N}_{\bar{\mu}, \bar{\mu}, \bar{\sigma}, \bar{\sigma}}(x) = \begin{cases} 
N_{\mu, \sigma}(x) & \text{if } x < \bar{\mu} - \bar{\sigma} \\
N_{\bar{\mu}, \bar{\mu}-x}(x) & \text{if } \bar{\mu} - \bar{\sigma} \leq x < \bar{\mu} - \hat{\sigma} \\
N_{\bar{\mu}, \bar{\sigma}}(x) & \text{if } \bar{\mu} - \hat{\sigma} \leq x < \bar{\mu} \\
N_{\bar{\mu}, \sigma}(x) & \text{if } \bar{\mu} \leq x < \bar{\mu} + \sigma \\
N_{\hat{\mu}, \bar{\sigma}}(x) & \text{if } \bar{\mu} + \sigma \leq x < \bar{\mu} + \hat{\sigma} \\
N_{\hat{\mu}, \hat{\sigma}}(x) & \text{if } \bar{\mu} + \hat{\sigma} \leq x < \hat{\mu} + \hat{\sigma} \\
\end{cases}
\]

Since we assume independence in the uncertainty attributes, we can safely determine \( \hat{N}_{\bar{\mu}, \bar{\mu}, \bar{\sigma}, \bar{\sigma}}(x) \) in each dimension separately. Please note that \( \hat{N}_{\bar{\mu}, \bar{\mu}, \bar{\sigma}, \bar{\sigma}}(x) \) is not really a probability density function as it does not integrate to 1 for the whole data space. It is the conservative approximation of a set of probability density functions.

Similarly to the other index structures from the R-tree family, the Gauss-tree is constructed by iteratively inserting new objects. A node split operation is performed whenever a node exceeds its defined capacity (\( M \)). For the selection of a branch of the Gauss-tree upon insertion of a new object and for the determination of a split dimension, strategies have been proposed which minimize the integral of \( \hat{N}_{\bar{\mu}, \bar{\mu}, \bar{\sigma}, \bar{\sigma}}(x) \).

Several query types for databases of Gaussian pfv have been defined. Query objects may either be conventional \( d \)-dimensional feature vectors (exact queries) or probabilistic feature vectors (probabilistic queries). Probabilistic queries can be processed like exact queries if the variances of the query are added to the corresponding variances of the pfv stored in the database. The first defined query type is the \( k \)-most likely identification query (\( k \)-MLIQ) which reports the \( k \) objects having maximum probability-based similarity. Given the query vector \( q \), the algorithm accesses the nodes of the Gauss-tree in
increasing order of $\tilde{N}_{\mu,\tilde{\sigma},\hat{\sigma}}(x)$. A priority queue [HS95] is used to support this access order. Query processing stops when $k$ pfv have been retrieved having a higher probability at the query point than the hull function $\tilde{N}_{\mu,\tilde{\sigma},\hat{\sigma}}(x)$ of the top page in the priority queue.

In a similar way, probability threshold queries are processed. For this query type, the user specifies the threshold $P_\Theta$ of the probability of the query answers rather than the number of answers.

3 Video Retrieval using Probabilistic Feature Vectors

In this section, we will formalize video summarization using sets of probabilistic feature vectors (pfvs) following a Gaussian density function. Additionally, we will provide the probabilistic framework for comparing queries to movies and specify the new types of queries.

As mentioned before, the video part of a movie is a sequence of images which can be transformed into $d$-dimensional feature vectors $f \in \mathbb{R}^d$. Applying summarization techniques, a video is represented by a set of pfvs. Let us note that there are other notions of pfvs which are based on different density functions, but in this paper the distribution function of a pfv is considered to be Gaussian. Thus, our pfvs are defined as proposed in definition 1.

To represent a movie, we employ a set of pfvs. Each pfv is considered to represent a set of similar frames in the movie. Let us note that a pfv does not necessarily correspond to a single shot. Instead, we summarize similar frames without considering shots first. Additional to each pfv, we consider a weight $w_i$ expressing the average amount of frames represented by the given pfv $v_i$ in the complete movie. Thus, pfvs representing more frames have larger weights than pfvs representing a smaller fraction of the frames. We can now define a movie descriptor as follows:

**Definition 3** A movie descriptor $M$ is a set of pfvs $\{v_1, \ldots, v_k\}$ and a weighting $\{w_1, \ldots, w_k\}$. $w_i$ corresponds to the a priori likelihood that a frame in the movie is described by the pfv $v_i$. Furthermore, the following condition holds: $\sum_{i=1}^{k} w_i = 1$

A query is posed by specifying a video clip or only a part of it. To calculate the likelihood that the query is contained in some database object, we first of all have to apply some feature transformation to the query as well. Thus, a query $Q$ can be considered as a set of feature vectors $\{q_1, \ldots, q_l\}$ with $q_i \in \mathbb{R}^d$. To calculate the probability that $Q$ is contained in a movie described by $M$, we first of all have to derive a probability for a single query frame $q_i$ for being contained in a given pfv $v_j \in M$ having the weight $w_j$. A pfv corresponds to a density function over $\mathbb{R}^d$. Thus, we can calculate the density of $q_i$ w.r.t. $v_j$. However, to calculate a probability for a single vector in a continuous space, we would have to integrate over some interval. Since for a single vector this interval converges to 0, the probability of the vector converges to 0 as well. However, since we already observed $q_i$, we actually do not need to calculate the probability that exactly $q_i$
occurs in the given video. Instead, we can apply the theorem of Bayes and calculate the conditional probability that \( q_i \) belong to \( v_j \) under the condition it appeared at all. To formalize this condition, we have to distinguish three cases. First, \( q_i \) belongs indeed to \( v_j \). Second, \( q_i \) belongs to some other pfv \( v_k \) in the same movie \( M \). Finally, \( q_i \) is not contained in \( M \) but is part of some other movie. To approximate the last case, we specify \( H_0(q_i) \) which is modeled by a uniform distribution or the average density of any known pfv for the vector \( q_i \). Additionally, we multiply this density with the number of pfvs in the compared movie descriptor to have a weighting which is equal to the movie descriptor.

Thus, the probability that \( q_i \) appears at all is the sum of the probabilities \( p(q_i|v_i) \) that \( q_i \) belongs to some \( v_i \) describing the current movie \( M \) and the probability that \( q_i \) is not contained in \( M \). The later probability is expressed by \( H_0(q_i) \). Formally, we can calculate the probability \( P(v_j|q_i) : \)

\[
P(v_j|q_i) = \frac{w_j \cdot p(q_i|v_j)}{\sum_{v \in V} \hat{w} \cdot p(q_i|v) + H_0(q_i)}
\]

Since a movie is given by a set of pfvs, the probability that a frame \( q_i \) is contained in the complete movie described by \( M \), can be computed by summing up the probabilities for each pfv:

\[
P(M|q_i) = \sum_{v_j \in M} P(v_j|q_i)
\]

Finally, we have to consider all frames \( q_i \in Q \) of a query. Thus, we calculate the average probability for any frame in the query \( q_i \) for being contained in the given movie descriptor \( M \) by:

\[
P(M|Q) = \frac{\sum_{q \in Q} P(M|q)}{|Q|}
\]

If a query comprises large numbers of frames this method yields performance problems. Thus, we have to reduce the number of frames for the query object as well. If the query must be answered in interactive time, sophisticated summarization techniques cannot be applied. Thus, we propose a simple reduction by considering every \( i \)th frame only. If time is less important, summarization by sets of pfvs is applicable. In this case, the query is represented by a movie descriptor itself. For calculating the probability that a movie descriptor \( M_q \) describes frames which are contained in the movie described by \( M \), we will proceed as follows. We first of all determine the probability that a query pfv \( v_q \) describes the same set of feature vectors as a pfv \( v_m \) contained in the movie. This probability can be defined as follows:

The probability density of two Gaussians for describing the same vector can be specified as follows:

\[
p(v_q, v_m) = \int_{-\infty}^{+\infty} p(v_q|x)p(v_m|x)dx
\]
Having this probability, we can calculate the conditional probability for $v_m$ under the condition of $v_q$ in the following way:

$$P(v_m | v_q) = \frac{w_m \cdot w_q \cdot p(v_q, v_m)}{\sum_{v \in M} w_v \cdot w_q \cdot p(v_q, v)} + H_0$$

Using this probability, we can proceed as above. The probability for $P(M | M_q)$ is the average probability of $P(M | v_q)$ which is the sum over all $P(v_j | v_q)$ in $M$:

$$P(M | M_q) = \frac{\sum_{v_q \in M_q} \sum_{v_j} P(v_j | v_q)}{|Q|}$$

Based on these probabilities, we can specify probabilistic queries retrieving any movie in the database having a large enough probability for containing a query video clip. To decide which probability is large enough for being contained in the result set, there are two general approaches. The first is to define a fixed probability threshold, e.g. 80%. Thus, we retrieve all movies containing the specified query frames with a probability of more than 80%. Formally, we can define a set-valued probabilistic threshold query on movie descriptors as follows:

**Definition 4 (Set-Valued Probabilistic Threshold Query) (SVPTQ)** Let $DB$ be a database of movie descriptors, let $Q$ be a set of query frames and let $P_0 \in [0, 1]$ be a probability threshold. The answer of a threshold identification query is defined as follows:

$$SVPTQ_{DB}(Q, P_0) = \{ M \in DB | P(M | Q) \geq P_0 \}$$

The second method for deciding containment in the query result is to retrieve the $k$ most likely results. Thus, the threshold is relative to the database content. An example for this type of query is: Retrieve the 5 movies from the database having the highest probability for containing the query scene. We will call this type of query set-valued probabilistic ranking query (SVPRQ). In the following we will formalize SVRCQs:

**Definition 5 (Set-Valued Probabilistic Ranking Query) (SVPRQ)** Let $DB$ be a database of movie descriptors $M$, let $Q$ be a set of query frames and let $k \in \mathbb{N}$ be a natural number. Then, the answer to a set-valued probabilistic ranking query (SVPRQ) on $DB$ is defined as the smallest set $RQ_k(Q) \subseteq DB$ with at least $k$ elements fulfilling the following condition:

$$\forall M_a \in RQ_k(Q), \forall M_{db} \in DB \setminus RQ_k(Q) : P(M_a | Q) > P(M_{db} | Q)$$

## 4 Indexing Summarized Videos

After describing the queries, we are now introducing our solution for efficient query processing based on sets of probabilistic feature vectors.
4.1 Answering Set-Valued Queries

In contrast to searching in a database where each object is represented by a single pfv, our application requires the use of set-valued objects for both the query and the database objects. For query processing, we have to match all the elements of the query representation (being traditional or probabilistic feature vectors) against all the movie descriptors in the database. The difficulty of this task lies in the problem that even if a movie descriptor offers a high likelihood for containing one of the elements of our query, the corresponding movie needs not necessarily to be a likely candidate for containing the complete query. Thus, in order to prune a movie descriptor from the search space, it is necessary to approximate the probability of the complete movie descriptor for matching the complete query.

Our new method for indexing movie descriptors uses a single Gauss-tree for managing all pfvs belonging to any movie descriptor in the database. Each pfv is identified by its movie ID and an additional sequence number identifying the pfv within the movie. To utilize this data structure for answering matching queries, we will describe conservative approximations of the likelihood that the elements of a query \( Q \) are described by some movie descriptor being stored in a set of nodes belonging to the Gauss-tree.

Therefore, we will first of all calculate the probability of a query element \( q_i \in Q \) that \( q_i \) is contained in some movie \( M \) descriptor which is completely stored in a set of nodes \( P \):

**Lemma 1** Let \( Q \) be a set-valued query, let \( P = \{p_1, \ldots, p_m\} \) be a set of nodes in the Gauss-tree \( T \) containing the pfvs of a movie Descriptor \( M \in DB \). We define the function \( \text{maxDense}_P(q) \) as follows:

\[
\text{maxDense}_P(q) = \max_{p_i \in P} N_{p_i}(q)
\]

Then the following condition holds for all \( q \in Q \):

\[
\forall M \in P : P(M|q) \leq \frac{\text{maxDense}_P(q)}{\text{maxDense}_P(q) + H_0}
\]

**Proof 1**

\[
P(M|q) = \frac{\sum_{v_i \in M} w_i \cdot p(q|v)}{\sum_{v_i \in M} w_i \cdot p(q|v) + H_0(q)} \leq \frac{\max_{p_j \in P} N_{p_j}(q)}{\max_{p_j \in P} N_{p_j}(q) + H_0(q)} \\
\Leftrightarrow \sum_{v_i \in M} w_i \cdot p(q|v) \leq \max_{p_j \in P} N_{p_j}(q) \\
\Leftrightarrow \sum_{v_i \in M} w_i \cdot p(q|v) \leq \sum_{v_i \in M} w_i \cdot \max_{p_j \in P} N_{p_j}(q) \\
= \max_{p_j \in P} N_{p_j}(q) \cdot \sum_{v_i \in M} w_i = \max_{p_j \in P} N_{p_j}(q) \cdot 1
\]
Based on this lemma, we can determine the maximum probability for each element $q$ of the query $Q$ of being contained in a movie $M$ which is completely stored in the set of pages $P$. To employ this lemma for approximating the likelihood of the complete query $Q$, we must take the average of the conservative approximations over all elements of the query $Q$. The average of a set of conservative approximations must be a conservative approximation of the average of the exact values. Since each part of the sum in the average of approximations is greater or equal to the exact value, the sum of approximations is greater or equal than the sum of exact values as well. The average is the mentioned sum divided by the number of elements. Therefore, the following condition holds:

$$\forall M \in P : P(M|Q) \leq \frac{1}{|Q|} \cdot \sum_{q \in Q} \frac{\max \text{Dense}_P(q)}{\max \text{Dense}_P(q) + H_0(q)}$$

Though we can now approximate the probability that $Q$ matches some movie $M \in P$, the approximation is potentially depending on several nodes $p \in P$ at the same time. For ranking and pruning nodes in the query algorithms, we therefore prove the following lemma:

**Lemma 2** Let $Q$ be a set-valued query, let $P = \{p_1, \ldots, p_m\}$ be a set of nodes in the Gauss-tree $T$ containing the pfvs of any movie descriptor $M \in DB$. Then the following condition holds:

$$\forall M \in P : P(M|Q) \leq \frac{\max_{p \in P} N_p(q)}{\max_{p \in P} N_p(q) + H_0(q)}$$

$\text{max}_{p \in P} N_p(q) = \max_{p \in P} \text{maxProb}(Q,n)$

**Proof 2**

$$\forall M \in P : P(M|Q) \leq \frac{1}{|Q|} \cdot \sum_{q \in Q} \frac{\max_{p \in P} N_p(q)}{\max_{p \in P} N_p(q) + H_0(q)}$$

$$\leq \frac{|Q|}{|Q|} \cdot \max_{q \in Q} \frac{\max_{p \in P} N_p(q)}{\max_{p \in P} N_p(q) + H_0(q)}$$

$$= \max_{q \in Q} \frac{N_p(q)}{\max_{p \in P} N_p(q) + H_0(q)} = \max_{p \in P, q \in Q} \frac{N_p(q)}{N_p(q) + H_0(q)}$$

We can now approximate the probability $P(M|Q)$ that $M$ is completely stored in the set of nodes $P$ on the basis of a single node $p_{\text{max}}$ where $p_{\text{max}}$ is the node $p$ maximizing $\text{maxProb}(Q,p)$. An important property of this approximation is that it can be used to rank the access order of the nodes in the Gauss-tree for query processing. Additionally, we will employ this lemma for pruning unnecessary pages and terminate our queries.

Our algorithms employ two data structures. The first is a priority queue containing the nodes of the Gauss-tree that have not been examined yet. The priority is ranked with respect to $\text{maxProb}(Q,p)$ in descending order. Due to Lemma 2, $\text{maxProb}(Q,p)$ yields an
upper bound of the probability of a movie descriptor to be completely stored in the remaining nodes of the tree. Additionally, $\text{maxProb}(Q, p)$ can be considered as the maximum probability for all query elements that are yet unknown.

The above lemmas describe the case that there is a set of the nodes which are guaranteed to contain the complete set of considered movie descriptors. However, during query processing we will encounter the case that we already retrieved some pfvs for a movie $M$, but there are still some $v \in M$ which are stored in the part of the Gauss-tree that has not been examined yet. For those movie descriptors, we have to store the already known densities in the so-called candidate table until the complete set of pfvs is retrieved. Each entry in the candidate table corresponds to a movie descriptor. For each movie stored in the candidate table, we additionally store the sum of the densities for each query element $q$ and each density function $v_i$ that has been retrieved so far. Let us note that each density $p(q|v_i)$ in each sum is weighted with $w_i$ which is the weight of the pfv $v_i$ in the descriptor $M$. Finally, we store the number of all already retrieved density functions for each movie descriptor $M$. Based on this data and the current $\text{maxProb}(Q, p)$ on the top of our priority queue, we can also approximate the density of any partly known movie descriptor. The approximation is formulated in the following lemma:

**Lemma 3** Let $M$ be a partially retrieved movie descriptor, $A \subset M$ be the set of already known pfvs with weight $w_a$ and let $B \subset M$ be the still unknown elements of $M$. Furthermore, let $P$ be the set of node in the Gauss-tree $P$ containing $B$. We define the function $\text{partDensity}_A(q)$ as follows:

$$\text{partDensity}_A(q) = \sum_{v_i \in A} w_i \cdot p(q|v_i) + (1 - \sum_{v_i \in A} w_i) \cdot \text{maxDense}_P(q)$$

Then, the following condition holds:

$$P(M|q) \leq \frac{\text{partDensity}_A(q)}{\text{partDensity}_A(q) + H_0(q)}$$

Furthermore, we can state for the complete query $Q$:

$$P(M|Q) \leq \frac{1}{|Q|} \cdot \sum_{q \in Q} \frac{\text{partDensity}_A(q)}{\text{partDensity}_A(q) + H_0(q)}$$

**Proof 3** The proof is analogue to the proof of lemma 2.

4.2 Set-Valued Probabilistic Threshold Query

In our first query, we have a fixed global probability threshold $P_\Theta$ which can be employed to decide whether a movie is part of the result set. We will now explain our algorithm for
SVPTQ(\text{Query } Q, \text{ float } P_\Theta)
activePages := new PriorityQueue(descending)
candidateTable := new CandidateTable()
result := new List()
pruned := new List()
activePagesQueue.insert(root, 1.0)
DO
\text{aktNode} = activePages.removeFirst()
IF \text{ aktNode} is a directory node THEN
FOR each node in \text{ aktNode} DO
activePages.insert(node, \text{maxProb}(Q, node))
END FOR
ENDIF
IF \text{ aktNode} is a data node THEN
FOR each pfv in \text{ aktNode} DO
IF pfv.MovieID in \text{ pruned} THEN
CONTINUE
END IF
candidateTable.update(pfv.MovieID, pfv(Q))
candidateEntry := candidateTable.get(pfv.MovieID)
IF candidateEntry.isComplete THEN
IF candidateEntry.probability(Q) \geq P_\Theta THEN
result.add(pfv.MovieID)
ELSE
IF candidateEntry.approximation(Q) \leq P_\Theta THEN
pruned.add(pfv.MovieID)
candidateTable.delete(pfv.MovieID)
ENDIF
ENDIF
ELSE
pruned.add(pfv.MovieID)
candidateTable.delete(pfv.MovieID)
ENDIF
END IF
END FOR
ENDIF
END IF
WHILE((not \text{ candidateTable.isEmpty})
or activePages.topProbability \geq P_\Theta)
and not activePages.isEmpty())
RETURN result;

Figure 3: Pseudocode of Set-Valued Probabilistic Threshold Query.

processing SVPTQs using the Gauss-tree. The pseudo code of this algorithm is displayed in Figure 3. The algorithm starts by reading the root node of the Gauss-tree. For each node \( p \) being a child node of the root, we now calculate \( \text{maxProb}(Q, p) \) and insert the nodes into the priority queue which is sorted in descending order. Afterwards, the algorithm enters its main loop which iterates until the priority queue is empty. Additionally, the algorithm terminates if we can guarantee that there cannot be any movie descriptor left matching the given query \( Q \) with a likelihood larger than \( P_\Theta \). In each step, the algorithm removes the top element of the queue. If the element is a node, it is loaded and pointers to its child nodes are inserted into the priority queue, ranked by \( \text{maxProb}(Q, p) \). If the top element of the queue is a pfv, we check if there is already an entry in the candidate table corresponding to the movie descriptor \( M \) of the pfv. If not, we insert a new entry into the candidate table. In both cases, we can update the sum for each query element for the movie descriptors in the candidate table. If the current entry for the movie descriptor \( M \) is complete, i.e. all of its pfvs have been retrieved, we can calculate the likelihood. If this likelihood is larger than \( t \), we can add \( M \) to the result set. Finally, the entry for \( M \) is removed from the candidate table.
If the movie descriptor $M$ is not complete after updating the priority queue, we approximate the current maximum likelihood of $M$ and $Q$. If the conservative approximation is smaller than $t$, we can exclude $M$ from the result set. Thus, we store the ID of $M$ in a separated pruning list and delete its entry from the candidate table. If we later encounter a pfv belonging to $M$, we can safely skip its computation after checking the pruning list.

Our algorithm terminates if $\max Prob(Q, p)$ for the top element of the priority is smaller than $P_{\Theta}$. Additionally, we have to continue processing until the candidate table is empty, to make sure that the result is complete.
4.3 Set-Valued Probabilistic Ranking Query

The second query type proposed in this paper are SVPRQs. For SVPRs the minimum probability for a result depends on the movie having the $k$ highest probabilities for containing the query set. The idea of the algorithm is quite similar to the previous algorithm. However, for this type of query, we need a second priority queue storing those $k$ movies which currently have the largest probabilities for containing $Q$. We will sort this second priority queue in ascending order and refer to it as result queue. The pseudo code for this algorithm is displayed in Figure 4. We start again by ordering the descendant nodes of the root page w.r.t. $\text{maxProb}(Q, p)$. Afterwards we enter the main loop of the algorithm and remove the top element of the queue. If this element is a node, we load its child nodes. If these child nodes are nodes themselves, we determine $\text{maxProb}(Q, p)$ and update the priority queue. If the child nodes are pFVs, we check the candidate table for corresponding movie descriptor $M$ and insert a new descriptor, in the case that there is not already a descriptor for the movie $M$. Afterwards, we can update the candidate table as mentioned before. If a movie descriptor $M$ has been read completely, we can delete it from the candidate table and compare its probability $P(M|Q)$ to the probability of the top element of the result queue, i.e. the movie descriptor encountered so far having the $k$ highest probability. If the probability of $M$ is higher than that of the top element, we need to add $M$ to the queue. However, to make sure that we do not retrieve more than $k$ elements, we have to check the size of the result queue. If there are already $k$ elements, we have to remove the top element before inserting $M$. In the case, that the entry in the candidate table does not contain the complete information about $M$ yet, we still can calculate a probability estimation and compare it to the top element of the result queue. If $P(M|Q)$ is smaller than the $k$ highest probability in the result queue, we can guarantee that $M$ is not a potential result. Thus, $M$ is deleted from the candidate table and stored in our list for excluded movie descriptors. The algorithm terminates if the top of the priority containing the remaining notes provides a lower value than the top of the result queue and the candidate table is empty.

5 Experimental Evaluation

5.1 Testbed

All experiments were performed on a workstation featuring a 2.2 GHz Opteron CPU and 8GB RAM. All algorithms are implemented in Java 1.5. We evaluated our SVTCQ, SVRCQ and their comparison partner using a database of 902 music video clips recorded from various TV stations. The average length of a video clip within our collection is 4 minutes and 6 seconds. We extracted the image representations of the videos on a per-frame basis, i.e. we generated 25 features/second for PAL and 30 features/second for NTSC videos. From each image, we extracted a color histogram. For the color histogram, we used the HSV color space which was divided into 32 subspaces, 8 ranges of hue and 4 ranges of saturation.
In order to obtain the summarization for each video clip, we applied the EM clustering algorithm. The EM clustering provided us with approximately 100 multivariate Gaussians per video clip. In our experiments, we performed video similarity search. As setup step, we picked 40 query videos from our database and manually selected a set of videos which are similar to the query videos.

To generate queries, we employed two methods for collecting query frames. The first method tried to capture the complete video clip. Thus, we sampled every 50th frame from the complete clip to derive a representative sample of frames. The second method simulated queries which are posed by giving only a scene or shot from the video. Therefore, we sampled a random interval from the sequence of all frames in the video corresponding to about 500 frames, i.e. 20 seconds. For this type of query, we used every 10th frame of the query interval, i.e. we used 50 frames per query. Additional to these queries, we also generated queries which are represented by sets of probabilistic feature vectors. For representing the complete video, we again employed EM clustering for 100 clusters on the complete set of frames in one video clip. For the queries on the scenes, we clustered the 500 frames, deriving 5 Gaussians.

To have comparison partners for retrieving videos on sets of ordinary feature vectors, we generated a database containing color histograms for all frames of every video clip in our test set. We employed two well-established distance measures for set-valued objects to pose queries to this database, the Hausdorff (HD) distance and the sum of minimum distances (SMD)[EM97]. For these methods we could only use the query consisting of sets of feature vectors.

Our first set of experiments examined the precision and recall of video retrieval for all 4 types of generated queries. Therefore, we performed \(k\)NN queries for our comparison partners and SVPRQ for the methods proposed in this paper. The result for the queries on the complete video clips is displayed in Figure 5. As a first result it can be seen that...
our new method significantly outperformed the compared methods w.r.t. precision and recall. For $k = 1$, we should retrieve the database object from which the query was generated, we achieved a precision of almost 1.0. For the 2nd nearest neighbor our method still achieved a precision of about 0.9 which is about 40% better than the best of our comparison partners (SMD). The chart displaying the recall of our query results displays a similar picture. The recall of our new methods considerably outperformed the compared methods. Furthermore, we achieved a recall of over 70% for $k = 3$ which is the average number of similar videos for a query object in our test bed.

The experiments on the queries on parts of video clips display similar results. Our methods outperformed the compared method w.r.t both precision and recall. Though the performance advantage w.r.t. precision was smaller than in the previous experiment, our proposed method still managed to outperform the best comparison partner, SMD, by more than 20% for all values of $k$. The results w.r.t. recall display similar improvements as
well. To conclude, representing video clips as sets of Gaussians is well suited for accurate video retrieval and outperforms method based on sets of feature vectors w.r.t. precision and recall.

For measuring the efficiency of our new methods for query processing, we recorded the time taken for processing all 40 queries representing the complete movie. For each query object, we performed several queries corresponding to several parameter setting (\(1 < k < 7\) and \(0.1 < P_{\Theta} < 0.7\)). The results are displayed in Figure 7. The average query time for our new methods was approximately 7 times smaller than that of the compared methods. Additionally, it can be seen that using sets of probabilistic feature vectors as query representation did not cause a considerable longer average query time. Let us note that the time for generating the Gaussians of the query was not added to the query time. To conclude our new query algorithm considerably outperformed the compared methods w.r.t. efficiency as well.

6 Conclusions

In this paper, we have proposed efficient techniques for high performance video retrieval. Our methods are based on a summarization technique using probabilistic feature vectors, i.e. Gaussian probability density functions. For storage and efficient retrieval of probabilistic feature vectors, a specialized index structure, the Gauss-tree, has been applied. Every video clip in the database is associated to a set of probabilistic feature vectors. A query video clip is also transformed into either a set of conventional feature vectors or into a set of probabilistic feature vectors. In both cases, query processing involves matching of sets of vectors. We have defined two kinds of set-valued queries, set-valued probabilistic ranking queries and set-valued probabilistic threshold queries, and have proposed efficient algorithms for query evaluation on top of the Gauss-tree. Our experimental evaluation using over 900 music video clips demonstrates the superiority of our approach with respect to both accuracy as well as efficiency of retrieval.

References


[BPS06a] C. Böhm, A. Pryakhin, and Matthias Schubert. “Probabilistic Ranking Queries on Gaussians”. In 18th Int. Conf. on Scientific and Statistical Database Management (SSDBM 2006), Vienna, Austria, 2006.


151
Autonomes Index Tuning –
DBMS-integrierte Verwaltung von Soft Indexen

Martin Lühring¹  Kai-Uwe Sattler¹  Eike Schallehn²  Karsten Schmidt³
¹Fakultät für Informatik und Automatisierung, TU Ilmenau
²Fakultät für Informatik, Universität Magdeburg
³FB Informatik, TU Kaiserslautern


1 Einführung


Vor diesem Hintergrund rücken seit wenigen Jahren Self-Management-Features in den Blickpunkt des Interesses sowohl der Datenbanksystemhersteller als auch im Forschungsbereich. Trotz großer Fortschritte in den aktuellen Systemversionen ist jedoch leicht einzu-sehen, dass es bei der Komplexität heutiger Systeme und der Vielzahl von „Tuning-Reglern“ noch ein weiter Weg bis zu wirklichen autonomen Tuning ist.
Eine der wichtigsten Tuning-Aufgaben ist das Index Tuning, oft auch als Index Selection Problem (ISP) bezeichnet. Hierbei geht es um die Bestimmung der Indizes (die Indexkonfiguration), die eine gegebene Menge von Anfragen (Workload) am besten unterstützen, wobei zwischen dem Gewinn durch die Indexe und die zusätzlichen Kosten für den Speicherplatz bzw. die Verwaltung (Anlegen und Aktualisieren) abgewogen werden muss. Obwohl das ISP seit den 80er Jahren in der Literatur behandelt wird, wurde es bis heute im Wesentlichen als Entwurfsproblem betrachtet. Entsprechend bieten die führenden kommerziellen DBMS sogenannte Index Advisors oder Wizards an, die einen gegebenen Workload analysieren und darauf aufbauend eine Empfehlung für anzulegende Indizes geben. Die Entscheidung ob und wann diese Indizes erzeugt werden, bleibt weiterhin dem DBA vorbehalten.


- Änderungen des Workloads wie in den oben genannten Beispielen,
- Änderungen des (relativen) Datenvolumens in Tabellen und der Datenverteilung,
- Änderungen am Datenbankschema,
- Änderungen der Infrastruktur, z.B. Austausch von Hardwarekomponenten,
- Interferenzen mit anderen Tuning-Maßnahmen, z.B. Partitionierung oder die Auswahl materialisierter Sichten.

2 Problemstellung und verwandte Arbeiten

Wie im vorangegangenen Abschnitt erwähnt, kann Index Tuning als Form des Index-Selection-Problems (ISP) betrachtet werden. Formal kann dies wie folgt ausgedrückt werden [CFM95]:

Gegeben sei eine Menge von Anfragen $Q_1, \ldots, Q_m$ sowie die Menge der Indexkandidaten $I_1, \ldots, I_n$, jeweils mit den Verwaltungskosten $mcost(I_i)$ (für die Aktualisierungen) und der Größe $size(I_i)$. Der Gewinn $profit$ eines Indexes $I_i$ für eine Anfrage $Q_k$ ergibt sich aus der Differenz der Ausführungskosten mit ($cost(Q_k, I_i)$) und ohne Nutzung ($cost(Q_k)$) von $I_i$:

\[
profit(Q_k, I_i) = \max\{0, cost(Q_k) - cost(Q_k, I_i)\}
\]

(1)

Gesucht ist eine Indexkonfiguration $C \subseteq \{I_1, \ldots, I_n\}$ aus materialisierten Indexen (d.h. für Anfragen nutzbar), die

\[
\sum_{i=1}^{m} \max\{profit(Q_i, I_j) : I_j \in C\} - \sum_{I_j \in C} mcost(I_j)
\]

(2)

maximiert und dabei eine vorgegebene Größenschranke $S$ nicht überschreitet:

\[
\sum_{I_j \in C} size(I_j) \leq S
\]

(3)

Es handelt sich hierbei um ein NP-Problem [Com78] als eine Variante des bekannten Rucksackproblems bzw. der ganzzahligen linearen Optimierung [KPP04]. In der Literatur sind hierzu verschiedene Lösungen beschrieben. Neben der naheliegenden Greedy-Variante und Ansätzen auf Basis der dynamischen Programmierung als exakte Lösungen wurden auch approximative Varianten vorgeschlagen [CFM95].

Autonomes Index Tuning als dynamische Lösung des ISP basiert auf grundlegenden Ansätzen zum Self Management in DBMS, die den Bemühungen in verschiedenen Teilgebieten gemein sind. Ein wichtiger Aspekt ist dabei das generelle Vorgehen zur Steuerung autonomen Systemverhaltens, an der auch die folgenden Darstellungen ausgerichtet sind. Grob kann das dynamische Verhalten in Phasen/Aufgaben unterteilt werden, welche kontinuierlich und iterativ durchlaufen werden.

(1) **Monitoring:** die Überwachung des vorhergehenden Systemverhaltens in Form von Messwerten und/oder Ereignissen über einen bestimmten Zeitraum, sowie eine geeignete Verwaltung der daraus gewonnenen Informationen.

(2) **Decision:** basierend auf den gewonnenen Informationen müssen Entscheidungen getroffen werden, die Änderungen des aktuellen Systemzustands nach sich ziehen, da zum Beispiel eine Optimierung des Systemverhaltens möglich ist oder sich abzeichnende Probleme oder Fehler vermieden werden können.

(3) **Action:** darauf folgend muss durch geeignete Aktionen die getroffene Entscheidung zur Änderung des Systemzustands umgesetzt werden, wobei der laufende Betrieb während der Aktionsausführung möglichst wenig beeinträchtigt werden soll.


Im Gegensatz zu früheren Arbeiten wie etwa [ISR83, FST88] sind diese Lösungen dadurch charakterisiert, dass sie den Anfrageoptimierer des DBMS nutzen. Auf diese Weise kann sichergestellt werden, dass bei der Ermittlung der Indexempfehlungen das gleiche Kostenmodell verwendet wird. Teilweise wird auch die Bestimmung eines Workloads unterstützt (Monitoring-Aspekt) – auch dies jedoch entkoppelt vom eigentlichen Betrieb. Techniken, die hier zum Einsatz kommen, sind u.a. das Clustering von Anfragen [GPSH02, CGN02].

Ein erster Ansatz eines echten autonomen Online-Tunings ist das auf DB2 basierende QUIET-System [SGS03, SSG04]. Dieses sammelt langfristig Informationen über vom Optimierer empfohlene Indexe und legt diese dann automatisch an bzw. löscht nicht mehr benötigte Indexe. Jedoch ist das System nicht in das DBMS integriert und verursacht einen gewissen Overhead.

3 Autonomes Index Tuning

Der in den folgenden Abschnitten vorgestellte Soft-Index-Ansatz stellt eine Weiterentwicklung des QUIET-Systems dar, indem neben einer engen Integration mit dem DBMS drei Ziele verfolgt werden: (1) das kontinuierliche Monitoring von Anfragen zur Identifikation von (potenziell) nützlichen Indexen, (2) das autonome Anlegen bzw. Löschen von Indexen unter Berücksichtigung von Speicherplatzschranken und Verwaltungskosten und (3) die Erzeugung von Indexen „on the fly“, d.h. durch Ausnutzung von Table-Scans.

3.1 Architektur

Darüber hinaus wurden die B+-Baum-Indizes um folgende Eigenschaften erweitert:

- **managed** sind alle Soft Indizes, die vom System automatisch verwaltet (angelegt bzw. entfernt) werden,
- **virtual** Indizes werden nur für die Anfrageoptimierung bzw. die Indexempfehlung angelegt, d.h. sind im Data Dictionary eingetragen (inkl. der notwendigen Größeninformationen) jedoch nicht materialisiert und für andere Anfragen nicht sichtbar,
- **deferred** Indizes werden dagegen zunächst „leer“ angelegt und erst später mit Daten gefüllt.

Die weiteren Betrachtungen beschränken sich daher auf B+-Baum-Indizes. Prinzipiell ist jedoch eine Übertragung auf andere Indexarten problemlos möglich.

### 3.2 Monitoring: Workload-Monitoring und Indexempfehlung

Das Monitoring zum Soft-Index-Management umfasst zwei Ebenen: die lokale Anfrageebene und die globale Workload-Ebene. Die **Anfrageebene** wird durch den Index Advisor behandelt, der dem in [VZZ+00] beschriebenen Ansatz folgt.

Bei der Analyse einer Anfrage $Q$ zur Ableitung von Indexempfehlungen wird diese zuerst auf konventionelle Weise optimiert und die Kosten $\text{cost}(Q)$ werden ermittelt, wobei hierbei zunächst alle Soft Indizes ignoriert werden. Dann wird die Anfrage syntaktisch analysiert um potenzielle Indexkandidaten zu ermitteln. Indexkandidaten sind zum Beispiel einzelne Spalten in den `where`, `group by`, `order by` und `select`-Klauseln sowie Spaltenkombinationen in diesen Klauseln. Für jeden Indexkandidaten wird ein virtueller Index mit abgeschätzten Statistikinformationen angelegt. Nun wird die Anfrage $Q$ unter Berücksichtigung der virtuellen Indizes noch einmal optimiert und die Kosten $\text{cost}(Q, I)$ werden bestimmt. Die im Plan genutzten Soft-Indizes (einschließlich der virtuellen Indizes) werden als Indexempfehlung $I = \{I_1, \ldots, I_k\}$ zurückgegeben. Der Profit für $I$ ergibt sich danach aus

$$\text{profit}(Q, I) = \text{cost}(Q) - \text{cost}(Q, I) \quad (4)$$

Der letzte Schritt lässt noch offen, wie der Gesamtprofit auf die einzelnen Elemente von $I$ aufgeteilt wird. In [SSG04] haben wir dazu verschiedene Alternativen untersucht. Ein praktikabler Ansatz ist die Aufteilung entsprechend der Indexgrößen, d.h. der Profit eines Index $I \in I$ berechnet sich aus:

$$\text{profit}(Q, I) = \frac{\text{profit}(Q, I) \cdot \text{size}(I)}{\sum_{I_j \in I} \text{size}(I_j)} \quad (5)$$

Hierbei ist zu berücksichtigen, dass es sich nur um Annäherungen handelt, da gilt:

$$\text{profit}(Q, I) \neq \sum_{I \in I} \text{profit}(Q, I) \quad (6)$$
Dies wird beispielsweise bei einer Anfrage mit einem Merge-Join deutlich: der Profit eines Plans mit zwei Indexen auf den Verbundattributen ist hier sicher größer als die Summe der Profite, wenn jeweils nur ein Index vorhanden wäre.

Ein zweites Problem ist die Behandlung von Update-Operationen, die ggf. eine Aktualisierung der Indexe erfordern. Ein möglicher Ansatz ist die Bestimmung eines negativen Profits für alle betroffenen Indexe auf Basis der Anzahl der betroffenen Tupel $n_{rows}$ der Operation $Q_U$ und der Höhe des Baums ($F$ bezeichnet hierbei einen empirisch ermittelten Kostenfaktor):

$$\text{profit}(Q_U, I) = -\text{height}(I) \cdot n_{rows}(Q_U) \cdot F$$  \hspace{1cm} (7)

Auf Workload-Ebene besteht die Aufgabe des Monitorings im Sammeln und Aggregieren der Indexempfehlungen für die laufenden Anfragen. Dies wird vom Soft Index Manager übernommen, der dafür den Index Advisor nutzt. Eintreffende Anfragen werden dabei wie oben beschrieben vom Index Advisor verarbeitet. Hierbei muss nicht jede Anfrage berücksichtigt werden – es genügt eine repräsentative Stichprobe zu beobachten. Die gewonnenen Indexempfehlungen werden in einem Indexkatalog $D$ gesammelt, der pro Index $I$ folgende Informationen enthält:

- den Gesamtnettoeffekt $\text{benefit}(I)$ aus den kumulierten Einzelernprofiiten pro Anfrage,
- die Größe $\text{size}(I)$, die für noch nicht materialisierte Indexe aus der Kardinalität der Basisrelation und der Attributgröße abgeschätzt wird und bei materialisierten Indexen durch den tatsächlichen (aktuellen) Wert ersetzt,
- den Status $\text{state}(I)$ des Index, d.h. ob der Index gegenwärtig materialisiert ist oder nicht.

Da die Bestimmung der optimalen Indexkonfiguration ein aufwändiger Prozess ist, wird dies nicht nach jeder Anfrage ausgeführt. Vielmehr wird jeweils eine Epoch von Anfragen betrachtet, in der die Indexempfehlungen zunächst nur gesammelt werden. Die Länge der Epochen kann einfach über die Anzahl der Anfragen oder durch das Erreichen eines vorgegebenen Gesamtnettoprofits definiert werden.

Das Monitoring in Epochen ermöglicht darüber hinaus eine „Dämpfung“ des Einflusses länger zurückliegender Indexempfehlungen, sodass eine bessere Anpassung an Drifts im Workload erzielt werden kann. Hierzu werden die Indexempfehlungen (inklusive des erwarteten Profits mit Zeitstempeln versehen (d.h. beispielsweise einen monoton wachsenden Anfragezähler). Der Gesamtnettoeffekt für eine Epoch, die zum Zeitpunkt $t_{E}$ endet, mit den Empfehlungen zu einem Index $I$ aus Anfragen mit dem Zeitstempeln $t_{s_1}, t_{s_2}, \ldots, t_{s_k}$ berechnet sich danach aus:

$$\text{benefit}(I) = \sum_{j=1}^{k} \frac{\text{profit}(Q_{t_{s_j}}, I)}{t_{s_E} - t_{s_j}}$$  \hspace{1cm} (8)

Dieser Gesamtnettoeffekt wird am Ende einer Epoch auf alle korrespondierenden Einträge im Indexkatalog aufsummiert.
Ein weiteres Problem ist die Berücksichtigung von sich überlappenden Indexen. Dies bedeutet beispielsweise, dass ein Index-Scan über einem Index auf $R(A)$ auch den Index auf $R(A, B)$ nutzen könnte. Demzufolge kann der Profit einer Indexempfehlung zu $R(A)$ auch dem Index zu $R(A, B)$ zugewiesen werden. Hierfür benötigen wir den Begriff des Index-Containment: ein Index $I_1$ ist in $I_2$ enthalten ($I_1 \subseteq I_2$) gdw. beide Indexe auf der gleichen Relation definiert sind und die Attribute von $I_1$ unter Berücksichtigung der Sortierordnung des Indexes (aufsteigend bzw. absteigend) ein Präfix der Attribute von $I_2$ sind. In diesem Fall kann der Profit entsprechend zugewiesen werden:

$$\forall I_i \in D : I_e \subseteq I_i \Rightarrow profit(Q, I_i) = profit(Q, I_e)$$ (9)

3.3 Decision: Soft-Index-Auswahl

Aufgabe der Soft-Index-Auswahl ist die Bestimmung der optimalen Indexkonfiguration unter Berücksichtigung des aktuellen Workloads und der gegebenen Speicherplatzschranke. Wie bereits in Abschnitt 2 dargestellt, handelt es sich hierbei um das Rucksackproblem, wobei der Fokus auf der Online-Verarbeitung liegt, d.h. jeweils zum Ende einer Epoche soll die Entscheidung darüber getroffen werden, ob neue Indexe anzulegen sind bzw. ob ggf. materialisierte Indexe gelöscht werden sollen.

Ein praktikabler Lösungsansatz für dieses Problem ist die Nutzung eines Greedy-Verfahrens, das wie folgt arbeitet: Die Indexkandidaten (d.h. alle materialisierten und nicht-materialisierten Soft Indexe) werden bezüglich ihres relativen Gewinns $relative\_benefit(I) = \frac{benefit(I)}{size(I)}$ absteigend sortiert. Anschließend wird Algorithmus 1 ausgeführt, der solange Indexkandidaten in eine initial leere Konfiguration aufnimmt, bis die Speichergrenze $plimit$ erreicht ist. Das Ergebnis ist eine neue Indexkonfiguration $C$, die jedoch nur dann realisiert wird, wenn der Gewinn gegenüber der alten Konfiguration $C$ oberhalb eines vorgegebenen (empirisch ermittelten) Schwellwertes $threshold$ liegt. Auf diese Weise wird das Thrashing – das wechselweise Anlegen und Löschen der gleichen Indexe – reduziert.

Das Realisieren einer Indexkonfiguration bedeutet prinzipiell, dass alle noch nicht materialisierten Indexe der neuen Konfiguration angelegt werden und die nicht mehr in der neuen Konfiguration vorhandenen materialisierten Indexe entsprechend zu löschen sind. Es sei angemerkt, dass sich Anlegen und Löschen nur auf den eigentlichen B+-Baum-Index bezieht – die Einträge im Indexkatalog bleiben in jedem Fall erhalten.

Der Aufwand dieses Auswahlverfahrens ist einschließlich der Sortierung offensichtlich $O(n \log n)$ bei $n$ Indexkandidaten. Ein Nachteil des Greedy-Ansatzes ist jedoch, dass keine Garantie für das Finden der optimalen Lösung besteht. Trotzdem haben bisherige Forschungsergebnisse gezeigt, dass der gewählte Ansatz mit vertretbarem Aufwand ausreichend gute Ergebnisse liefert. Dies liegt an der Größe des Suchraums, der Verwendung
Algorithm 1 Greedy-Algorithmus zur Indexauswahl

1: $\mathcal{I}[1 \ldots n] := \text{sort}(\mathcal{D})$ by relative benefit;
2: $\mathcal{C} := \emptyset$;
3: avail_space := plimit;
4: overall_benefit := 0;
5: for all $k := 1 \ldots n$ do
6: \hspace{1em} if avail_space - size($\mathcal{I}[k]$) > 0 then
7: \hspace{2.5em} $\mathcal{C} := \mathcal{C} \cup \{\mathcal{I}[k]\}$
8: \hspace{1em} avail_space := avail_space - size($\mathcal{I}[k]$
9: \hspace{1em} overall_benefit := overall_benefit + benefit($\mathcal{I}[k]$)
10: end if
11: end for
12: if overall_benefit < threshold then $\mathcal{C} := \mathcal{C}$ end if
13: return $\mathcal{C}$

von Abschätzungen als Eingabeparameter und der Nutzung des Ergebnisses als Vorhersage des zukünftigen Anfrageverhaltens.

3.4 Action: Planoperatoren für Indexerstellung


Um sowohl eine möglichst aktuelle Indexkonfiguration als auch geringen Mehraufwand zu erreichen, verfolgen wir hier folgenden Ansatz. Die Indexe werden während der Anfrage erzeugt, indem z.B. ein Table-Scan auf der Basisstabelle genutzt wird, um die Tupel zu lesen und in den Index einzufügen [Gra00]. Als weitere Optimierung wird der angelegte Index wenn möglich direkt in der Anfrage genutzt, indem der Plan entsprechend angepasst wird. Zur Umsetzung werden zwei neue Planoperatoren benötigt, die nachfolgend beschrieben werden. Basis hierfür bilden die neu eingeführten Deferred-Indexe – die Materialisierung der Indexe besteht somit im Erzeugen von Deferred-Indexen. Dies bedeutet, dass zunächst nur leere Indexe angelegt werden, die im Schemakatalog entsprechend markiert sind und erst später gefüllt werden.
### 3.4.1 IndexBuildScan

Der IndexBuildScan (notiert als $\xi$) ist eine Erweiterung des Table-Scan-Operators, die als zusätzlichen Parameter eine Liste von Deferred-Indexen erwartet. Der Operator liest zunächst wie der normale Scan-Operator blockweise alle Tupel der Basisrelation, fügt diese jedoch gleichzeitig in die entsprechenden Deferred-Indexe ein. Somit lassen sich prinzipiell in einem Scan mehrere Indexe zur gleichen Basisrelation füllen. In Algorithmus 2 ist das Verhalten entsprechend der klassischen Iterator-Schnittstelle dargestellt.

#### Algorithmus 2 IndexBuildScan-Operator $\xi_L$ auf einer Relation $r$ mit opt. Prädikat $P$

<table>
<thead>
<tr>
<th>State:</th>
<th>Next:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: scan_complete := false</td>
<td>18: tuple := seqscan_next($r$)</td>
</tr>
<tr>
<td>2: for all $I \in ILIST$ do</td>
<td>19: if tuple $\not= \perp$ then</td>
</tr>
<tr>
<td>3: if deferred($I$) then</td>
<td>20: for all $I \in L$ do</td>
</tr>
<tr>
<td>4: prepare_index($I$)</td>
<td>21: if deferred($I$) then</td>
</tr>
<tr>
<td>5: end if</td>
<td>22: insert($I$, tuple)</td>
</tr>
<tr>
<td>6: end for</td>
<td>23: end if</td>
</tr>
<tr>
<td>7: open_relation($r$)</td>
<td>24: end for</td>
</tr>
<tr>
<td>8: close_relation($r$)</td>
<td>25: if $P$(tuple) then</td>
</tr>
<tr>
<td>9: if scan_complete then</td>
<td>26: return tuple</td>
</tr>
<tr>
<td>10: for all $I \in L$ do</td>
<td>27: else</td>
</tr>
<tr>
<td>11: if deferred($I$) then finish($I$) end if</td>
<td>28: next()</td>
</tr>
<tr>
<td>12: end for</td>
<td>29: end if</td>
</tr>
<tr>
<td>13: else</td>
<td>30: else</td>
</tr>
<tr>
<td>14: for all $I \in L$ do</td>
<td>31: scan_complete := true</td>
</tr>
<tr>
<td>15: if deferred($I$) then reset($I$) end if</td>
<td>32: return tuple</td>
</tr>
<tr>
<td>16: end for</td>
<td>33: end if</td>
</tr>
<tr>
<td>17: end if</td>
<td></td>
</tr>
</tbody>
</table>

Eine besondere Behandlung erfordert dabei der Fall, dass der Scan nicht vollständig ausgeführt wurde, z.B. durch Abbruch der Anfrage. Da hierbei nicht mehr garantiert werden kann, dass der Index vollständig gefüllt ist, muss der Index in einem solchen Fall zurückgesetzt werden.

#### Rewriting Regeln. Der IndexBuildScan-Operator kann im Anfrageplan überall dort eingesetzt werden, wo der Table-Scan $\sigma_{SEQ}^P$ (mit dem optionalen Selektionsprädikat $P$) auf der gleichen Basisrelation ausgeführt werden soll. Als Randbedingung gilt, dass die Deferred-Indexe der Parameterliste $L$ alles Indexe auf der jeweiligen Basisrelation sein müssen:

$$\sigma_{SEQ}^P(r) \Leftrightarrow \sigma_P(\xi_L(r)) \quad \text{falls } \forall I \in L : I \text{ ist Deferred-Index auf } r$$  \hspace{1cm} (10)

\[
idx\_pages = \left\lfloor \frac{|r|}{\text{page\_size} \cdot (\frac{\text{size\_attr}}{\text{page\_size}} + \frac{\text{size\_rowid}}{\text{page\_size}})} \right\rfloor \cdot \frac{1}{\text{fill\_factor}}
\]  

(11)

Da die Anzahl der inneren Knoten typischerweise viel kleiner als die der Blattknoten ist (< 5%), kann diese Zahl ignoriert werden:

\[
cost = \text{pages}(r) + \sum_{l \in L} (|r| \log |r| + idx\_pages(I) \cdot \text{cost_{page\_write}})
\]  

(12)


Weiterhin sei angemerkt, dass der IndexBuildScan prinzipiell auch in Kombination mit dem Sort-Operator eingesetzt werden kann, indem Tupel sortiert ausgelesen und entsprechend in den B+-Baum eingefügt werden.

3.4.2 SwitchPlan

Die Verwendung des IndexBuildScan-Operators allein erlaubt noch nicht die Nutzung des erzeugten Indexes in der gleichen Anfrage. Hierfür wird der SwitchPlan-Operator (notiert als ⇔) eingesetzt, der eine spezielle Variante des in [GW89] vorgestellten Choose-Plan-Operator darstellt. Aufgabe dieses Operators ist es, während der Ausführung eines Plans zwischen zwei Teilplänen umzuschalten. In unserem Fall ist die Umschaltung zwischen einem IndexBuildScan und einem IndexScan als Teil einer Nested-Loops-Operation (z.B. eines Verbundes). In Abbildung 2 ist dies an einem Beispielplan dargestellt: Der Nested-Loops-Join mit den beiden Table-Scans auf den Relationen \( r \) und \( s \) wird durch einen Plan
ersetzt, der einen IndexBuildScan zum Erzeugen eines Indexes auf dem Verbundattribut von \( s \) mit dem SwitchPlan-Operator kombiniert. Der IndexBuildScan wird dabei nur während der ersten Iteration auf der \( s \)-Relation ausgeführt, in allen folgenden Durchläufen wird der neu erzeugte Index mit dem Index-Scan verwendet – der Nested-Loops-Join wird somit zum Index-Nested-Loops-Join.

Abbildung 2: Beispiel für den Einsatz des SwitchPlan-Operators

Die Realisierung des Operators ist einfach: Es wird nur eine Zustandsvariable benötigt, die zwischen den beiden Phasen (links und rechts) unterscheidet (Algorithmus 3).

**Algorithmus 3** SwitchPlan-Operator

<table>
<thead>
<tr>
<th>State:</th>
<th>Next:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: phase := 0</td>
<td>9: if phase = 1 then tuple := left.next() end if</td>
</tr>
<tr>
<td>Open:</td>
<td>10: if phase = 2 then tuple := right.next() end if</td>
</tr>
<tr>
<td>2: if phase = 0 then</td>
<td>11: return tuple</td>
</tr>
<tr>
<td>3: phase := 1</td>
<td></td>
</tr>
<tr>
<td>4: left.open()</td>
<td></td>
</tr>
<tr>
<td>5: else if phase = 1 then</td>
<td></td>
</tr>
<tr>
<td>6: phase := 2</td>
<td></td>
</tr>
<tr>
<td>7: right.open()</td>
<td></td>
</tr>
<tr>
<td>8: end if</td>
<td></td>
</tr>
</tbody>
</table>

**Rewriting Regeln.** Der SwitchPlan-Operator kann grundsätzlich genutzt werden, um zwischen einem normalen Scan und einem IndexBuildScan umschalten zu können (Regel 13 und 14). Daher kommt der Operator primär als (rechter) Kindknoten von Nested-Iteration-Knoten wie z.B. Nested-Loops-Join, Mengenoperationen oder geschachtelte Selektionen zum Einsatz (15). Voraussetzung für eine Ersetzung ist weiterhin die Existenz eines IndexBuildScan weiter unten im Baum, wobei einer der zu erzeugenden Indexe die Verbund- bzw. Selektionsbedingung der Nested-Iteration-Operation unterstützen muss:

\[
\begin{align*}
\sigma_p^{\text{SEQ}}(r) & \iff (\sigma_\phi(\xi_L(r)) := \sigma_p^{\text{IND}}(r)) & (13) \\
\sigma_p^{\text{IND}}(r) & \iff (\sigma_\phi(\xi_L(r)) := \sigma_p^{\text{IND}}(r)) & (14) \\
r \bowtie_p s & \iff (r \bowtie_p (\sigma_P(\xi_L(s)) := \sigma_p^{\text{IND}}(s))) & (15)
\end{align*}
\]
Kostenabschätzungen. Die Gesamtkosten für einen Teilplan mit einem SwitchPlan-Operator setzen sich aus den Kosten des linken Teilplans und den \((\text{card}_{\text{left}} - 1)\)-fachen Kosten des rechten Teilplans zusammen, wobei \text{card}_{\text{left}} die (geschätzte) Kardinalität des Ergebnisses des linken Teilplans bezeichnet.

\[
\text{cost} = \text{cost}_{\text{left}} + (\text{card}_{\text{left}} - 1) \cdot \text{cost}_{\text{right}} \tag{16}
\]

Ist der Teilplan mit der Nested Iteration wiederum rechter Teil einer weiteren Nested Iteration, muss beachtet werden, dass der IndexScan-Kostenanteil rekursiv pro Iteration in die Gesamtkosten eingeht, während der Anteil der Indexerstellungskosten nur einmal eingeht.


3.4.3 Entscheidungsmodell

Werden die Erstellungskosten für Indexe einfach in die Gesamtkosten eines Plans einbezogen, so würde dieser Plan aufgrund der höheren Kosten vom Optimierer zugunsten eines anderen, nicht-indexerzeugenden Plans verworfen. Daher werden die Gesamtkosten in Ausführungs- und Erzeugungskosten aufgeteilt, wobei letztere die Summe aller Erzeugungskosten von IndexBuildScan-Operatoren im Plan sind. Der Optimierer wählt dann einen indexerzeugenden Plan (mit oder ohne SwitchPlan-Operator) aus, wenn folgende Bedingungen erfüllt sind:

1. die Verwendung von Soft Indexen ist eingeschaltet (Konfigurationsparameter),
2. es liegen Deferred-Indexe für die in der Anfrage referenzierten Basisrelationen vor,
3. der Plan ist bezüglich der reinen Ausführungskosten der günstigste,
4. der Gesamtprofit der zu erzeugenden Indexe abzüglich der Erzeugungskosten ist positiv bzw. überschreitet ein gegebenes Limit.

Das Anlegen der Deferred-Indexe erfolgt davon unabhängig durch den Soft-Index-Manager. Dies bedeutet, dass (1) nur dann indexerzeugende Pläne ausgewählt werden, wenn zuvor passende Soft Indexe angelegt wurden und dass (2) die Materialisierung von Indexen bis zum Ausführen einer entsprechenden Anfrage verzögert werden kann.
3.5 PostgreSQL-Implementierung

Die beschriebenen Komponenten wurden vollständig in PostgreSQL 7.4.8 implementiert. Hierzu waren Erweiterungen in folgenden Bereichen notwendig.

**Systemkatalog.** Der Systemkatalog wurde um Relationen zur Verwaltung der Indexkandidaten erweitert. Konkret sind dies

- die Erweiterung der `pg_index`-Relation um ein Feld zur Kennzeichnung von Soft-Indexen bzw. Deferred-Indexen,
- die Relation `pg_softindex_indexadvice` zur Speicherung der Empfehlungen einer Epoche,
- die Relation `pg_softindex_indexinfo` als Realisierung des Indexkatalags.

Die beiden letzteren Relationen werden nur vom Soft-Index-Manager genutzt, während die Informationen der ersten Relation auch vom Planner verwendet werden.


**Executor.** Die Executor-Erweiterungen betreffen im Wesentlichen die Umsetzung der neuen Planknoten. In der Implementierung wurden zur Vereinfachung speziell der Anfrageplaner die beiden Operatoren `ξ` und `⇌` in einem Planknoten zusammengefasst, der beim ersten Open-Next-Close-Zyklus den IndexBuildScan (sofern Deferred-Indexe vorhanden sind, sonst einfach den TableScan) realisiert und – wenn benötigt – bei allen Folgeaufrufen in die zweite Phase umschaltet.

**Konfiguration.** Das Verhalten des Soft-Index-Managements kann (zumindest in der Testphase) über einige Parameter beeinflusst werden. Der wichtigste Parameter ist dabei die Variable `softindex_level`, die interaktiv über eine `set`-Anweisung belegt

4 Evaluierung


Die Zeiten für die Scans wurden mit einer Anfrage der Form „select count(*) from ...“ bestimmt. Die Zeitunterschiede bei der Indexerstellung sind durch die unterschiedliche Anzahl der Distinct-Werte der jeweiligen Spalte bedingt. Ein Vergleich mit der Kombination „create index + IndexScan“ ist an dieser Stelle nicht angegeben, da die Zeit für den IndexScan im Wesentlichen von der Selektivität der Bedingung abhängig ist bzw. ein In-

---

\(^1\)www.tpc.org/tpch
dexScan über die gesamte Relation nicht sinnvoll ist. Wie aus dem Diagramm ersichtlich wird, verursacht der IndexBuildScan wie erwartet die gleichen Kosten wie ein `create index`, hat demgegenüber jedoch den Vorteil, die Indexerstellung „nebenbei“ auszuführen. Der Preis dafür ist allerdings eine Verzögerung der Anfrage – im Falle eines einfachen Scans – um 50 – 150%.

Prinzipiell bietet der IndexBuildScan noch die Möglichkeit, mehrere Indexe auf der gleichen Tabelle in einem Scan zu erstellen. Allerdings haben unsere Experimente gezeigt, dass dies gegenüber einer sequenziellen Indexerstellung aufgrund des konkurrierenden Indexaufbaus und der Dominanz der Erstellungskosten keinen Gewinn liefert.


Vor allem die Verarbeitung geschachtelter Anfragen profitiert von der dynamischen Indexerzeugung, allerdings nur bei Plänen mit Nested Iterations (Nested Loops, Subselects). Bei anderen Verbundoperationen ergeben sich aus dem SwitchPlan-Operator dagegen keine Vorteile. Die neuen Operatoren unterstützen komplexe Anfragen, können jedoch den `create index`-Befehl noch nicht komplett ersetzen. Die Integration in die Anfrageverarbeitung lässt allerdings die sofortige Indexnutzung zu und reduziert die Externspeicherzugriffe.


Die Index-Pool-Größen wurden dabei so gewählt, dass alle Indexe materialisiert werden können („groß“ = `G`) bzw. nur die für einen Anfragemix optimalen Indexe, d.h. jeweils die großen `lineitem`-Indexe und die entsprechenden kleinen („mittel“ = `M`) bzw. nur die kleinen Indexe („klein“ = `K`). Die Konfigurationen sind in Abbildung 5 entsprechend durch `EE-P-I` kodiert, wobei `EE` für die Epochenlänge, `P` für die Poolgröße (groß ... klein) und `I` für die Art der Indexerstellung (`C` über `create index`, `S` über `IndexBuildScan`) stehen. Die Ergebnisse zeigen insbesondere, dass die Größe des Index-Pools hier einen wichtigen Einflussfaktor darstellt (von `01-K-C` zu `01-G-C` eine Verbesserung um 32%) und dass die Einführung der Epochen zu einem Gewinn führt (von `01-K-C` zu `49-K-C` 21%). Allerdings wird auch ersichtlich, dass die On-the-Fly-Indexerstellung nicht zwingend einen Vorteil bedeutet (`33-M-C` vs. `33-M-S`) – sondern nur dann zum Tragen kommt, wenn Anfragen direkt von den zu erstellenden Indexen profitieren.

glichen (Abbildung 6). Im Diagramm sind deutlich die Adaptionsschritte (Ausreißer nach oben) der „Soft Index“-Kurve zu erkennen.


Schließlich wurde noch der Overhead des Soft-Index-Managements untersucht. Dazu wurden bei einer Epochenlänge von 33 und einer Index-Pool-Größe von knapp 60.000 Seiten die Zeiten für die Bearbeitung der Workloads gemessen. Abbildung 7 gibt die prozentualen Anteile für die einzelnen Schritte an, ohne den Anteil für die Anfrageausführung. Hierbei hat sich gezeigt, dass der Aufwand der eigentlichen Auswahl (Greedy) sowie der Behandlung der virtuellen Indexe vernachlässigbar ist. Neben dem nachvollziehbaren hohen Anteil zur Realisierung der Konfiguration sind in unserer Implementierung die Katalogzugriffe recht teuer: Nach jeder betrachteten Anfrage werden die Indexempfehlungen in die PostgreSQL-Katalogtabellen des Soft-Index-Managers übernommen, was mit Lese- und Schreibzugriffen auf den Katalog verbunden ist. Hier sind noch Optimierungen notwendig, etwa durch Halten der benötigten Informationen in einem Shared-Memory-Bereich.
5 Fazit & Zusammenfassung


Die Erfahrungen aus Implementierung und Evaluierung in PostgreSQL und der Vergleich mit unserem Vorhaben in QUIET haben gezeigt, dass eine enge Integration in die Anfrageoptimierung und -ausführung möglich und sinnvoll ist, speziell um das Kostenmodell zu nutzen und den Overhead zu reduzieren. Auf diese Weise ist ein autonomes Tuning möglich, dass insbesondere für sich ändernde Workloads geeignet ist.


Ein weiterer Punkt ist, dass der Aufwand der Indexerstellung nicht vollständig eliminiert werden kann – vermeiden lassen sich nur mehrfache Scans. Hierbei muss auch noch berücksichtigt werden, dass die vorgestellte „Huckepack“-Indexierung eine gewisse Verzögerung der Anfrage verursacht und nicht immer ist dies akzeptabel. Ein möglicher Ausweg sind eventuell dynamische sich „selbst-tunende“ (d.h. zugriffsbalancierte) Indexstrukturen [SSG05].

Literatur


Abstract: With information infrastructures getting more and more complex, it becomes necessary to give automated support for managing the evolution of the infrastructure. If changes are detected in a single system, the potential impact on other systems has to be calculated and appropriate countermeasures have to be initiated to prevent failures and data corruption that span several systems. This is the goal of Caro, our approach to change impact analysis in large information infrastructures. In this paper we present how we model the metadata of information systems to make a global analysis possible, regardless of the data models used, and how the analysis process works.

1 Introduction

In today’s businesses, information infrastructures are getting more and more complex. There are many heterogeneous systems with a manifold of mutual dependencies leading to unmanageability of the overall infrastructure. New dependencies between existing systems evolve and new systems are added. Generally, there is no central management of all systems.

Small, local changes to system metadata can have a major impact on a company-wide scale due to the dependencies between systems. This can lead to failures of large parts of the information infrastructure. Probably all companies have some sort of change management to prevent this from happening. However, the more complex the information infrastructure gets, the more difficult it gets to ensure the correct operation of all systems. It is necessary to automate as much of the change management as possible. This will avoid human failures, and in general will be faster than manually keeping track of all dependencies between systems. For our generic approach, we use a broad interpretation of the terms system, change, and metadata. A system is an abstract concept which provides services that can be used by other systems, and in turn may depend on other systems, too. Concrete systems could be DBMS, libraries, websites, or end user applications. The metadata of a system consists of every information that could, if it is changed, influence the correct operation of other systems. A classic example for metadata are database schemas, but also APIs, file locations, or system configurations. Even non-functional aspects like quality and performance assertions are included in our definition of metadata. Our goal is to be able to process all forms of metadata, to make automated CIA as precise as possible.
Problem Statement  The core of change management is the analysis of the potential impact of a local change on other systems. This is called change impact analysis (CIA). Ideally, the analysis is performed before applying a change (preventive CIA) to prevent failures. However, this is not always possible, if the changing system does not know about or does not care about a dependent system because it belongs to a different organizational domain. A good example here is a web data integration system relying on screen scraping of external websites. It is then necessary to perform reactive CIA. The faster this happens, the more likely it is to avoid or reduce the impact of system failures from happening.

There exist several problem areas that have to be coped with to perform automated, generic CIA at a company-wide scope:

- The heterogeneity of the metadata makes a direct analysis practically impossible, since an analyzer would have to understand all existing metadata models or formats.
- For analysis, metadata has to be provided in an explicit way. In some cases, this is not a problem; for example, SQL databases provide a standardized way to read schema information and file system structures can be easily scanned. In many cases this is more difficult, because the systems provide no appropriate way to read the metadata, or metadata is only implicitly defined in compiled code. Some metadata, like non-functional aspects, may not be extractable automatically at all. Thus, in CIA it has to be assumed that the provided metadata is not complete.
- System autonomy and organizational constraints lead, on one hand, to failure or non-applicability of preventive CIA, and on the other hand, make reactive CIA more difficult because changes may go undetected for a long time.

Contribution  In this paper, we present a meta-model used to store arbitrary, heterogeneous metadata, and an analysis component for a fine-grained and customizable impact analysis. In practice, it is impossible to resolve the mentioned problems of metadata incompleteness and system autonomy. Therefore, our principle is to work with what we have, and not to require the input metadata to be of a certain quality. Of course, if the provided metadata is incomplete, the analysis results will not be as exact and reliable as otherwise. We want to avoid overlooking a potential problem in all cases. Therefore, with incomplete metadata, the probability of more false alarms increases. In the worst case, when it is only known that system A accesses system B “in some way”, every change of B, regardless of its real impact, will result system A to be marked as impacted by a change, and may trigger counteractions. With more complete metadata, analysis results will improve, reducing the probability of false positives. This “best effort” approach makes Caro applicable to arbitrary environments. The trade-off between the effort of providing complete metadata and the quality of the analysis results can be determined separately for each case.

Related Work  In the context of information integration, much research has been done. Some approaches are complementary to ours, and others are similar to Caro in some aspects. The most important distinguishing features of Caro are its genericity, robustness and scope. It makes no assumptions about the environment it operates in, and can be used for any scenario where change impact analysis is necessary.
Dorda et al. [DSS04] present an approach which is similar to Caro with respect to the problems addressed. However, the solution they propose is different in two fundamental points: They require a central documentation (or metadata) repository and a strict process policy. This constrains their approach to scenarios where it is feasible to have a central repository and to enforce adherence to defined processes. While they want to avoid integration clusters, we think that such a clustering (and thus decentralization) in large enterprise information systems cannot be avoided.

Deruelle et al. [DBGN99] present another approach to change impact analysis. They use a multigraph and change propagation rules for analysis, which is very similar to Caro. Their approach has several limitations: The focus lies on preventive CIA, thus they lack a framework to support reactive CIA. Apparently, they do not consider the problem of incomplete metadata. Also, their meta-model and rules are rather specialized, which makes the extension to support other data models and change types more difficult than with Caro.

Various other approaches to CIA in information systems exist that are limited with respect to the supported data models [Ke02] or scope and support of exact analysis [MAL+05]. The concepts of change impact analysis in software systems [BA96, Aji95, RT01] are similar to the ones we use. However, the models and analysis procedures focus on the elements that are found in software: methods, signatures, classes, attributes and so on. In addition, CIA for software systems is usually done preventively. Aspects of heterogeneity, metadata incompleteness and distribution are not as relevant as they are in information systems.

Research done in the field of schema evolution [Rod92, AFK+04, ZR98], schema matching [RB01, MRB04, MP99] or model management [Ber03] are complementary to our approach. Especially the latter approaches are used to plan and realize integration, generally between only two or a small group of systems, as well as adapt systems to changing requirements. Caro is not designed for use in the initial stages of an integration project. It will take the results of such a project, namely the dependencies between the systems that were created based on schema matches or mapping definitions, and monitor them for changes. When a change occurs, Caro will analyze the impact of it. Depending on the results, different actions may be taken. The most simple thing to do is to notify a responsible person, or to shut down the system to prevent further damage. Also, Caro may interface other information integration tools and provide analysis results to them, e.g. for automatic repair. Caro focuses on the monitoring of systems participating in the overall information infrastructure and the detection of the global impact of changes. As such, it “fills the gap” to an overall management of a heterogeneous integrated environment.

The Object Management Group (OMG) defined with the Common Warehouse Metamodel (CWM) [CWM03] a way to relate elements of different data models to each other with respect to structure. While our type system is similar to CWM in some places, the focus in Caro is to express change impact semantics, not schema structure. It is easy to to use CWM or other existing models in Caro, as we show in section 3.

The algorithms we use for CIA are heavily based on graph matching [Bun00]. We use a modified version of the CSI-Algorithm [KH04], which is in turn an improved version of Ullman’s algorithm [Ull76].
This paper is structured as follows. First, in section 2, we give a short overview of Caro, and position this contribution in the larger context of the Caro architecture as well as a high-level overview about our metamodel and the analysis process. The formal foundations for model and analysis follow in section 3. Our current research prototype is presented in section 4. Besides the implementation itself, we discuss some performance issues there, too. Finally, we conclude in section 5 and give an outlook on further work to be done in this field of research.

2 Caro Overview

Caro is an approach to change management in large information system infrastructures. Basically, Caro monitors all participating information systems. If changes occur, they are analyzed for their impact on the rest of the information infrastructure. In figure 1, the three-layered architecture of Caro is depicted. The lowest layer consists of the information systems themselves. The middle layer consists of the metadata agents (MDAs). For each system, there is a metadata agent responsible for it\(^1\). An MDA monitors an information system and notifies the server in the upper layer of changes. It is also responsible for transforming the systems metadata into the common format which is introduced in the next section. Finally, if during change impact analysis potential problems are detected, the MDAs are responsible for initiating countermeasures ranging from notifying the administrator to shutting down the system. The upper layer of our architecture consists of the Caro servers. There, the inter-system dependencies are stored and the analysis process is coordinated.

We assume that it is not feasible to have a single Caro server responsible for a whole information infrastructure. Due to organizational circumstances, it may be necessary to run more than one Caro server, as shown in figure 1. Between different departments each running a Caro server, there exists a single connection point to other departments, which simplifies setup and communication greatly in comparison to a centralized approach.

Caro servers can be used for preventive as well as reactive CIA:

\[^{1}\text{A single MDA implementation may be responsible for more than one IS, but logically, there is one MDA per information system.}\]
• For preventive CIA, users can propose a hypothetical change request to Caro, and get the CIA results without actually modifying the systems. This way, the user can make sure that either his proposal does not affect other systems, or that the other systems administrators know in advance of the change. A common use of this is when modifying database schemas by adding, deleting or altering views and tables.

• In the case of reactive CIA, an MDA notifies its server of a change in the information system it monitors, and impact analysis is initiated. If a potential impact on other systems was found, the corresponding MDAs will be notified and can then take adequate actions. Caro reacts to a change, and while it is not always possible to undo or mask the change, every system that is potentially affected will at least be notified. This way, change impact is detected quickly after the change, making it possible to prevent possible damage like data corruption or system failures. For example, in a screen scraping scenario, if the website from which information is extracted is changed, several actions are possible, depending on the abilities of the impacted systems and the availability of corresponding tools. The most simple action is to notify an administrator and/or to shut down dependent systems. There may also be means to notify users that current results may not be accurate due to a change, or even modify the extraction procedure to match the new website structure. Caro acts as agent that mediates between systems, users, administrators and tools.

In this paper, we focus on the model we use to represent the heterogeneous metadata of the information systems, and how the metadata is analyzed for changes. The metadata extraction and transformation as well as the monitoring tasks that the MDA has to handle are not in the scope of this paper. However, in section 4 we discuss some of our experiences with extracting and transforming SQL metadata in our prototype.

To illustrate our meta-model and the analysis process, we will use the example depicted in figure 2. There is a DBMS named DB with a schema consisting of two tables, and a reporting application RA that uses two different select statements to access the data in the DBMS.
Systems and Dependencies Our approach targets large information infrastructures with many systems that are related to each other by their dependencies. Each system is described by its metadata, which also contains the description of the dependencies a system may have. We define the terms usage and provision to specify the parts of a system’s metadata which express dependency information.

A provision is the part of the metadata that captures services provided to some other system. For a DBMS, this is the part of the schema that is accessible from other systems. For web services, the accompanying WSDL file can be seen as provision. Since access rights may not be the same for everyone, a system can have more than one provision. The counterpart of a provision is the usage. In an usage, a system specifies the services provided by another system that it depends on. Usages are always subsets of the corresponding provision. We avoided the well known term “export schema” and “import schema” for provisions and usages, since they may contain more than only schema information.

In our example, the provision of DB consists of the complete schema in figure 2. The usage of RA (in respect to DB) is depicted in figure 3. Only the elements highlighted in bold belong to the usage. The problem of incomplete metadata becomes visible here. The two select statements do not specify RA’s metadata completely. Probably RA makes assumptions about the data types of the columns and about primary keys. The only information we can deduce from the select statements is that the deadline column is expected to be of type date because it is compared with a date in the where clause. Also, we have to assume that RA needs all columns of the projects table. That means that if any column is added to or deleted from the projects table in DB, Caro must assume that it has an impact on RA. If, however, the salary column of the employee table is deleted, or an Clim is added, Caro should recognize that this has no impact on RA.

Of course, it is not realistic to have such a fine-grained documentation of all schema dependencies in an large and complex environment. But in general, it is at last possible to find out about dependencies at the system level, which is “better than nothing”. The likelihood of false alarms will rise, but changes will not go undetected. In some areas, however, it is feasible to get fine-grained dependency information, e.g., by extracting it from BPEL [BPE06] documents.

The Caro Meta-Model The model we use to represent metadata is a simple digraph with typed nodes. Each node represents a metadata element or a relationship between two metadata elements, similar to the E/R model [Che76]. Nodes are atomic information units. So, each change that occurs can be described by a set of added and deleted nodes and edges. Figure 4 shows how metadata of DB and RA as well as their provision and usage

```
create table employees (  
    id integer primary key,  
    name varchar(50),  
    salary integer  
);  
create table projects (  
    pid integer primary key,  
    p_name varchar(50),  
    deadline date,  
    manager integer references employees(id)  
);  
```

Figure 3: Usage of the reporting application
(which are just subgraphs) look like in our graph model. The reporting application is a proprietary application with a custom meta model, indicated by the **Report** and **Field** node types. Note that the notation used is informal and serves only for illustration purposes. Nodes of type *Literal* are shown in boxes with rounded corners and in double quotes, all other nodes are labeled with their type. For the sake of simplicity, we omitted the relationship nodes and assigned their type to the edges of the graph. Element nodes belonging to a usage may have a completeness specification for incoming and outgoing relationships. Values can be atLeast or exact, both shown in the figure. The **employee table node** specifies that it needs at least the hasCol relationships that are specified in the usage, and thus, analysis won’t care if other columns are added or deleted. The **projects table**, however, specifies that it needs exactly the hasCol relationships that are specified. Thus, if any column is added or deleted here, the analysis will recognize a potential problem. The usage is connected to the rest of the graph via various derivation relationships like derived, use or copy that express the different semantics of the dependencies. These types are defined in the change impact description model defined below.

It is straightforward to represent arbitrary metadata in our model, since any type system may be used. For a generic change impact analysis, these metadata representations will be useless, since they are not meta model independent. For meta model independence, Caro defines the *change impact system description model* (CISDM) which expresses the semantics of metadata nodes that are relevant for CIA. Figure 5 shows some of the types the CISDM defines in a UML-like [UML03] notation. Relationship nodes always connect two element nodes (which can be of type **Element** or **Literal**). For each property that is relevant to CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined. The associations between the CIA, subclasses of **Element** and **Relationship** are defined.
existing meta models, like the CWM [CWM03], the CISDM does not focus on an abstraction layer for metadata exchange, modeling similar concepts in different meta models as subclasses of a common superclass. The CISDM only captures the semantics needed for CIA, which simplifies the addition of new meta models. The analysis process only works on the CISDM types, and does not mind the underlying data model.

Performing Impact Analysis Each node in a metadata graph has a set valued status attribute, the change status, which will be set during CIA. Initially, the status attribute of all nodes is empty. If a change occurs, either proposed by a user or determined by an MDA, each node that was added to the graph gets the status added, and each deleted node is not actually deleted but gets the status deleted. After that, CIA is conducted as a two-step process that may cascade over several metadata graphs, if there is a chain of dependent systems. First, intra-model analysis performed, followed by inter-model analysis:

- In the intra-model analysis phase, the status attribute of all nodes affected by the change is set accordingly. For example, if a part is deleted from a compound, the compound will get the status partDeleted. In our example, that happens with the table where a column was deleted.
- The inter-model analysis just copies all change status values of a provision to the corresponding nodes in the related usage.
- The process repeats for the metadata graph of the dependent systems, and stops when at some point all potentially impacted nodes are marked accordingly.

We illustrate this using our example scenario. Consider the changes at top of figure 7. We included only the actually changed parts of the provision. Nodes marked with ⊘ have been deleted, and nodes marked with ⊕ have been added (1). In the intra-model analysis for DB, the Table elements will get the status partDeleted/partAdded (2). The subsequent inter-model analysis copies all change status values to the usage of RA (3). Without preceding intra-model analysis, no status values would have been copied.
since the usage does not mention the salary column, and cannot know yet about the description column. As a last step, intra-model analysis is performed in the metadata graph of RA, adding the status sourceChanged to the report urgent_projects.

Technically, intra-model analysis is done by applying a set of change propagation rules to the metadata graph until no more rules can fire. As example, a rule might look for all added parts, and mark the corresponding compound as with the partAdded status, as it happened in our example. In the example, only one report was marked as problematic, which is correct, but it is not obvious why the other report was not marked. The rule responsible for this can be stated in an informal way as “if a relationship was added or removed from an element, and the completion specification for this relationship is exact, add the status sourceChanged to all elements connected via a derived or copy relationship”. In chapter 3 we will show a formal way to specify such rules. Inter-model analysis works by first matching corresponding usages and provisions, and then copying the change status values to the usages.

3 Formal Model

After having given an overview of our model and analysis process, how metadata is represented and how the generic CIA process works in the previous section we are now ready to present the formal model of the Caro metadata graph and the analysis process. In all formulas, lower case letters stand for single elements of a set, upper case letters represent sets, and letters in gothic stand for sets of sets. If an element is a tuple \(x = (a, b, c, \ldots)\), subelements are referred to as \(a_x, b_x, c_x\) etc. Figure 8 summarizes the naming convention for variables used throughout the paper and may serve as a reference.
3.1 Meta-Model

Let us first describe how the meta-model concepts informally introduced in section 2 can be captured in a formal metadata model. A digraph $d$ is an ordered pair $d = (V_d, E_d)$, where $V_d$ is the set of nodes, and $E_d \subseteq V_d \times V_d$ with $E_d = \{(v_a, v_b) | v_a, v_b \in V_d, v_a \neq v_b\}$ a set of ordered pairs defining the edges. A subgraph $s \subseteq d$ is defined as $s = (V_s, E_s)$ with $V_s \subseteq V_d$, $E_s \subseteq V_s \times V_s$, $E_s \subseteq E_d$. A metadata graph $g$ is a graph with additional properties $g = (V_g, E_g, Pr_g, Us_g)$ such that $d = (V_g, E_g)$ is a digraph; $G$ is the set of all metadata graphs. $Pr$ and $Us$ are the sets of provision and usage subgraphs: $Pr \subseteq \wp(d)$, $Us \subseteq \wp(d)$, with $\wp(d)$ being the set of all subgraphs of $d$. The attribute function $a$ will be defined later.

In the formal model, a node type as introduced in figure 5 is just a named set of nodes. We call the set of all nodes $Node$, and thus, all node types $T$ are a subset of $Node$. The basic node types are defined as follows:

$$AbstractElement = Element \cup Literal; AbstractElement \subset Node$$
$$Relationship \subset Node$$
$$Relationship \cap AbstractElement = \emptyset; Literal \cap Element = \emptyset$$

Attributes of a node are defined by the attribute function $a : AN \times V \rightarrow AV_{AN}$. $AN$ is the set of attribute names $AN = \{\text{type, status, value, completeness}_{\text{in/out}}\}$, and $AV_{AN}$ is the set of possible values for the corresponding attribute. The function $a$ is defined as

$$a(\text{type}, v) \mapsto T; T \subset Node$$
$$a(\text{status}, v) \mapsto \begin{cases} \text{if } v \notin \text{Literal} : S_v; S_v \subseteq S \\ \text{else: undefined} \end{cases}$$
$$a(\text{value}, v) \mapsto \begin{cases} \text{if } v \in \text{Literal} : z; z \in \text{String} \\ \text{else: undefined} \end{cases}$$
$$a(\text{completeness}_{\text{in/out}}, v) \mapsto \begin{cases} \text{if } v \in \text{Element} : Cs_v; Cs_v \subseteq Cs \\ \text{else: undefined} \end{cases}$$

For $a(x, v)$ we also write $x(v)$. Each node is assigned a most specific type. Only literal nodes have a value (the literal they represent), which is always a
The status is a subset of all possible status values $S$, with $S = \{\text{added}, \text{deleted}, \text{partAdded}, \text{partDeleted}, \text{partChanged}, \text{sourceChanged}, \ldots\}$. Literals have no status attribute, because they represent only values.

The completeness attribute is only specified for element nodes; relationship nodes are always complete according to equation (2). The completeness specification $Cs$ is defined as $Cs = \{(T, cs) | T \subseteq \text{Relationship}, cs \in \{\text{atLeast}, \text{exact}\}\}$. To avoid conflicts, subtypes must have the same or a stronger completeness value, and each type may only have one completeness value: Let $Cs_v$ be a completeness specification, then $\forall x, y \in Cs_v :$

$$T_x \subset T_y \implies cs_x = cs_y \lor cs_x = \text{exact}$$
$$x \neq y \implies T_x \neq T_y$$

Generally, all nodes have an identity, and nodes are not shared between metadata graphs. Literals are an exception to this, because literal values are unique: $\forall n_1, n_2 \in \text{Literal} : \text{value}(n_1) = \text{value}(n_2) \Leftrightarrow n_1 = n_2$. Therefore, nodes in the intersection of the sets of nodes of two graphs are always literal nodes:

$$\forall x, y \in G : x \neq y \implies V_x \cap V_y \subseteq \text{Literal} \quad (1)$$

Graph edges can only connect elements with relationships and vice versa, enforcing a bipartite element/relationship graph. $\forall (a, b) \in E :$

$$a \in \text{Element} \implies b \in \text{Relationship},$$
$$a \in \text{Relationship} \implies b \in \text{AbstractElement}$$
$$a \notin \text{Literal}.$$ 

Subtypes can be further restricted, as we discussed in section 2. Moreover, all relationship nodes must have exactly one incoming and one outgoing edge:

$$\forall v \in \text{Relationship} \exists a, b \in V : (a, v) \in E \land (v, b) \in E \quad \text{(at least one edge)}$$
$$\forall a, b \in \text{AbstractElement}, v \in \text{Relationship} :$$

$$\{(a, v), (b, v) \in E \implies a = b \quad \text{(at most one edge)}$$
$$\quad (v, a), (v, b) \in E \implies a = b$$

3.2 Analysis

As already outlined in the previous section, analysis is performed using intra-model and inter-model analysis steps. These are formalized in the this subsection. In both steps, matching two graphs is an important operation. In intra-model analysis it is used to find out where to apply rules, and in inter-model analysis it is used to match a usage against a provision graph.

We define a node match relation $K \subseteq \text{Node} \times \text{Node}$:

$$K = \{(u, l) | u, l \in \text{Node} \land (u, l \notin \text{Literal} \land u \neq l) \lor (u, l \in \text{Literal} \land u = l)\}.$$
We chose \( u \) and \( l \) are mnemonics for upper and lower, if we imagine an “upper” graph being matched against a “lower” graph. Literals match only themselves, whereas other nodes match only different nodes. This is due to definition (1). Depending on the context, a refined \( K_x \subset K \) will be defined. Note that \( K_x \) is not necessarily symmetric!

Let \( g_u, g_l \in G \) be two graphs. We define a graph match relation \( \mathcal{M}_{g_u, g_l} \subseteq \wp(K) \) as

\[
\mathcal{M}_{g_u, g_l} = \{ M | M \in \wp(K) \land \forall m_1, m_2 \in M: (u_{m_1}, u_{m_2}) \in E_{g_u} \land (l_{m_1}, l_{m_2}) \in E_{g_l} \land u_{m_1} = u_{m_2} \leftrightarrow l_{m_1} = l_{m_2} \},
\]

(bijection)

For analysis, only the set of matches of greatest common subgraphs \( \mathcal{G}_{g_u, g_l} \subseteq \mathcal{M}_{g_u, g_l} \) with \( \mathcal{G}_{g_u, g_l} = \{ M | M \in \mathcal{M}_{g_u, g_l} \land \exists M' \in \mathcal{M}_{g_u, g_l} : |M'| > |M| \} \) is relevant.

If \( |\mathcal{G}_{g_u, g_l}| = 1 \), the match between \( g_u \) and \( g_l \) is called unique, if \( |\mathcal{G}_{g_u, g_l}| > 1 \) it is ambiguous. In the following, we refer to \( M_{u,l} \in \mathcal{G}_{g_u, g_l} \) as match between \( g_u \) and \( g_l \). A matched subgraph \( s_{1,M} = (V_s, E_s) \) of \( g_l \) (and accordingly, \( s_{u,M} \)) itself is defined as

\[ V_s = \{ v | v \in V_{g_l} \land \exists v_u \in g_u : (v, v_u) \in M \} \text{ and } E_s = \{ e | e, e_b \in V_s \land e \in E_g \}. \]

Generally, finding the greatest common subgraph of two graphs is a hard problem. The metadata graphs that are analyzed by Caro, however, have a very regular structure. That is true for the example graph in figure 4 and we argue that in all practical cases, the structure will be similarly regular. This speeds up the matching process significantly. Our benchmarks (see section 4) show that the approach is fast enough to be used in real world scenarios.

**Intra-Model Analysis** The intra-model analysis works by applying a set of propagation rules onto the graph \( g_c \). These rules add change status values, which may enable other rules to fire. This continues until no more rules can fire. Rules have a premise, basically a digraph that is matched against the metadata graph, and a conclusion specifying a node and the status to add to this node. First, we define the node match relation for intra-model analysis:

\[ K_r = \{ k | k \in K \land \text{type}(k_u) \supseteq \text{type}(k_l) \land \text{status}(k_u) \subseteq \text{status}(k_l) \land \text{completeness}_{\text{in/out}}(k_u) \subseteq \text{completeness}_{\text{in/out}}(k_l) \}. \]

A rule node \( k_u \) matches a graph node \( k_l \) if its type is a supertype of the graph nodes type, its status is a subset of the graph nodes status, and the completeness specification also is compatible. This node match relation is not symmetric.

A rule \( r \) is a pair \( r = (p, c) \). The premise \( p \) is a triple \( p = (V_p, E_p, a_p) \). \( V_p, E_p, a_p \) defined like for metadata graphs. The conclusion \( c \) is a pair \( c = (v, s) \) with \( v \in V_g, s \in S \). \( R \) is the set of all rules. As an example, figure 9 shows the rule that marks compounds as changed if parts were added, as it happened in step (2) in figure 7. Rules are applied to the graph
\[ r = (p, c); \]
\[ p = (V_p, E_p, a_p); \]
\[ c = (v_1, \text{partAdded}); \]
\[ V_p = \{v_1, v_2, v_3\}; \]
\[ E_p = \{(v_1, v_2), (v_2, v_3)\}; \]

with the apply function: \(\text{apply}(g) \mapsto g' = (V_{g'}, E_{g'}, a'_{g'}, Pr_{g'}, Us_{g'})\) such that

\[ a'_{g'}(\text{attr}, v) \mapsto \]

\[ \begin{cases} 
\text{if } \text{attr} = \text{status} \land \exists r \in R, \exists M \in \mathcal{S}_{g_r,g} : \\
M = |V_{p_r}| \land \\
\exists m \in M : m_u = v_{c_r} \land m_l = v : \\
\text{there is a matching rule} \\
\text{(rule matches completely)} \\
\text{(rule changes this node)} \\
\text{else } a_g(\text{attr}, v). 
\end{cases} \]

We define the analysis result relation \( A \subseteq G \times G \), with

\[ A = \{(g, g') | g, g' \in G \land \text{apply}(g') = g' \land \exists n \in \mathbb{N} : \underbrace{\text{apply} \circ \text{apply} \circ \ldots \circ \text{apply}}_{n \text{ times}}(g) = g'\}. \]

**Inter-Model Analysis** In the inter-model analysis, a provision and a corresponding usage are matched, and the change status values are copied from provision to usage. Let \(g_u, g_l\) be two metadata graphs where system \(u\) depends on system \(l\). Then inter-model analysis transforms \(g_u\) to \(g'_u = (V_{g_u}, E_{g_u}, a'_{g_u}, Pr_{g_u}, Us_{g_u})\).

Let \(pr_l \in Pr_l\) be a provision subgraph, \(us_u \in Us_u\) the corresponding usage subgraph, and \(M \in \mathcal{S}_{us_u,pr_l}\) a graph match. The used node match relation is \(K_l = \{k | k \in K \land \text{type}(k) = \text{type}(k_l) \land \text{added} \notin \text{status}(k_l)\}\). That is, the usage is matched against the provision without the added nodes to avoid ambiguous matches.

The new attribute function \(a'_{g_u}\) is defined as

\[ a'_{g_u}(\text{attr}, v) \mapsto \]

\[ \begin{cases} 
\text{if } \text{attr} = \text{status} \land \exists m \in M, v_l \in V_{g_u} : v = m_u \land v_l = m_l : \\
a_g(\text{attr}, v_l) \\
\text{else } a_g(\text{attr}, v) 
\end{cases} \]

**Computational issues** With subgraph matching, there exists the possibility of ambiguous subgraphs, that is, there is no unique greatest common subgraph according to the chosen \(K\). In intra-model analysis, if \(|\mathcal{S}_{g_r,g}| > 1\) the rule \(r\) matches more than one time,
which is expected, since there may be more than one place that was changed. In inter-
model analysis, however, \(|E_{g_1,g_2}| > 1\) means that there is an ambiguity in matching
the graphs. In theory, this leads to non-deterministic behaviour of the analysis, which we
want to avoid. In practice, the issue is not so severe, since the usage is a subgraph of the
provision, and will be matched completely. An ambiguous match indicates an error in the
usage or provision subgraph.

Another important issue is the computability and termination of our algorithms. We re-
quire \(|Node| < \infty, |R| < \infty, |S| < \infty\). All other sets of elements used in our definitions
are built using set operations and thus are finite, too. That makes it possible to evaluate all
expressions by enumerating all possibilities.

Only for the analysis result relation \(A\) we need to show that it is computable, i.e., the pro-
cess terminates. This is intuitively clear, since the apply function only adds monotonically
change status values. To prove it, we have to show that

\[
\forall g \in G \exists n \in \mathbb{N} \exists g' \in G : \text{apply}(g') = g' \land \text{apply} \circ \text{apply} \circ \cdots \circ \text{apply}(g) = g'.
\]

We define \(scount : G \rightarrow \mathbb{N}\) as

\[
scount(g) : \sum_{v \in V_g} |a_g(status, v)|.
\]

Since \(\forall v \in V_g : a_{\text{apply}(g)}(status, v) \supseteq a(status, v)\), also \(scount(\text{apply}(g)) \geq
scount(g)\), that is, apply is monotonic. Further, \(\forall v : a(status, v) \subseteq S\). That means
\(\max(|a(status, v)|) = |S|\). Therefore, \(\forall g \in G : scount(g) \leq |V_g| \cdot |S|\), so there is an
upper bound for \(scount(g)\), which proves (3).

3.3 Extensibility

While we try to handle as many cases as possible in the CISDM and the corresponding
analysis rules, there will naturally be cases where our rules and types do not suffice or
even give wrong results with certain metadata. The analyzer makes no difference between
CISDM types or data models subclassing it. And while the provided rules only work on
CISDM types, user defined rules can use any node type in their premises. Users can ex-
tend/customize Caro by providing their own types, nodes, change status values and rules.
This way they can provide missing functionality, and even “override” the system provided
analysis rules by just ignoring the change status values they produce. Of course, this is
not advisable in a distributed environment, since it has to be made sure that change status
values propagated to other systems are understood by their metadata agents. However, in
some cases customizing may be necessary, since we cannot anticipate all possible scenar-
or
4 Implementation and Performance

The current prototype implementation of Caro is written in Java. The analysis component is capable to do intra- and inter-model analysis. The prototype is already in use in a small setup: The computer science department of the University of Kaiserlautern uses a custom-made exam registration system, PAS, for students. PAS is a web application using the PostgreSQL [PGS05] DBMS to store data. To ensure that the schema in the production system matches the current implementation, we deployed Caro. In terms of our model, the DBMS is one system, and the web application another, dependent system. Caro extracts metadata from the information schema for the DBMS, and parses the DDL and DML statements in the web applications source code. This will support developers by pointing out changes that were made to the schemas since the last update.

In this section, we present the important components of the implementation and give performance results. As base data for all performance tests, we used an SQL schema consisting of 350 tables with a total of 4923 columns, which amounts to 46609 nodes in the metadata graph. The schema is borrowed from HISPOS [HIS], an administration application for universities. For testing, we used a machine with four dual core 1.5 GHz processors, and assigned 1.75 GB ram to the (single threaded) benchmark application. Finally we make some remarks about deployment effort of Caro based on our current experiences.

As we have seen in section 2, metadata agents are responsible for extracting the metadata from the information systems, converting it to the graph model, and sending it to the change manager. Currently we are able to extract metadata directly from the information schema of an SQL DBMS, or from DDL statements. We also are able to read DML statements. We support tables, views, primary and foreign keys and, to a limited degree, constraints. For data transfer and export, we use GXL [WKR01], an XML format for storing graphs.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix : <http://caro.bstumm.de/rdf/Caro/CISDM#>.

:Node a owl:Class.
:Element rdfs:subClassOf :Node, [a owl:Restriction; owl:onProperty :outrange; owl:allValuesFrom :Relationship].
:Literal rdfs:subClassOf :Node, [a owl:Restriction; owl:onProperty :outrange; owl:allValuesFrom owl:Nothing].
:AbstractElement rdfs:subClassOf :Node; owl:unionOf (:Element ;Literal).
:Relationship rdfs:subClassOf :Node, [a owl:Restriction; owl:onProperty :outrange; owl:allValuesFrom :AbstractElement].
:hasPart rdfs:subClassOf :Relationship, [a owl:Restriction; owl:onProperty ciam:inrange; owl:allValuesFrom :Compound].

:hasPart rdfs:subClassOf :Relationship, [a owl:Restriction; owl:onProperty ciam:outrange; owl:allValuesFrom :Part].

Figure 10: Subset of the CISDM definitions
**Model Implementation**  The model implementation consists mainly of a simple, hash map based graph class and the type system. Types as well as edge restrictions are defined in OWL [OWL04] for convenience reasons: One can easily define subtype relationships and restrictions for relationship types, and then let the OWL reasoner of the Jena Framework [HP05] infer all implicit knowledge, such as the transitive hull of the subtype relationship. Figure 10 shows a small subset of the type definitions. We use the N3 syntax [N3] here.

**Inter-Model Analysis Implementation**  We use a modified version of Krissinel’s and Henrick’s CSI algorithm described in [KH04], which is in turn an improved version of Ullman’s algorithm [Ull76]. To measure performance, we randomly selected a subset of the HISPOS tables as provision. From this provision, we randomly deleted a certain percentage of tables and columns to construct a usage subgraph. Figure 11 shows the results of our measurements. On the x-axis, the number of nodes in the provision is plotted, the y-axis shows how many seconds it took to match the subgraphs. The different lines denote averages of test runs with different sized usage subgraphs. The benchmark results indicate a roughly quadratic runtime, which is the best case for the algorithm we chose. To improve performance, we can make some assumptions about the metadata graphs that are to be matched. First, there exists a set of already matched nodes to start with, namely the literals. As we required in our definitions, literal nodes are unique, and if two literal nodes in different graphs have the same value, it is always a match. Second, even if metadata graphs are not completely connected, in each part there will exist at least one literal. This is reasonable for metadata, since in general, most metadata elements, such as tables, columns, elements, etc., are named. Finally, the metadata graph is “almost a tree”, meaning that the number of relationship nodes is similar to the number of element nodes, and that there is a spanning tree representing a main hierarchy.

These assumptions are reasonable for all practically relevant metadata graphs. By exploiting them (which our current implementation does not do very well), much larger metadata graphs could be handled as it is currently the case. This is a part of our ongoing work.
Set<Node> changedNodes = getAllChangedNodes();
while (!changedNodes.isEmpty()) {
    Node v = changedNodes.removeANode();
    for (Rule r: getRules()) {
        Set<Match> M = getMatches(r, v);
        for (Match m: M) {
            Set<Node> changed = apply(r, m);
            changedNodes.add(changed);
        }
    }
}

**Figure 12: Rule matching algorithm**

**Intra-Model Analysis Implementation** We argue that most practical rules will be similar to the example in figure 9 in complexity, having one node where a change status test is made and another node where a change status is added. Our current implementation is based on this fact. The basic matching algorithm is sketched in figure 12.

First, we get the set of changed nodes as obtained through change determination or inter-model analysis. We then iterate over this node set. We take out a node \(v\) and check if there is a rule with matches containing \(v\). If so, we apply the conclusion to each match, and put all nodes that were changed by the conclusion back into the set of changed nodes. Each node can be modified at most once by each rule, which means that each node can be put back into the set at most \(|R|\) times. The while loop will thus be executed at most \(|V| \times |R|\) times. The outer for loop will be executed \(|R|\) times. Assuming rule premises consisting of three nodes like in figure 9, there will be at most in the order of \(|V|\) matches. The matching process itself has the same complexity. It makes at most two hops through the graph, and since relationship nodes always are connected to two other nodes, the search space is bound by \(|V|\). This leads to an overall complexity of our implementation of at most \(O(n^2 r^2)\) in the average case, with only simple rules. More complex rules with more than a few nodes will be more expensive to process in terms of computing time.

To measure rule matching performance, we used the HISPOS schema again. We marked one node as added, and used a set of “flooding” rules: each rule matched two nodes of any type, connected by an edge, where one node had a certain change status \(X\), and then added the status \(Y\) to the other node. The next rule would then look for nodes with status \(Y\) and add status \(Z\), and so on, so that finally, all nodes in the graph will be marked with all possible change status values. Figure 13 shows the results of the benchmarking. It shows that the time needed increases proportional to the graph size and the number of rules that are matched.

The complete HISPOS schema with 350 tables and a ruleset of 159 rules took around four minutes to be analyzed. Since this is a worst case scenario, and most times, changes will cover only a small part of a metadata graph, intra-model analysis can be done in an interactive fashion, which is important for preventive CIA.

**Deployment Effort** While the Caro model and analysis component works independent of any data model and is able to analyze arbitrary metadata graphs, the effort to extract and transform metadata into the graph model must be considered, too. Although discussing this issue is not the focus of this paper, we want to share some experiences that we got during the implementation of the prototype. Our SQL metadata importer prototype took us about
four man days to write, and is currently capable of extracting tables, views, columns, types, primary and foreign keys, check constraints as well as view and constraint dependencies out of an SQL information schema as defined in the SQL:2003 standard [ANS03]. This can serve as a very approximate measure of the effort it takes to extract and transform other metadata, like XML schemas, WSDL files, etc. Import components only have to be written once per data model/system type. Ideally, vendors would directly supply MDAs for their products, which is of course not possible for legacy or home-brew systems. Here, a decision has to be made between extracting and transforming the metadata at a fine-grained level, and therefore getting good analysis results, or to be satisfied with a coarse-grained metadata description and a loss of analysis precision. The trade-off is between putting much initial effort into the MDAs, or to later put more effort in detecting the exact problem causes.

5 Conclusion and Outlook

The research area of autonomic computing [KC03] has become increasingly important over the last years. Most times, the scope is local to a single system, or a group of tightly coupled systems, for load balancing or disaster prevention. While the approach we presented in this paper does not directly enable self-* in a system, it contributes to autonomic computing at a larger scale. It provides a mechanism for communication between systems, so each system is aware of changes in its surroundings, which in turn enables the systems to act more autonomically.

Caro performs change impact analysis by storing metadata and analyzing the impact of metadata changes. This happens in a generic way, such that arbitrary data models are supported, ranging from SQL or XML schemas over configuration files, directory structures, APIs to non-functional information like quality or performance assertions. No assumptions are made about any processes for preventive change impact analysis, and at least reactive CIA can always be performed. Our approach also works in a reasonable way if
the provided metadata is incomplete, accepting a higher rate of false positives as trade-off. This is an important fact, because in most real-world scenarios it is not feasible to provide complete metadata descriptions. Thus, our approach is not only generic, but also adapts to the environment where it is deployed. The better the provided input data, the better the analysis results will be, but even with bad input, Caro is able to conduct CIA on a best-effort basis.

We gave a formal description of the our model and analysis process, showing that model and analysis process are well defined. Analysis has shown to be deterministic and computable, which is very important for change impact analysis. The performance benchmarks we conducted show that our approach performs well for reasonable sizes of metadata graphs, and that intra-model analysis is fast enough to be used in an interactive scenario for preventive CIA. Finally, Caro is extensible at all points: Node types, rules and specific change statuses can be added to the model as needed, if the built-in constructs are not sufficient in a certain case.

There are, however, some areas which need to be researched further. We need to look for a way to gather and add change provenance information to the graph to further improve the automated support given by Caro. Also, performance of inter-model analysis could be increased by taking into account more the regular graph structure of metadata graphs. Another important task is to provide interfaces to other approaches which can be used in a complementary way: The results Caro provides can be used in schema evolution tools to adapt systems to the new situation. On the other hand, information gathering tools could be used to help Caro getting the metadata from the systems and watching for changes (e.g., [SSKS95, MAL+05]). This way, we think that our approach can act as a framework binding together many research efforts that often only work in a very constrained environment.

We stated in section 4 that the analysis works fast enough to be used in an interactive fashion. Of course, nobody is able to work interactively with a 45000 node graph. We currently work on an abstracting graph editor which is able to provide a natural (e.g., SQL DDL statements) way of editing a metadata graph. For this, we use style sheets for specifying how the graph should be presented to the user.

References


[HAS] HISPOS-GX. http://www.his.de/Abt1/HISPOS.


Entwurf von Informationsintegrationssystemen auf der Basis der Merkmalsmodellierung

Susanne Busse  Johann-Christoph Freytag
Technische Universität Berlin Humboldt-Universität zu Berlin
Computergestützte Informationssysteme Institut für Informatik
sbusse@cs.tu-berlin.de freytag@dbis.informatik.hu-berlin.de


1 Einleitung

Informationssysteme setzen meist auf eine Reihe verschiedener autonomer und damit heterogener Datenquellen auf, deren Daten gesammelt und integriert werden müssen. Wir bezeichnen diese Systeme als Informationsintegrationssysteme (IIS). Beim Entwurf eines IIS müssen sowohl die gewünschten funktionalen und nicht-funktionalen Anforderungen späterer Nutzer also auch die Charakteristika der zu integrierenden Daten und Datenquellen berücksichtigt werden. Die dabei auftretende Vielfalt führt zu einem entsprechend großen Spektrum existierender IIS, das von föderierten Datenbanksystemen bis hin zu Suchmaschinen reicht, die auf sehr unterschiedlichen Integrationsansätzen beruhen.

Eine Hilfestellung zur Orientierung bieten Diskussionen wichtiger Entscheidungskriterien und 'Trade-Offs'. Als Beispiele seien hier nur die Diskussionen bzgl. semantischer Heterogenität in [Hu97], bzgl. des Umgangs mit der unterschiedlichen Strukturiertheit der unterliegenden Daten in [Ha03], bzgl. von Qualitätsaspekten in [BP04] und [NFL04] oder die Klassifikation in [PBC00] genannt. Neben der Betrachtung spezifischer Aspekte haben sich außerdem verschiedene Typen von IIS herausgebildet, deren Beschreibung zur Wahl eines geeigneten Integrationsansatzes herangezogen werden, etwa mediatorbasierte Informationssysteme ([Wi97]), Data Warehouses ([In96]), Peer Data Management Systeme ([Ha03]), Portale ([Ma02]) oder Information Retrieval (IR)-basierte Systeme wie Suchmaschinen ([Ar01]). In [LC03] werden IIS im Bereich der Bioinformatik diskutiert, was
einen Eindruck der vielfältigen Herausforderungen beim Entwurf eines IIS gibt.

Die genannten Arbeiten bieten zwar einen guten Ausgangspunkt zur Orientierung beim Entwurf eines IIS, ihre Einbindung in den Entwicklungsprozess ist jedoch unzureichend. Weder existiert eine gemeinsame Basis, die die für IIS relevanten Kriterien systematisiert, den verschiedenen Phasen innerhalb des Entwicklungsprozesses zuordnet, noch kann eine adäquate Dokumentation getroffener Entscheidungen erfolgen, die Entwurf und Anforderungsspezifikation in Bezug setzen und so die Nachvollziehbarkeit des Entwurfs gewährleisten würde.

Genau an dieser Stelle setzt unsere Arbeit an. Wir zeigen in diesem Papier, wie die Merkmalsmodellierung für den Entwurf von IIS benutzt werden kann. Die Merkmalsmodellierung, eingeführt im Rahmen der FODA-Methode\(^1\) ([Ka90]), wird im Bereich von Produktlinien ([CN01]) und der generativen Softwareentwicklung ([CE00]) verwendet. Ein Merkmalsmodell beschreibt die variablen und gemeinsamen Merkmale der Produkte einer Produktlinie oder Systeme einer Systemfamilie. Das Modell kann zur Anforderungsspezifikation eines zu entwickelnden Produkts verwendet werden und ist Basis für die anschließende Generierung eines Entwurfs oder einer konkreten Konfiguration des Produkts entlang der für die verschiedenen Typen von IIS definierten Referenzarchitekturen.

Dieses Vorgehen lässt sich im Rahmen der modellgetriebenen Softwareentwicklung in jeder Phase einsetzen: Das Merkmalsmodell beschreibt die Möglichkeiten der in der jeweiligen Phase zu treffenden Entscheidungen und steuert die Richtung des Übergangs in die nächste Entwicklungsphase. Es dient damit der Verknüpfung von Anforderungen bzw. Entscheidungsalternativen und der späteren Lösung. Entgegen der bisherigen textuellen Dokumentation bietet die formale Basis und Werkzeugunterstützung der Merkmalsmodellierung so einen erheblich besseren Rahmen in Bezug auf Nachvollziehbarkeit und Änderbarkeit von Softwarelösungen.

Wir wollen hier konkret die Wahl eines passenden Typs von IIS im Rahmen des Entwurfs betrachten, indem eine Unterstützung bzgl. folgender Fragestellungen angeboten wird:

1. Welcher der bekannten Integrationsansätze bietet sich für das geplante IIS an?
2. Welche spezifischen Herausforderungen müssen noch gemeistert werden?


---

1. FODA = Feature-Oriented Domain Analysis
Beispiel: Das 'European Migration Network'

Der Aufbau des 'European Migration Network' (EMN)\(^1\) wurde von der Europäischen Kommission initiiert mit dem Ziel, die Zusammenarbeit von Organisationen und Individuen im Bereich Migration und Asyl auf europäischer wie nationaler Ebene zu unterstützen. In diesem Kontext wurde 2004/05 ein Informationssystem entwickelt, das die Daten der sogenannten 'National Contact Points' (NCPs) der beteiligten Staaten integriert, um eine vergleichbare und umfassende Informationsbasis für den Bereich zu schaffen.\(^2\) Das Informationssystem umfasst Kontaktdaten der Mitglieder des Netzwerks, Publikationen, Gesetzestexte sowie Urteile, Statistiken, Presseinformationen sowie Beschreibungen relevanter Informationsquellen, etwa online verfügbarer Bibliothekskataloge. Eine detailliertere Darstellung ist in [Bu05] zu finden.

Das EMN-Informationssystem bietet sich hier als Beispiel an, da sowohl eine Umsetzung als mediatorbasiertes IIS als auch als IR-basiertes IIS denkbar ist: Ein mediator-basiertes System kommt der geforderten engen Integration entgegen, wohingegen ein IR-basiertes System im Sinne einer Suchmaschine beste Möglichkeiten bietet, der Heterogenität seitens der NCPs zu begegnen.

2 Merkmalsmodellierung


2.1 Grundlagen der Merkmalsmodellierung


Ein Merkmalsmodell besteht aus einem Diagramm, das die Merkmale hierarchisch anordnet, und einem Glossar, das die Merkmale dokumentiert. Abbildung 1 zeigt ein Beispiel eines Merkmalsdiagramms in der hier verwendeten, an [Ka90] angelehnten Notation. Die hierarchische Beziehung zwischen Merkmalen spiegelt eine Verfeinerung der Beschrei-

Wir werden jedoch ausschließlich die Basiskonzepte der Merkmalsmodellierung verwenden, um den eigentlichen Fokus der Merkmalsmodellierung zu verdeutlichen: die Beschreibung von Variabilität, d.h. die Beschreibung der möglichen Varianten in der betrachteten Domäne. Variabilität findet sich in einem Merkmalsdiagramm in zweifacher Form: Jedes Merkmal beschreibt eine gemeinsame Eigenschaft (ist obligatorisch) oder eine optionale Eigenschaft. Zum anderen können Merkmale zu Alternativen gruppiert werden, wobei xor- und or-Gruppen unterschieden werden. Neben diesen direkt im Diagramm ausgedrückten Abhängigkeiten zwischen den Merkmalen können Abhängigkeiten auch zusätzlich textuell ergänzt werden, wobei insb. zwei Beziehungen betrachtet werden: Requires beschreibt die Abhängigkeit eines Merkmals von einem anderen, Excludes den gegenseitigen Ausschluss.

Abb. 1: Beispiel eines Merkmalsdiagramms ([Ka90])


Betrachten wir insbesondere den Prozess der Konfiguration, lässt sich ein Merkmalsmodell daher auch dahingehend interpretieren, dass es eine Menge möglicher Konfigurationen beschreibt, wobei jede Konfiguration die in dem Merkmalsmodell definierten Bedin-

Wenden wir die Merkmalsmodellierung auf Informationsintegrationssysteme an, so werden wir die Klasse von IIS als ein Merkmalsmodell beschreiben, spezifische Typen von IIS, wie mediatabasierte IIS, als Spezialisierung davon und die vollständige Beschreibung eines existierenden oder geplanten IIS als Konfiguration. Um die Beschreibung eines IIS-Typs noch etwas feiner angeben zu können, erweitern wir dabei die Charakterisierung eines Merkmals um die Angabe ’common’ (üblich) bzw. ’uncommon’ (unüblich), um besonders häufige bzw. seltene Merkmale zu identifizieren.

2.2 Formalisierung von Merkmalsmodellen

Ein Merkmalsmodell definiert hierarchisch angeordnete Merkmale eines Konzepts sowie deren Variabilität. Entsprechend definieren wir ein Merkmalsmodell \( FM \) als ein Tupel

\[
FM = (\text{Concept}, \text{Features}, \text{FeatureGroups}, \text{name}, \text{subfeatureOf}, \text{memberOf}, \text{variability}, \text{groupVariability}, \text{Constraints})
\]

mit der in der folgenden Tabelle beschriebenen Bedeutung.

<table>
<thead>
<tr>
<th>Element</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept</td>
<td>ein Konzeptelement, das die beschriebene Familie bezeichnet – die Wurzel des Diagramms</td>
</tr>
<tr>
<td>Features = ( \text{SolitaryFeatures} \cup \text{GroupedFeatures} )</td>
<td>eine Menge von Merkmalen – alleinstehende (wie Air conditioning) oder gruppierte (Automatic, Manual)</td>
</tr>
<tr>
<td>FeatureGroups</td>
<td>eine Menge von Merkmalen</td>
</tr>
<tr>
<td>name: Features ( \cup { \text{Concept} } \rightarrow \Sigma^+ )</td>
<td>zur Beschreibung des Namens der Elemente</td>
</tr>
<tr>
<td>subfeatureOf: Features ( \rightarrow \text{Features} \cup { \text{Concept} } )</td>
<td>zur Ermittlung des Vaters eines Merkmals</td>
</tr>
<tr>
<td>memberOf: GroupedFeatures ( \rightarrow \text{FeatureGroups} )</td>
<td>zur Ermittlung der Merkmalsgruppe eines gruppierten Merkmals</td>
</tr>
<tr>
<td>groupVariability: FeatureGroups ( \rightarrow { \text{or}, \text{xor} } )</td>
<td>zur Definition der Variabilität einer Merkmalsgruppe</td>
</tr>
<tr>
<td>variability: Features ( \rightarrow { \text{mandatory}, \text{optional} } )</td>
<td>zur Definition der Variabilität eines Merkmals</td>
</tr>
<tr>
<td>Constraints = ImplicitConstraints ( \cup \text{ExplicitConstraints} )</td>
<td>der Menge von Konfigurationsbedingungen, die explizit angegeben wurden oder sich implizit aus dem Merkmalsdiagramm ergeben</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Funktionen zur Abbildung der Merkmalshierarchie</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \text{name: Features} \cup { \text{Concept} } \rightarrow \Sigma^+ ]</td>
</tr>
<tr>
<td>[ \text{subfeatureOf: Features} \rightarrow \text{Features} \cup { \text{Concept} } ]</td>
</tr>
<tr>
<td>[ \text{memberOf: GroupedFeatures} \rightarrow \text{FeatureGroups} ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variabilitätsüberlegungen und dadurch beschriebene Konfigurationsbedingungen</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \text{groupVariability: FeatureGroups} \rightarrow { \text{or}, \text{xor} } ]</td>
</tr>
<tr>
<td>[ \text{variability: Features} \rightarrow { \text{mandatory}, \text{optional} } ]</td>
</tr>
<tr>
<td>[ \text{Constraints} = \text{ImplicitConstraints} \cup \text{ExplicitConstraints} ]</td>
</tr>
</tbody>
</table>

Tabelle 1: Formalisierung von Merkmalsmodellen
Konfiguration und Spezialisierung

Eine **Konfiguration** bezieht sich auf ein Merkmalsmodell. Es benennt das konkret beschriebene Produkt (die konkrete Ausprägung des Konzepts oder der Wurzel des Merk-
malsmodells) und gibt eine Bindung für die Merkmale des Merkmalsmodells an (wobei
natürlich die durch das Merkmalsmodell beschriebenen Constraints erfüllt sein müssen).
Formal beschreiben wir eine Konfiguration \( \text{Config} \) eines Merkmalsmodells \( FM \) als ein Tupel
\[
\text{Config} = (\text{Concept}, \text{name}, FM, \text{binding})
\]

wobei \( \text{Concept} \) wiederum ein Konzeptelement ist, \( \text{name} \) die namensgebende Funktion, \( FM \)
ein Merkmalsmodell mit den Merkmalen \( \text{Features} \) und \( \text{binding} \) eine Funktion
\[
\text{binding}: \text{Features} \rightarrow \{ \text{bound}, \text{removed} \}
\]
die angibt, ob ein Merkmal bei der Konfiguration gebunden oder gelöscht wurde.

Analog lässt sich eine **unvollständige Konfiguration** \( \text{IncompleteConfig} \) definieren als
\[
\text{IncompleteConfig} = (\text{Concept}, \text{name}, FM, \text{incompleteBinding})
\]
wobei lediglich die bindende Funktion auch ein Offenlassen der Bindung erlaubt:
\[
\text{incompleteBinding}: \text{Features} \rightarrow \{ \text{bound}, \text{removed}, \text{undecided} \}
\]

Eine **Spezialisierung** \( \text{Specialization} \), definiert als ein Tupel
\[
\text{Specialization} = (\text{Concept}, \text{name}, FM, \text{specBinding})
\]
lässt wiederum als Erweiterung der bindenden Funktion auch die Spezifikation von ‘common‘ bzw. ‘uncommon‘-Merkmalen zu:
\[
\text{specBinding}: \text{Features} \rightarrow \{ \text{bound}, \text{common}, \text{uncommon}, \text{removed}, \text{undecided} \}
\]

### 3 Klassifikation von Informationsintegrationssystemen

Die hier vorgestellte Klassifikation von Informationsintegrationssystemen (IIS) versteht
sich nicht als neue Definition der Merkmale von IIS, sondern stützt sich auf existierende
Klassifikationen und Diskussionen spezifischer Aspekte. Insbesondere seien dabei die ein-
führende Diskussionen in [LC03] sowie die Klassifikation wissensbasierter IIS in
[PBC00] genannt. Diese führt ebenfalls eine Präzisierung von Eigenschaft ein, die einem
Merkmsmodell ähnelt, bezieht sich allerdings nur auf einen Teil der hier betrachteten
IIS. Bezuglich spezifischer Aspekte sind insbesondere die Übersicht in [Hu97] sowie die
Diskussion von Datenqualitätsaspekten ([SMB05]) zu nennen.

Ebenfalls zu erwähnen ist, dass die Klassifikation nicht alle Aspekte von IIS detailliert
abbilden soll, sondern auch eine Strukturierung von Eigenschaften hinsichtlich ihrer Rele-
vanz im Rahmen des Entwicklungsprozesses eines IIS anstrebt. Die hier vorgestellte Klas-
sifikation konzentriert sich daher auf die Eigenschaften, die für die Wahl eines bestimmten
Typs von IIS relevant sind. Dies sind insbesondere die für den Nutzer sichtbare Funktionalität und Qualität bei der Informationssuche sowie die Merkmale der dem IIS zugrundeliegenden Daten und Datenquellen.

Andere Merkmale, etwa die Charakterisierung einer Materialisierung von Daten zur Beschleunigung der Anfragebearbeitung oder die Verfeinerung der Suchfunktionalität hinsichtlich der Anfragesprachen, die speziell bei mediatorbasierten IIS zum Einsatz kommen (vgl. etwa eine entsprechende Klassifikation in [Wa01]), würden Bestandteil einer Verfeinerung für eine spätere Entwicklungsphase sein. Auch die Beschreibung der hier exemplarisch betrachteten Typen – mediator-basierte und IR-basierte IIS – sowie die Anforderungen des EMN-IIS werden anschließend auf dieser Abstraktionsebene diskutiert.

3.1 Informationsintegrationssysteme im Allgemeinen

Da die Beschreibung des gesamten Merkmalsmodells für IIS den Rahmen dieses Papiers sprengen würde, geben wir im folgenden einen Überblick über das gesamte Modell und beschreiben dann detaillierter die Elemente, die sich auf die Funktionalität der Informationssuche beziehen.

Für ein Informationsintegrationssystem sind die folgenden Unterscheidungsmerkmale von besonderer Bedeutung und daher Bestandteil des Merkmalsmodells (vgl. Abbildung 2):


- Welche Form der Interaktion steht dem Nutzer für die Informationsermittlung zur Verfügung? Wir unterscheiden hierbei zunächst Systeme, die eine Navigationsmöglichkeit bieten von solchen, die eine Schnittstelle zur Suche anbieten. Bei der Suche sind insbe-
sondere Stichwortsuchen von Anfragen in einer Anfragesprache, wie SQL, zu unter-
seiden. Letztere basieren stets auf einem Schema, das das Vokabular für die Suche
definiert. Im Falle der Stichwortsuche muss dies nicht der Fall sein: Suchmaschinen
etwa unterstützen in der Regel ausschließlich eine Suche mit freien Stichworten. Für
Systeme, die eine Suchmöglichkeit bieten, sollten die Semantik der Suche sowie das zu
erwartende Suchergebnis genauer charakterisiert werden. Wir werden auf diese Ele-
mente später detaillierter eingehen.

• Wird ein kontrolliertes Vokabular für die Suche verwendet, kann weitergehend unter-
schieden werden, in welcher Form dies definiert ist. Die hier vorgenommene Unter-
scheidung lehnt sich an das in [Mc02] definierte Ontologie-Spektrum an.

Abb. 2: Merkmalsmodell von IIS – Überblick
• Wie erfolgt die *Integration* von Daten verschiedener Quellen? Dabei wird zunächst eine einfache Sammlung von Daten (wie sie etwa bei Suchmaschinen erfolgt) und die Integration mit der Fusion und Verknüpfung von Datenobjekten (z.B. in mediatorbasierten IIS) unterschieden. Bei der Integration kann weitergehend charakterisiert werden, inwieweit Konflikte bei der Integration betrachtet und ggf. gelöst werden.

• Welche *Qualität* haben die vom IIS gelieferten Daten und das Integrationssystem selbst? Da die Qualität eine herausragende Bedeutung für die Bewertung des IIS insgesamt hat, wurde dieser Aspekt separat aufgenommen. Es werden hier die für den Nutzer relevanten Qualitätsaspekte beschrieben, wobei verschiedene existierende Klassifikationen herangezogen werden können. Wir haben uns zunächst auf die in [SMB05] diskutierten Qualitätsattribute gestützt.

• Welche Merkmale haben die *Integrationskomponente* sowie die *datenbereitstellenden Komponenten*? Hierbei spielen insbesondere die Arten der Schnittstelle der Datenquellen sowie die von ihnen gelieferten Daten eine Rolle.

Abb. 3: Merkmalsmodell von IIS – Daten

Bei der detaillierteren Betrachtung der Suche sollte zunächst die Semantik der Suche charakterisiert werden, wobei folgende Varianten auftreten können (vgl. Abbildung 4):

• Manche Systeme führen eine *Modifikation der Anfrage* durch, um die Genauigkeit oder den *Recall* der Anfrage zu erhöhen. Auch eine Personalisierung durch Berücksichtigen von Kontextwissen über den Nutzer sind dabei denkbar.

• Der *Suchraum* kann sich stark unterscheiden: So decken Web-Suchmaschinen grundsätzlich nur einen unvollständigen Teil des Webs ab, wohingegen mediatorbasierten IIS potentiell alle Datenquellen berücksichtigen.

• Ein weiteres Merkmal ist die Art, wie der Vergleich von Datenbasis und Anfrage erfolgt: ist dabei eine exakte Übereinstimmung notwendig oder lediglich eine angemessene Nähe der Datenwerte? Diese Unterscheidung ist sowohl in syntaktischer wie auch semantischer Richtung sinnvoll. Information Retrieval Methoden auf der Basis des Vektorraummodells führen bspw. eine semantisch approximierte Suche durch.

• Das Ergebnis einer Suche kann weitergehend in der Ergebnismenge (vollständig, die top-k Ergebnisse oder unvollständig?) sowie in der Anordnung der einzelnen Ergebnisse unterschieden werden.
Für den Nutzer unmittelbar sichtbar ist das Suchergebnis selbst, wobei wir zum einen charakterisieren, welche Art von Daten er dabei erhält und zum anderen, in welcher Form die Ergebnismenge repräsentiert ist.

Abb. 4: Merkmalsmodell von IIS – Suche

3.2 Spezielle Typen von Informationsintegrationssystemen

Von den eingangs genannten Typen von IIS betrachten wir exemplarisch mediatorbasierte und Information Retrieval-basierte Systeme, da sie sich bzgl. ihrer Eigenschaften bei der Suchfunktionalität stark unterscheiden.

Ein mediatorbasiertes Informationssystem ([Wi97], [Kn01], [Bu02]) bietet einen integrierenden lesenden Zugriff auf eine dynamisch änderbare Anzahl heterogener Datenquellen. Die Integration ist für den Nutzer transparent – der Mediator übernimmt die Ermittlung benötigter Daten aus den Datenquellen und die quellübergreifende Integration. Dem Nut-
zer wird ein globales Schema sowie eine Anfragesprache zur Verfügung gestellt. Mediatorbasierte IIS verfolgen einen virtuellen Integrationsansatz, d.h. es wird erst bei der Anfragebearbeitung auf die Datenquellen zurückgegriffen. Zur Integration werden Wrappers eingesetzt, die die Datenquellen kapseln und für die Lösung technischer und logischer Heterogenität verantwortlich sind. Sie bieten dem Mediator in der Regel eine eingeschränkte Anfrageschnittstelle, die bei der Integration spezifiziert und bei der Anfragebearbeitung des Mediators ausgewertet wird.

<table>
<thead>
<tr>
<th>Mediatorbasiertes IIS</th>
<th>IR-basiertes IIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Interface</td>
<td>Search Interface</td>
</tr>
<tr>
<td>Mediator</td>
<td>Index-based Query Processor</td>
</tr>
<tr>
<td>Wrapper</td>
<td>Wrapper</td>
</tr>
<tr>
<td>Wrapper</td>
<td>Wrapper</td>
</tr>
<tr>
<td>Proprietary Interface</td>
<td>Web Source Interface</td>
</tr>
<tr>
<td></td>
<td>DB Interface</td>
</tr>
</tbody>
</table>

Abb. 5: Mediator- und IR-basierte IIS


Abbildung 5 zeigt den typischen Aufbau mediatorbasierter und IR-basierter IIS im Vergleich. Eine präzise Beschreibung, etwa mit Hilfe der UML, bildet die Basis der Referenzarchitektur, die als Entwurfsgrundlage bei Wahl des jeweiligen Typs generiert werden würde. Für die Wahl ist die Definition entlang des Merkmalsmodells für IIS erforderlich. Im folgenden werden die charakteristischen Eigenschaften präzisiert. Es wird dazu die Bindung bei der jeweiligen Spezialisierung angegeben. Nicht aufgeführte Merkmale bleiben unverändert ("undecided").

202
3.3 Anforderungsspezifikation für das 'European Migration Network'


Die folgende Tabelle zeigt die wesentlichen Anforderungen an das IIS des EMN. Dem Nutzer soll eine Stichwortsuche angeboten werden, wobei auch auf ein global vorhandenen Thesaurus zurückgegriffen werden kann. Neben der Stichwortsuche soll eine Navigationsmöglichkeit über Länder und Informationstyp (Kontakt, Publikation etc.) angeboten werden. Die Suche soll alle passenden Informationen in vergleichbarer Form ermitteln. Daher wird auf eine Integration (nicht nur Sammlung) der Informationsobjekt wert gelegt, bei der insb. mit der Multilingualität umgegangen werden muss. Die Daten (wie übrigens auch die Datenquellen) sind sehr heterogen. Es sind im EMN sowohl strukturierte Daten als auch unstrukturierte Dokumente vorhanden.

<table>
<thead>
<tr>
<th>Mediatorbasiertes IIS</th>
<th>IR-basiertes IIS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Offered Data</strong></td>
<td><strong>Data Type:</strong></td>
</tr>
<tr>
<td>&gt; Unstructured</td>
<td>&gt; Structured</td>
</tr>
</tbody>
</table>

**Access Interface – Interaction Model**

| bound | bound: Search – Query – Ad hoc | bound: Search – Keyword Search – Ad hoc |

**Vocabulary Definition**

| bound | bound: Global, removed: Local Representation: |
| common: Data Object Net | |

| bound | optional |
| bound: Global, removed: Local |

**Search Semantics**

| Search String Modification | uncommon | optional |

| Search Space | bound: Complete |
| uncommon: Restrictable |

| bound | uncommon: Complete |
| bound: Complete |

| bound | uncommon: Syntactically Approximated |
| bound: Complete |

**Matching**

| bound | common: |
| bound: Semantically Exact |

| uncommon: Syntactically Approximated |

| bound: Common: |
| bound: Semantically Approximated |

**Result Extension**

| bound | bound: Complete |
| common: Top-k |

| bound | removed: Ranked |
| uncommon: Sorted – Pre-defined |

| bound: Ranked |

| bound: Bound |
| bound: Result Kind – Domain Data |

| bound: Result Kind – References |

| common: Representation – List |

Table 2: Definition mediator- und IR-basierter IIS
4 Merkmalsbasierte Wahl eines Integrationsansatzes

4.1 Vergleich von Merkmalspezifikationen

Die Wahl eines Integrationsansatzes und der damit verbundenen Referenzarchitektur soll durch Vergleich einer Anforderungsspezifikation mit den Merkmalspezifikationen verschiedener Typen von IIS unterstützt werden. Allgemein wird dabei eine Matching-Funktion

\[
\text{match (FM-Spec, FM-Conf): record (set (MatchResult), Similarity)}
\]

verwendet. Sie erhält als Parameter

- eine Spezialisierung \( \text{FM-Spec} \) eines Merkmalsmodells \( \text{FM} \) (im Falle von IIS eine Spezialisierung, die einen bestimmten Typ von IIS beschreibt) und
- eine Anforderungsspezifikation \( \text{FM-Conf} \), d.h. eine ggf. noch unvollständige Konfiguration des gleichen o.g. Merkmalsmodells \( \text{FM} \) (im Falle von IIS etwa die Anforderungsspezifikation für das EMN-Informationssystems)

und liefert als Ergebnis

- eine Menge von Vergleichsergebnissen \( \text{MatchResult} \) sowie
- eine quantifizierte Aussage \( \text{Similarity} \) zur Ähnlichkeit mit einem Wert \( \in [-1, 1] \)

Der Matchingalgorithmus besteht im wesentlichen aus zwei Teilen: Zum einen aus dem direkten Vergleich der Bindung eines Merkmals, aus dem auch ein detailliertes Vergleichsergebnis \( \text{MatchResult} \) resultiert, zum anderen aus der Traversierung des
Vergleich der Bindung eines einzelnen Merkmals


Tabelle 4: Vergleich von Merkmalen aus Anforderungsspezifikation und Spezialisierung

<table>
<thead>
<tr>
<th>Anforderungsspez. Spezialisierung</th>
<th>bound</th>
<th>removed</th>
<th>undecided</th>
</tr>
</thead>
<tbody>
<tr>
<td>bound</td>
<td>++ (Match)</td>
<td>-- (Mismatch)</td>
<td>-- (Potential Mismatch)</td>
</tr>
<tr>
<td>common</td>
<td>+(Common)</td>
<td>--</td>
<td>-- (Potential Uncommon)</td>
</tr>
<tr>
<td>undecided in changed FeatureGroup</td>
<td>-- (Uncommon)</td>
<td>+ (Common)</td>
<td>--</td>
</tr>
<tr>
<td>undecided</td>
<td>+ (Possible)</td>
<td>+ (Possible)</td>
<td>+ (Possible)</td>
</tr>
<tr>
<td>uncommon</td>
<td>-- (Uncommon)</td>
<td>+ (Common)</td>
<td>-- (Potential Uncommon)</td>
</tr>
<tr>
<td>removed</td>
<td>-- (Mismatch)</td>
<td>++ (Match)</td>
<td>-- (Potential Mismatch)</td>
</tr>
</tbody>
</table>

Zu beachten ist die unterschiedliche Bewertung des 'undecided’ bei gruppierten Merkmalen einer Spezialisierung, d.h. in der Definition eines Typs von IIS: Wurde kein Merkmal der Gruppe bei der Spezialisierung als häufig oder immer auftretendes Merkmal ausgewählt, so gehen wir davon aus, dass alle Alternativen gleichermaßen unterstützt werden. So unterscheidet ein MBIS bspw. nicht, ob in dem System Daten oder Metadaten als 'Domain Data’ verwendet werden. Wurde hingegen eines der Merkmale bevorzugt (bound oder common), so bedeutet dies in der Regel, dass die anderen noch möglichen Merkmale zwar denkbar sind, aber nicht gesondert unterstützt werden. Die Wahl ist daher eher ungewohnlìch (Ergebnisklasse ’Uncommon’). So ist es in einem IR-basierten IIS zwar möglich auch eine Navigationsmöglichkeit zum Datenbestand anzubieten (’Browse & Fetch’), aber nicht durch die für ein IR-basiertes IIS charakteristischen Konzepte.

Für die definierten Ergebnisklassen ist auch eine Quantifizierung als Wert zwischen -1 und +1 definiert (vgl. Tabelle 5), die später die Basis für die Nähe (**Similarity**) bildet und die
Passgenauigkeit des Integrationsansatzes quantifiziert. Potenzielle Konflikte, die sich aufgrund einer noch unvollständigen Anforderungsspezifikation ergeben, werden dabei von der Betrachtung ausgeschlossen.

### Tabelle 5: Quantifizierter Vergleich von Merkmalen

<table>
<thead>
<tr>
<th>Match</th>
<th>Uncommon</th>
<th>Potential Uncommon</th>
<th>Mismatch</th>
<th>Potential Mismatch</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ (Match)</td>
<td>-1</td>
<td>(Potential Uncommon)</td>
<td>-1</td>
<td>(Potential Mismatch)</td>
</tr>
<tr>
<td>+ (Common)</td>
<td>+0.5</td>
<td>(Mismatch)</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>+ (Possible)</td>
<td>+0.5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Vergleich des gesamten Merkmalsmodells**

Der direkte Vergleich zweier Bindungen eines Merkmals bildet die Basis des Matchingalgorithmus. Der Gesamtalgorithmus bestimmt, welche Merkmale verglichen werden und welches Maß für quantifizierte Angaben sinnvoll ist. Das Ergebnis über die Ähnlichkeit von Anforderungsspezifikation (einer ggf. unvollständigen Konfiguration) und der Definition eines IIS-Typs (definiert als Spezialisierung im Sinne der Merkmalsmodellierung) beinhaltet eine Sammlung detaillierter **MatchResults** sowie eine quantifizierte Angabe der Ähnlichkeit. Dieser Wert liegt wiederum zwischen -1 und +1. Ein negativer Wert bedeutet, dass der Integrationsansatz eher ungeeignet, ein positiver, dass er geeignet ist.

Der Matchingalgorithmus beinhaltet grob die im folgenden beschriebenen Schritte (vgl. die Beschreibung in Pseudocode in Abbildung 6, die auf der Formalisierung von Merkmalsmodellen in Kapitel 2 beruht):

1. Startpunkt ist die Wurzel der Anforderungsspezifikation, die im folgenden traversiert wird. Die Orientierung an der Anforderungsspezifikation ermöglicht den Umgang mit unterschiedlich weit fortgeschrittenen Konfigurationen.
2. Für jeden Knoten wird die Ähnlichkeit des Teilbaumes mit diesem Knoten als Wurzel ermittelt: zum einen die Menge von detaillierten **MatchResults**, zum anderen die Quantifizierung als ein Wert zwischen +1 und -1.

Der quantitativen Berechnung liegen folgende Annahmen zugrunde:

a) Variable Eigenschaften, die in der Anforderungsspezifikation nicht betrachtet wurden (*undecided*) werden nicht berücksichtigt.
b) Die betrachteten Untermerkmale eines Merkmals gehen *gleichberechtigt* in die Berechnung ein.
c) Es wird beim Vergleich nur dann weiter ins Detail gegangen, d.h. die Untermerkmalsbäume werden nur dann weiter betrachtet, wenn der *Vergleich* des Merkmals selbst *positiv* verläuft ist. Ein negatives Ergebnis bedeutet, dass die Spezialisierung dieses Merkmals als eher untypisch definiert hat. Untermerkmale werden daher dort nicht weiter spezifiziert, so dass ein Vergleich nicht sinnvoll wäre.
Wir berechnen daher die Ähnlichkeit des (Teil)baums mit dem Wurzelmerkmal $f$ als Produkt des Einzelvergleichs von $f$ und der Summe der anteiligen Ähnlichkeit der relevanten Untermerkmalsbäume von $f$.

```plaintext
match (FM-Spec, FM-Conf):
    return match(FM-Conf.Concept, FM-Spec, FM-Conf)

match (f, FM-Spec, FM-Conf):
    // Vergleich des betrachteten Knoten: qualitativ sowie quantitativ
    nodeMatch := nodeMatch(f, FM-Spec, FM-Conf);
    nodeSimilarity := nodeSimilarity(f, FM-Spec, FM-Conf);

    // weitere Untersuchung sinnvoll? (Punkt c)
    if (nodeSimilarity < 0)
        return (nodeMatch, nodeSimilarity);
    else
        // Initialisierung für beide Ergebniselemente
        subMatches := [ ];
        sumSubSimilarity := 0;

        // detaillierte Vergleichsergebnisse für 'undecided' Merkmale sammeln
        undecidedSubfeatures := { sub ∈ Features | subfeatureOf(sub) = f ∧ incompleteBinding(sub) = undecided }
        for each sub ∈ undecidedSubfeatures do
            subMatches := subMatches ∪ nodeMatch(sub, FM-Spec, FM-Conf);

        // Traversierung der relevanten Untermerkmale (Punkt a)
        relevantSubfeatures := { sub ∈ Features | subfeatureOf(sub) = f ∧ incompleteBinding(sub) ≠ undecided }
        for each sub ∈ relevantSubfeatures do
            // Vergleich des Unterbaums
            subMatchRecord := match(sub, FM-Spec, FM-Conf);
            subMatches := subMatches ∪ subMatchRecord [ 1 ];
            sumSubSimilarity := sumSubSimilarity + (weightSubfeature * subMatchRecord [ 2 ]); // Gleichberechtigung der Unterbäume (Punkt b)

        // Gesamtergebnis ermitteln
        return (nodeMatch » subMatches, nodeSimilarity * sumSubSimilarity);
    end if;

nodeMatch (f, FM-Spec, FM-Conf)
gibt das detaillierte Vergleichsergebnis zurück, das sich aus dem direkten Vergleich von $f$ entsprechend der Tabelle 4 ergibt.

nodeSimilarity (f, FM-Spec, FM-Conf)
gibt den Ähnlichkeitswert zwischen -1 und 1 zurück, der sich aus dem direkten Vergleich von $f$ entsprechend der Tabelle 5 ergibt.
```

Abb. 6: Algorithmus zum Vergleich von Anforderungsspezifikation und Spezialisierung
Bei der Ermittlung der detaillierten Vergleichsergebnisse MatchResult gelten die gleichen Abbruchsbedingungen bei der Traversierung. Allerdings werden die Ergebnisse von nicht betrachteten Merkmalen in der Anforderungsspezifikation (‘undecided’) noch mit aufgenommen, um potentielle Konflikte zu identifizieren.

4.2 Vergleich am Beispiel des 'European Migration Networks'


Abb. 7: Beispiel eines Vergleichs


Der tatsächlich vorgenommene Entwurf des EMN-IIS folgt einer materialisierten Form der Mediation, da diese insbesondere auch den Anforderungen bzgl. der gewünschten engen Kopplung (der Integrationssemantik) entgegen kommt. Eine der größten Herausforderungen war bei der Entwicklung die Realisierung einer Stichwortsuche auf dieser Basis – wie es die hier vorgenommene Analyse auch bestätigt.

5 Diskussion

Ausgangspunkt für die hier vorgestellte Arbeit ist die Beobachtung, dass der Entwurf eines Informationsintegrationssystems (IIS) weitgehend manuell erfolgt, obwohl eine Vielzahl von Diskussionen bzgl. der ’Trade-Offs’ sowie Beschreibungen der spezifischen Merkmale und Architektur bestimmter Typen von IIS existiert. Wir haben hier gezeigt, wie die Merkmalsmodellierung aus der generativen Softwareentwicklung genutzt werden kann, um diese Lücke zu schließen. Es wird mit dem hier gezeigten Ansatz folgendes erreicht:

- Das Merkmalsmodell für IIS schafft mit der Präzisierung der Eigenschaften von IIS eine geeignete Basis sowohl für den Vergleich und die Dokumentation konkreter Systeme als auch für die präzise Abgrenzung verschiedener Typen von IIS. Es bietet eine geeignete Grundlage für eine strukturierte Anforderungspezifikation und zeigt bereits wichtige Abhängigkeiten zwischen Eigenschaften von IIS auf.


Die quantifizierte Aussage zur Ähnlichkeit kann einen ersten Hinweis auf die Passgenauigkeit eines IIS-Typs geben, sollte allerdings nicht überbewertet werden. So sind die hier vorgenommenen Annahmen, etwa über die Gleichberechtigung aller Merkmale auf allen Detaillierungsebenen, in weiteren realen Beispielen zu überprüfen und ggf. anzupassen.

Die hier begonnene Richtung zu einer durchgängigen Methode für die modellgetriebene Entwicklung und Evolution von IIS soll in folgenden Arbeiten weiterverfolgt werden. Insbesondere sollen dazu
• weitere Typen von IIS, etwa Portale und Peer Data Management Systeme, definiert werden und für eine weitergehende Analyse quantifizierter Aussagen genutzt werden;
• das Merkmalssmodell zur Unterstützung späterer Entwicklungsphasen verfeinert werden, etwa für die Wahl geeigneter Datenmodelle und Anfragesprachen, aber auch existierender Algorithmen zur Realisierung der Merkmale;
• weitere Algorithmen zur Unterstützung der Entwicklung, etwa zur Generierung eines Rahmens für die Architektur eines IIS, definiert und im Rahmen eines Werkzeugs angeboten werden.

Literaturverzeichnis


Towards an Integrated Model  
for Data, Metadata, and Operations

Jürgen Göres  
goeres@informatik.uni-kl.de
Stefan Düsloch  
düsloch@informatik.uni-kl.de

Abstract: Information integration requires manipulating data and metadata in ways that in general go beyond a single existing transformation formalism. As a result, a complete source-to-target mapping can only be expressed by combining different techniques like query languages, wrappers, scripting, etc., which are often specific to a single integration platform or vendor. Such a mapping is not portable across different alternative deployment scenarios, thus limiting the mapping’s reusability and putting the considerable investment required to create it at risk. To avoid this vendor lock-in, we define an integrated representation for operations on arbitrary data and metadata that is independent of any specific metamodel or transformation language. Using it, we can express mappings in an abstract, vendor-neutral form, improving the interoperability of integration tools and the flexibility for the deployment of mappings.

1 Introduction

The goal of information integration is to provide a homogeneous, integrated view over multiple heterogeneous data sources. To overcome heterogeneity, a mapping or integration plan has to be developed that transforms the data found in the sources into the data model, structures, and formats of a desired target schema. Requiring a thorough understanding of data source semantics, integration planning remains essentially a manual process, making it perhaps the most time-consuming and expensive aspect of information integration. Once an integration plan has been created, it has to be deployed on a runtime platform. This can be full-fledged integration middleware like a federated DBMS or an ETL tool, but can also be a generic programming tool, a stylesheet, scripts used in a hand-crafted solution, or a combination of all of the above. Together, integration plan and runtime platform constitute the integration system that provides the target schema.

Information integration subsumes numerous tasks, which have to represent and modify metadata and data in some way: Metadata and data have to be translated between different metamodels or have to be analyzed for semantical correspondencies (schema and record matching), mappings that translate data between different schemas have to be developed, etc. While powerful integration tools and information system implementations exist for specific problems, the interoperability between these systems remains limited. E.g., there is no agreed standard for the exchange of discovered matches between schema matching and mapping tools. Even worse, the mapping tools themselves are commonly built to create mappings for a single integration platform only – consequently, the resulting integration plan is specific to the platform’s internal data model (IDM) and infrastructure.
(e.g., the wrappers). This tight coupling of integration tools, integration plan, and runtime platform is often an accepted fact in the traditional integration scenarios within a single organization. Here the requirements on the integrated view, the available data sources, and the resources dedicated to the integration system are assumed to be stable. But once new requirements go beyond what the original integration platform can handle, a migration to a new platform may be unavoidable.

In dynamic, open-world integration settings such as a grid environment, such restrictions are even less acceptable. With sources and users coming from different contexts, the degree of heterogeneity between data sources and user requirements is likely to be even greater than in a static, closed-world setting. Furthermore, grid data sources remain autonomous and can join and leave the grid at any time, making the tedious and expensive manual integration approach even less feasible. To enable the use of integration technology in such scenarios, the PALADIN project (Pattern-based Approach to LAnge-scale Dynamic INformation integration) [Go05b] explores concepts and techniques to bring cost-effective services for the planning and operation of integration systems to these new environments. After an integration plan has been created, a redeployment of a grid integration system can be necessary for several reasons: Service providers compete for customers in both price and performance. Grid services – being inherently unreliable – can become unavailable. Varying loads or changes in available data replicas can also require a migration.

To gain the necessary flexibility for redeployment, both the static and the dynamic integration scenarios require an abstract, vendor-independent formalism to develop and describe mappings. This allows not only to choose the best tools for each task during development, but also gives flexibility for the deployment of the result.

In this paper, we introduce a metamodeling approach based on typed, attributed multigraphs that – unlike existing approaches – is well suited not only for the efficient representation of metadata, but also for the representation of data. This is important, as many integration tasks do not operate on metadata alone, e.g., operators that turn data to metadata or vice versa, or instance-based schema matching. We then introduce a formalism based on graph transformations that is capable of describing arbitrary operations on our data/metadata representation, as well as sequences of these operations, which we refer to as transformation plans. We will demonstrate the formalism’s practical use by describing the operational semantics of well-known data management operators and the chaining of these operators to form an abstract integration plan. We will then discuss how these abstract plans give us the desired flexibility, as they can be transformed into concrete plans for deployment into specific runtime environments.

The remainder of this paper is structured as follows: Section 2 discusses the requirements on an integrated method to represent data and metadata independently of specific metamodels and introduces the Paladin Metamodeling Architecture (PMA), our graph-based approach to satisfy these requirements. Section 3 introduces our graph transformation formalism. Section 4 classifies the types of operations it has to support. Section 5 provides an example of its use for data management. Section 6 briefly discusses how abstract integration plans can be mapped to concrete plans. Section 7 gives an overview of related work. Section 8 closes with a summary and an outlook on future work.
2 Graph-based Data and Metadata Representation

As the data/metadata representation plays a pivotal role for all subsequent operations, it has to fulfill a number of sometimes contradictory requirements: (1) It should be able to represent data and metadata naturally. This is required both as a conceptual foundation to describe the semantics of data management operations which modify this data representation (and its associated metadata), and to support model management operations, which also often depend on data samples or statistics. (2) It has to be extensible to support any existing or future metamodel, but (3) should ideally be able to express common aspects between the different metamodels, to allow their uniform handling. At the same time, the representation must be (4) lossless, i.e., preserve metamodel-specific characteristics. Finally, the representation of both data and metadata should be (5) efficient.

2.1 Paladin Metamodeling Architecture

Like similar approaches (e.g., [OMG06]), our graph model for generic data/metadata representation, the Paladin Metamodeling Architecture (PMA) (Figure 1), is based on the concept of metamodeling. The principal idea is to define a stack of (meta)layers. By instantiating elements on a layer \( M_n \), elements of the layer \( M_{n-1} \) are modeled. Most approaches use four layers: A meta-metamodel (M3), which is usually fixed, provides the central concepts for defining different (new or existing) metamodels (M2), like SQL or XML. The instances of the metamodels represent concrete application models or schemas (M1), whose instances finally represent the application data (M0). The Paladin meta-metamodel (PMM) (M3) defines all the concepts available to define concrete metamodels in a graph model: A TypeGraph consists of GraphElements, the most important being Node and EdgeType. AttributeTypes can be attached to any GraphElement, their Domain is defined by referring to one of the built-in simple types. Note that the PMM is self-defining, i.e., every PMM element is also an instance of a PMM element. The PMM can therefore be understood as a type graph for type graphs. Examples of type graphs representing concrete metamodels (the SQL and XML metamodels) are shown in Figure 1 (M2). Any existing or future metamodel can be introduced into the PMA by specifying a suitable type graph (req. 2), which can be as detailed as required for a lossless representation (req. 4).

Each metamodel’s type graph elements should inherit from the elements of the Core type graph, which capture common properties of existing data models like typing, inheritance, nesting of features or namespaces (req. 3). This later allows any model to be interpreted in terms of the Core metamodel, when model-specific aspects are not relevant. An essential differentiating feature is the availability of the PMM as a base of inheritance for concrete metamodels. Note how all elements of the Core, SQL and XML metamodels are both instances (indicated by the underlined part of their label), as well as subclasses of PMM elements (indicated in angle brackets). This allows the instances of type graphs (i.e., the concrete models or schemas, like the excerpt of a small human resources SQL schema shown in Figure 1 (M1)), to be themselves interpreted as type graphs that finally define the structure of those graphs that represent data, as illustrated in Figure 1 (M0), fulfilling
req. 1. This way we receive a model-specific data representation, which is – like that of the metadata – very efficient when compared to the generic representations found in the literature (req. 3): Schema, Table and Column are subclasses of NodeType, so their instances define nodes on the data layer M0. SQlType and its subclasses are instances of NodeType and a subclass of AttributeType. An instance of SQLType can therefore appear generically as a node in the model diagram, but also in its specific interpretation as an attribute of a node type instance to which it is connected by a hAtt edge (or one of its subclasses). The SQL schema in Figure 1 shows the two value SQLTypes in both interpretations. Instantiating the specific Table and Column instances with their attributes and the model-specific EdgeTypes between them, we can represent the actual tuples and the values of the tuple attributes. We reuse XML Schema’s [BMD4] powerful simple type system for atomic domains and for expressions and functions on them.

Figure 1: The PMA metlayer stack

2.2 Related metamodelling approaches

Existing metamodelling approaches have inspired the PMA. However, all these approaches have shortcomings, which we have addressed in PMA: The Common Warehouse Metamodel (CWM) [OMG03a] aims to ease the exchange of metadata in data warehousing scenarios. With its MOF meta-metamodel it is extensible (req. 2) and lossless (req. 4).
A Core metamodel, from which concrete metamodels inherit, captures common aspects of different metamodels (req. 3). However, to represent data, the CWM uses a generic instance metamodel. This violates the metalayer concept and is not natural (violating req. 1), because data now resides on the same layer as metadata. This also results in a very verbose data representation (see [OMG03a] 4.6), violating req. 5.

The Graph eXchange Language (GXL) [HSSW06] is primarily an XML format for the exchange of graphs between different applications. But GXL also defines the language's underlying metamodel (the GXL metaschema, M2) of directed, nested hypergraphs, with typed and attributed edges and nodes. Type graphs (M1) are used to define the valid structure of graphs, and to specify the available node and edge types with their attributes. To fulfill req. 2, type graphs have to be used to represent different metamodels. While this allows the lossless representation of metadata (req. 4), GXL type graphs cannot import and inherit from other type graphs. It is therefore impractical to define something like a Core model (not satisfying req. 3). As actual metadata already resides on M0, we cannot instantiate concrete models again, as they are not type graphs themselves. This leaves us with no further metalayer to represent data, violating req. 1. A generic representation like CWM’s instance model could be defined, which would, however, violate req. 5.

The semantic web standard RDF [MM04] provides the schema language RDFS, which allows to define complete RDF schemas for any metamodel (fulfilling reqs. 2 and 4). Arbitrary inheritance is possible between schemas, so a Core model can be defined (fulfilling req. 3). RDF(S) abandons the rigid separation of metalayers. As a consequence, an RDF Schema instance can inherit from RDF Schema constructs and can itself be interpreted as an RDF Schema. This essentially allows an arbitrary number of metalayers (fulfilling req. 1). However, RDF models tend to quickly become very complex and thus difficult to understand and modify, as RDF does not have attributes for resources and properties. While attributes for resources can be adequately represented with literal nodes, adding attributes to properties requires the cumbersome reification mechanism (violating req. 5).

3 Graph Transformations

With the generic data/metadata representation introduced in the previous section, we will now discuss how to describe operations that manipulate this representation by using a formalism based on graph transformations (GTS) (see [Hec06] for an overview, [AEH'99] for an in-depth discussion). Graph transformations have been subject to research for approximately 30 years. However, despite being a powerful tool to describe modifications of graph-like structures, they are still rarely used outside the graph community. On the one hand, this is due to the lack of an accepted standard for intuitive specification of graph transformations and differences in the operational semantics of these systems. On the other hand, few graph transformation systems (GTSs) have reached a state of maturity beyond that of research prototypes. Formalisms for specifying rules vary widely in their general approach, their formal properties and in their expressiveness (which is closely related to intellectual manageability of the resulting rule set). In this paper, it is not our intention to discuss their individual benefits and drawbacks, but to propose the use of the Paladin Graph
Transformation Algebra (GTA), our variant of *algebraic graph grammars* (AGGs), both for its expressiveness and — in our opinion — superior clarity of the resulting specification. For a comparison of the different approaches refer to [BFG96]. AGGs are a generalisation of Type-0 Chomsky grammars to non-linear structures. Like string grammars, an AGG consists of a set of *production rules*, which specify individual graph transformations.

### 3.1 Production Rule Language

To meet our specific requirements, we defined a variant of existing production rule formalisms. We will illustrate its essential semantics by means of the simple production rule shown in Figure 2. A production rule consists of a left- and a right-hand side (LHS, RHS), a *morphism* $M$ and an optional application condition (AC). The LHS consists of a graph pattern, an abstract subgraph that describes which elements (nodes and edges) have to be found in the input graph (often called *host graph*) for the production rule to be applicable. To differentiate nodes and edges in a host graph from the nodes and edges on the LHS, we refer to the latter as *pattern nodes* and *pattern edges*. In a GTS with typed graphs, the LHS elements can specify a type test. The type test is given as the part of an element's label behind a colon. A subgraph of the host graph that matches a rule's LHS is called a *witness graph* or *occurrence* of the LHS. In general, a rule can have several witness graphs in a given host graph. A witness graph can be expressed as an *occurrence morphism*, i.e., a mapping of host graph elements to the pattern elements on the LHS of the rule. Morphisms are expressed by *binding variables*. In general, occurrence morphisms can be non-injective, i.e., map a host graph element to more than one LHS element. Since arbitrary homomorphism can result in unexpected matches, groups of elements on an LHS among which general morphisms are allowed have to be explicitly specified. These groups are indicated by the homomorphic clause.

The right-hand side describes how a matched witness graph is modified by the rule. To connect LHS and RHS of a rule, we use a *rule morphism* to determine corresponding elements on the LHS and RHS, indicated by binding variables. By using non-injective rule morphisms, several LHS nodes can be merged into one, maintaining all edges that originally connected to one of the LHS nodes.

In the usual interpretation, a production rule is understood as replacing the structure in the host graph identified on the LHS with the structure found on the RHS, i.e., the host graph is modified *in-place*: Elements appearing on the LHS but not on its RHS are deleted, while elements on the RHS that have no corresponding element on the LHS are created. Since we often want to simulate algebra operators that are free of side-effects, we would have to repeat the elements of the LHS (i.e., the input for an operator) on the RHS, as the input elements (e.g., relations) would otherwise be directly modified by the transformation. This will not only lead to a lot of redundancy in rules, but also make specification of the newly created elements on the RHS more difficult. To suit this special requirement, we split the LHS elements into an *in-place* and a *copy set*. The in-place set has the usual semantics and can be used to do modifications on the host graph. The copy set indicates that corresponding elements on the RHS represent copies of this induced subgraph of the
witness graph. With the copy set, we can easily use host graph elements as templates
for new graph elements. Copied RHS elements refer to the template LHS element via a
morphism and can specify a new label. By connecting elements of the copy set to those
of the in-place set, we can indicate an embedding of created and copied elements into the
host graph.

Application conditions are a generalization of various mechanisms found in other graph
transformation languages to restrict a rule’s applicability to those subgraphs of a host
graph that meet additional structural and non-structural restrictions. An AC is essentially
a boolean expression with the usual conjunction, disjunction, and negation operators. Its
elements can be “flat” expressions on attributes of the elements identified on the LHS, as
well as graph patterns that must or must not appear together with a potential witness graph.
Morphisms are used to identify elements in these graphs with those on the LHS of a rule.

Beyond the basic elements for specifying production rules that GTA has in common with
many GTs, different language extensions can increase the degree of expressive power
of individual rules, but not of the GTA as a whole. E.g., optional elements (indicated by
dashed lines) help to reduce the number of production rules required, as they allow to cap-
ture simple variants within a single rule. Multinodes on a LHS can match a set of nodes
in the host graph and are indicated by double lines. However, the match semantics of two
multinodes connected directly by an edge are not clear. One option is to prohibit edges
between multinodes, as done, e.g., by PROGRES [Zi96]. Instead, we use cardinality con-
straints on edges to indicate whether, e.g., each of the matches to the first multinode must
have exactly one corresponding match in the other multinode, or whether they have to be
fully connected. Another problem occurring when using multinodes is binding precedence.
Often a given host graph node can be mapped to more than one multinode in a rule. By
specifying priorities for multinodes, the assignment becomes unambiguous.

3.2 Operational Semantics of Graph Transformations

So far, we have only discussed the elements used to construct individual production rules.
Another important aspect of GTs – and another aspect where they often differ consider-
ably – is their operational semantics, i.e., the sequence in which rules are applied.
The basic usage method for graph grammars is to apply the rules non-deterministically and as long as possible until no more rules are applicable. Non-determinism is caused by two aspects, (1) choosing the rule and (2) choosing a concrete LHS occurrence (i.e., a witness graph or morphism). However, for most practical applications of graph transformation systems, a way to enforce a strict or partial order in which rules are applied is needed, as well as a way to restrict the possible occurrences to certain parts of the graph. While arbitrary control flows can be enforced by adding control flow elements to the host graph, this results in a tight coupling of rule and host graphs and therefore prohibits reuse. Approaches to structuring rule application go from a simple layering to full-fledged graph programming languages.

Rules for managing data/metadata are rarely applied completely non-deterministically: Since most of the operations on data/metadata graphs will only create new elements, a rule would be applicable to the same occurrence in the host graph again and again. ACs can prohibit the rule if the new elements already exist. To express an actual use of an operator, we need a mechanism to apply rules in a strict or partial ordering, and to narrow down or determine where in a host graph they can be applied. E.g., to represent a more complex operator, it could be desirable to split its description into a strict or partial sequence of rules and connect them by passing the output of one rule as input to the next. On a coarser level, i.e., to describe an entire transformation plan, different operators (which could themselves be either atomic rules or a sequence of several rules) have to be connected in a similar fashion. Both aspects are covered by morphisms: A morphism connecting elements of a host graph with the pattern elements of a LHS pattern graph can be used to indicate complete bindings of the rule (i.e., to enforce a certain witness graph) as well as partial bindings to limit its applicability to parts of the host graph. Morphisms between one rule's RHS pattern elements and the LHS elements of another rule indicate both the sequence in which they are applied, as well as bindings between rules that indicate how the results of the first rule are to be used as input to the second. Together, production rules and their operational semantics constitute GTA, our Graph Transformation Algebra for modifying data and metadata represented using the PMA.

4 Types of Operations on Metadata and Data

Information integration deals with two general categories of operations on data/metadata: Data management operations are those classical operations that form the parts of an integration plan, e.g., SQL queries, relational algebra operator plans, wrapper configurations, stylesheets, etc. They focus on querying and modifying data and its associated metadata. Model management operations subsume those steps that aid the creation of an integration plan. We argue that our graph transformation approach is powerful enough to describe the operational semantics of any existing data or model management language in one uniform framework.

Existing data management languages are commonly mapped to or defined in terms of an underlying algebra, or have a minimal subset of elements that can be interpreted as an algebra. We will therefore use the terms algebra and language as synonyms.
The specific requirements of information integration have widened the scope and type of data management operations: One deficit of both relational algebra and SQL is the lack of operations to turn metadata into data or vice versa, i.e., to perform inter-layer transformations. Such functionality is often required during integration to resolve problems of schematic heterogeneity [BKLM99]. For example, a source table could model data category information as attribute values, while the target schema requires this information to be represented as individual columns. To resolve this, a pivot operation is needed, which is not supported in SQL, making it an intra-layer language. SchemaSQL [LSS01] addresses some of these problems by extending SQL, but lacks a proper theoretical foundation. FIRA and FISQL [WR03] continue where SchemaSQL left off and provide a complete set of data to metadata operators for RA and SQL, respectively, creating what the authors refer to as a transformationally complete algebra. An algebra for a given data model has this property, if it has the ability to transform arbitrary data into metadata, or vice versa. For the relational model, this requires to transform column and relation names to tuple values, or vice versa. FIRA and FISQL can be classified as inter-layer languages for the relational and SQL models, resp. For the XML metamodel, both XSLT and XQuery can be classified as inter-layer languages as well.

Another common problem is not solved by inter-layer languages: Often, data coming from sources of different data models has to be integrated, which requires a transformation of data and metadata between data models. Most existing approaches use a variant of the wrapper/mediator approach [Wie92]. A wrapper is a configurable software component specific to an integration platform that translates from the source’s data model to the IDM of the platform. Since the operations performed by a wrapper cannot be mapped to any data-model-specific algebra, their effects cannot be described transparently in an integration plan, but only as a black box with their configuration parameters. For some types of mappings between data models, declarative languages have been proposed, often coming as an extension to an existing language. The most prominent example is SQL/XML, which allows to create XML fragments from SQL data. Together, these languages and wrappers represent inter-metamodel data management operations.

In section 5, we will illustrate how the operational semantics of individual data management operations can be represented by using GTA production rules, and how we can use this representation in conjunction with the rule chaining mechanism introduced in section 3.2 to describe sequences of these operators, i.e., abstract transformation plans.

Recently, the area of model management, introduced by the work of Bernstein, Melnik et al. [MRB03a, MRB03b], has received increasing recognition. [MRB03b] defines logical operators (i.e., an algebra) that work on models or schemas. Besides precisely defined concrete operators like the merging of models, or the deletion or extraction of parts of a model, some abstract operators stand for a number of alternative realizations. For example, Match is an abstract operator representing different schema matching heuristics to discover semantic correspondencies. Our graph transformation formalism is able to represent both concrete operations and specific realizations of abstract operators. Schema matching is an additional example of how model management can benefit from the FMA's ability to generically represent data (either completely or aggregated). Many instance-based approaches to schema matching use statistical information on the distribution of
data values. The PMA allows instance-based matching to be done independently of the concrete implementation and metamodel of the data sources to be matched.

Going beyond the existing model management operations that can support the manual creation of integration plans, our pattern-based approach to information integration presented in [Go05a] aims at the automated creation of integration plans. It makes intensive use of graph transformations to represent both patterns and the resulting integration plans. Patterns are a machine-understandable, reusable representations of information integration problems and of knowledge on how to recognize and resolve them.

5 Graph Transformations for Generic Data Management

We will now demonstrate the use of GTA with a small example scenario. We will use GTA to describe (sequences of) production rules that represent operators and sequences of these operators, i.e., transformation plans. Figure 3 shows the schema and data of a small human resources database HR, with tables for employees and departments. Consider an application that needs an integrated relational view on this data source and has some specific requirements regarding its schema: The information about an employee’s department should be combined into a single table row with his or her other data. The combined employee-department information should then be divided into individual tables for each city that is the location of a department. As departments open, close or are relocated, this information can change at any time, so the number of resulting tables is not predetermined.

The denormalization could be done with a simple join. The partitioning of the table, in order to be independent of the actual values for city found in the database, can be expressed with the HRA partition operator $\neq$.

![Figure 3: The human resource database represented using the PMA](image-url)
5.1 Simulating a Basic Data Modeling Operation

This simple example shows that even though information integration has to deal with operations that go beyond the capabilities of basic data modeling languages, they are still the mainstay of many integration tasks. Usually, a considerable part of an integration plan will likely be modeled in a single data model and will only require basic functionality, like that provided by SQL. The left of Figure 4 shows how the relational join operator (the generalized \( \theta \)-join) can be represented using the GTA. The operator is split into two production rules, one for the schema and one for the data part. On the LHS, the Join_Schema rule selects several elements out of the host graph: two tables \( t_1 \) and \( t_2 \) with multinode for all their respective columns \( c_1 \) and \( c_2 \), the data types of the columns \( ty_1 \) and \( ty_2 \), and the (shared) schema the two tables reside in. Note that a homomorphic group is declared for the tables (to allow self-joins) and for the column types (to allow several columns to have the same type). The column sets \( c_1 \) and \( c_2 \) and table \( t_1 \) are chosen for copying, while the other elements are used in-place on the RHS. A copy of \( t_1 \) labeled \( t_1 \) is created, which is attached to the same schema to which \( t_1 \) and \( t_2 \) already belong. \( t_1 \)'s name is set to the concatenation of the names of \( t_1 \) and \( t_2 \). All columns of \( t_1 \) and \( t_2 \) are copied and attached to the new table. The copied columns are connected to the same type nodes as their respective template.

The LHS of the rule handling the data part of our join (Join_Data) selects pairs of instances of the tables \( t_1 \) and \( t_2 \) (i.e., the tuples), together with instances of their respective columns (i.e., the tuples’ attributes). Note that the only binding between the two rules is actually the type test of the LHS and the type assignments of the RHS. Copies of the instances of columns are attached to a new instance for the new table \( t_\) (i.e., a tuple of \( t_\)). The application condition consists of a placeholder which is bound to a boolean expression that represents the join condition parameter, so that only those pairs of tuples \( (tu_1, tu_2) \) that fulfill the join condition result in the creation of a \( t_\) tuple for the result table.

![Image](image-url)

**Figure 4:** The \( \theta \)-join \( (t_1 \bowtie_{joinCond} t_2) \) and partition \( (\delta_{t_\bowtie}) \) operators
Using morphisms between the LHS of operator rules and the host graph, and between the rules and the graph representing the join condition expression, we can now parametrize the GTA join operator to use it on the `emp` and `dept` tables (`emp`\(\times_{\text{emp}, \text{dept} = \text{dept}, \text{dept} = \text{dept}}\) of the HR database of Figure 3. Variable \(\text{t1}\) is bound to the `emp` table, \(\text{t2}\) to `dept`, and \(\text{s}\) to the HR schema node. The join condition, given as an expression graph, is bound to the application condition of the `Join Data` rule. This binding describes the first operator of our abstract integration plan shown in Figure 5.

![Graph Transformation Diagram](image.png)

Figure 5: The abstract integration plan represented as chained graph transformations

### 5.2 Translating Data to Metadata

FIRA [WR05] defines operations that can translate data to metadata and vice versa. The partition operator \(\text{pCol}\) partitions an input relation \(\text{R}\) into newly created output relations based on the values contained in a column \(\text{pCol}\) given as parameter. Partitioning is generally used on categorical attributes, which have few distinct values when compared to the cardinality of the input relation. In our example HR database, we use the partition operator \(\text{pCol}_{\text{emp}, \text{dept}}\) to split the result of the previous join, the `emp`, `dept` table, into separate tables depending on department’s location. Its graph transformation representation is shown on the right in Figure 4. The operator is again split into a schema and a data rule. For each value of the partition column \(\text{pCol}\), we create a new table that contains identical columns as the input relation, except for the partitioning column. Its respective value is used to define the name of the resulting table. As this rule would create a new table for each occurrence of the value (not for each distinct value) we define a NAC stating that a table based on this value must not already exist in the schema. The data part of the operator takes instances of the input relation (tuples and their attributes) and creates an instance of the specific output relation that corresponds to the input tuple’s value in the partition column.

We bind the GTA partition operator to the table node that represents the result table `emp`, `dept` of the previous join operation. To parametrize the operator, the \(\text{pCol}\) pattern

---

223
node is bound to the city column node of the same table. This binding gives us the second operator of our abstract integration plan in Figure 5.

6 Deployment of Mappings

An abstract representation for integration plans such as the one described above is a valuable tool: It allows us not only to describe integration plans in a way that is independent of a specific platform. Even more important, we can now express integration plans that require functionality going beyond that of a single platform (and thus also beyond the expressiveness of a single operator language), i.e., plans that would otherwise require a combination of languages and integration platforms. But with the graph transformation formalism, we not only have described an abstract operator algebra, but also a directly executable specification for a generic graph transformation system. While such a system is certainly useful for the creation and testing of mappings or the development of new operators, such a generic system could never be as efficient or be optimized to the same degree as the mature, but specialized existing information systems.

Therefore, once the tasks that benefit from a generic representation have been accomplished, e.g., the integration plan has been created, their results represented in the generic representation should be deployed to a suitable platform or a federation of platforms, if no single platform offers the full expressive power that would be needed. This is analogous to the idea of the Model Driven Architecture (MDA) [OMG03b]: GTA represents a domain-specific language for the domain of data and metadata management. A generic transformation plan described with GTA represents a platform-independent transformation model (PIM), which is then transformed into a platform-specific transformation model (PSM). Since we already have a graph representation for graph transformations and their combination, it is natural to integrate the deployment task into the overall PMA framework: For each concrete integration platform, a metamodel that represents its respective native transformation language is defined. Instances of this model represent concrete transformation sequences. For example, a metamodel for relational algebra would have a node type for each of the operators, with attributes representing parameters and edges indicating how the results of one operator are input for the next. To deploy an abstract plan to the specific representation, we can again use graph transformations to define a deployment specification, i.e., the mapping from abstract to concrete operators. A deployment rule’s LHS refers to operators and morphisms of the abstract GTA representation, and on its RHS creates one or more appropriately configured platform-specific operators.

7 Related Work

As motivated in section 2.2, existing metamodelling approaches like the CWM [OMG03a], the Graph Exchange Language [HSSW06] and the Resource Description Framework RDF [MM04], have deficits when it comes to the efficient representation and handling of data,
which are corrected by the Paladin Metamodeling Architecture (PMA). Existing data and metadata management formalisms and systems that go beyond the expressiveness of established query languages, e.g., the FIRA/FISQL languages [WR05], or the Rondo model management framework [MRB03a, MRB03b] all focus only on supporting specific aspects of the general data/metadata management problem. To the best of our knowledge, our approach is the first to subsume the expressiveness of all these approaches.

A fundamental concept of the Model Driven Architecture [OMG03b] is the transformation of platform-independent models (PIM) of application logic to platform-specific models (PSM). This is comparable to our concept for deploying generic mappings to concrete runtime platforms. A number of approaches considers the use of graph transformations to perform these model-to-model transformations (e.g., [GLR+02]). However, none of these approaches includes the modeling or transformation of data.

8 Conclusion and Outlook

We motivated how the area of data/metadata management in general, and information integration in particular, could benefit from a generic formalism for describing all possible operations on data and metadata. We introduced a generic data/metadata representation using an attributed, typed graph model. We defined a graph transformation formalism that allows us to define arbitrary operations on this representation in a natural way. We illustrated how this formalism can be used to represent data of existing data models and to mimic existing operations of the data model. We further motivated how the graph transformation formalism can also be used to specify transformations of transformation plans, e.g., to map the platform-independent transformation plans to platform-specific plans during deployment.

The power of our formalism can be used to perform a number of other data/metadata management tasks that we have yet to explore in more detail. For example, graph transformations can describe algebraic rewriting rules to optimize both abstract and the concrete transformation plans, structural schema matching approaches, or metadata management operations. While our current graph transformation formalism and its concrete syntax have shown to have adequate expressiveness, they need further streamlining to ease the specification of operators. At the same time, the amount of commonalities between different concrete metamodels that are captured by the core metamodel has to be increased, to enable the specification of more generic, metamodel-independent operators.

While our formalism is primarily intended as a theoretical framework, a reference platform for further refinement, like the refactoring of metamodels and developing operators, is needed. Existing graph transformation systems have proven to be inadequate for mapping some of the advanced constructs of our language and its copy semantics to their respective language concepts. Therefore, a prototype graph transformation system custom-tailored to our requirements is under development in our group.

Some open questions regarding the deployment process have still to be answered: The flexibility of GTA leaves many equivalent options to specify the same operator. This makes
their identification during deployment difficult. Besides the obvious solution of establishing standard representations for operators, one possibility we are looking into is to define a canonical form for graph transformations.

References


Abstract: In many application areas like e-science and data-warehousing detailed information about the origin of data is required. This kind of information is often referred to as data provenance or data lineage. The provenance of a data item includes information about the processes and source data items that lead to its creation and current representation. The diversity of data representation models and application domains has lead to a number of more or less formal definitions of provenance. Most of them are limited to a special application domain, data representation model or data processing facility. Not surprisingly, the associated implementations are also restricted to some application domain and depend on a special data model. In this paper we give a survey of data provenance models and prototypes, present a general categorization scheme for provenance models and use this categorization scheme to study the properties of the existing approaches. This categorization enables us to distinguish between different kinds of provenance information and could lead to a better understanding of provenance in general. Besides the categorization of provenance types, it is important to include the storage, transformation and query requirements for the different kinds of provenance information and application domains in our considerations. The analysis of existing approaches will assist us in revealing open research problems in the area of data provenance.

1 Introduction

With the increasing amount of available storage space and the acceleration of information flow induced by the internet, a growing interest in information about the creation process and sources of data has developed.

Scientists in the fields of biology, chemistry or physics use data from so-called curated databases. Most of the data stored in a curated database is a result of manual transformations and derivations. Researchers that use data from a curated database are interested in information about the sources and transformations that were applied to this data. This information can be used to assess the data quality, examine or reconsider derivation processes or re-run a specific experiment.

Data warehouses are used to integrate data from different sources and with different data representations, and to analyze the integrated data. Such analyses could benefit from in-
formation about the original data sources and transformations used to create the data in the data warehouse.

In workflow management systems and more generally in systems based on SOA (service oriented architecture), several, probably distributed services are used to accomplish complex computational tasks. For the user it would be interesting to understand how a result from such a computation was created.

When combined with timestamped secure certificates, data provenance could be used to investigate copyright issues. For example, a scientist could prove that he was the first one to perform a certain experiment or that he is the creator of a specific piece of information.

In business applications, document management systems are used to manage the company documents and permit multiple users to work on the same document. A supervisor can use provenance information to gain a better understanding of the workflow in his company or find the origin of erroneous documents or document parts. The same ideas apply to distributed software development tools.

Other application domains that could benefit from provenance information include interactive statistical environments, visualization and KDD (knowledge discovery in databases).

While this broad range of application areas would benefit from provenance information, the type of provenance data, manipulation and querying facilities needed differ from application to application. Therefore, we try to find out the differences and similarities between the various application and data model provenance needs and present a general scheme for the categorization of provenance. By defining this scheme and applying it to existing work we hope to reveal open questions in the area of data provenance.

The remainder of this paper is organized as follows. In section 2 we discuss existing research approaches for managing provenance and introduce a consistent terminology. In section 3 we present our categorization scheme based on a generic view of provenance. The application of this scheme to study the properties of existing approaches is presented by means of an example in section 4. Finally in section 5 we cover open research questions and conclude in section 6.

2 Overview of Existing Approaches

A broad diversity of terms is used in the literature for provenance related concepts. To prevent confusion we introduce a consistent terminology for the most important concepts in data provenance and relate them to the terminology used by other researchers. A number of synonyms like lineage or pedigree are used for data provenance. We decided to use the term provenance, because it is short and intuitive. The terms provenance model and provenance management system are used to distinguish between a conceptual model for provenance and a system for the management of provenance information. Provenance was studied for different data models and levels of detail. Each data model has its own terminology for data and hierarchical structures. We use the term data item for a structural unit of data, which is the target of provenance management and the notion level of detail
for the granularity of a data item. For example a data item could be an XML-document, a
database tuple, a database relation, a database schema construct or a a file in a file system.
Tuple and relation are two possible levels of detail for a data item in a relational database.
To abstract from different systems for storing data like relational databases, object oriented
databases and file systems, we refer to these storage systems as data repositories. The term
hierarchy position is used for the position of a data item in the structural hierarchy of a
data model. For example, for database tuples the hierarchy position could be represented
by the database and relation the tuple is stored in.

There are two basic views of provenance. The first one describes the provenance of a data
item as the processes that lead to its creation and the other one focuses on the source data
from which the data item is derived from. In contrast to [Tan04], where these concepts
are called provenance of data and provenance of a data product, we use the terms source
provenance and transformation provenance because these notions seem to be more intuitive. The term transformation refers to the creation process itself and the terms source
and result refer to the input and output of a transformation.

Most of the existing research can be classified by their approach to provenance recording.
One research direction focuses on computing provenance information when data is cre-
ated, while the other computes provenance data when it is requested. Tan [Tan04] refers
to these approaches as lazy and eager. Most of the eager approaches are based on anno-
tations about source data items and transformations, and most of the lazy approaches rely
on inversion or input tracing of transformations.

Buneman et al. distinguish in [BKT01] between Why- and Where-provenance. While
Why-Provenance captures all source data items that contributed to the creation of a result
data item, Where-provenance captures the concrete origin of a result. We use the terms
contributing source and original source instead of Why- and Where-Provenance. In sec-
tion 3 these concepts are discussed more in detail.

Several surveys on provenance have been published [BF05, SPG05a, SPG05b], but most
of them cover only a specific type of provenance or separate different kinds of provenance
according to their application domain. In [SPG05b] a taxonomy of provenance systems
is introduced. While forming a valuable overview of existing approaches, this taxonomy
fails in seeing provenance from a more abstract, conceptual point of view.

A number of authors address provenance in the context of services and workflow manage-
ment. The PReServ (Provenance Recording for Services) [GMM05, GJM+06a, GJM+06b]
approach uses a central provenance management service. In [GMM05] this web service
receives messages, called p-assertions, from the web services used for data transforma-
tion. The provenance data is stored by the provenance management service in a so-called
provenance store. PReServ uses a common interface to enable different storage systems as
a provenance store. The system relies on the user or service developer to modify existing
services to support p-assertions. Simmhan et al. [SPGM06] expect services to post prove-
nance information on a message board. A provenance service collects these messages and
stores them in a provenance store. The myGrid system [SRG03] provides middleware for
biological experiments represented as workflows. myGrid records all service invocations
including parameters and data items. This log information can be used to derive the prove-
Chimera [FVWZ02, Fos03] offers a Virtual Data Catalog for provenance information. A user registers transformations, data objects and derivations (an execution of a transformation) in the Chimera Virtual Data Catalog (VDC). The VDC is implemented as a relational database. VDL (Virtual Data Language) provides query and data definition facilities for the Chimera system. While the first prototype is limited to file system data objects and executable program transformations, the system is to be extended to support relational or object-oriented databases and SQL-like transformations. Chimera is a part of the GryPhyN project [AF00], a research project developing techniques for processing and managing large distributed data sets in data grids.

Cui et al. [CWW00, Cui02] focus on lazy computation of provenance for data warehouses. They study views in data warehouses and develop algorithms to generate queries for provenance tracing. These queries trace all source data tuples that lead to the creation of a tuple in a view. In [CW01] their approach is extended to general transformations that could not be expressed in SQL. While rather complete in the domain of transformations and views in data warehouses, their approach is limited to this domain and does not include other data models or user generated provenance. In the work of Fan and Poulouvasilis [FP05], provenance data is recorded at schema level in the context of schema transformations in a data warehouse.

The topic of provenance for relational databases was first discussed in the context of visualization [WS97]. Here data transformations are represented as functions from one attribute domain to another. Provenance is traced by using inversions of these functions. Another approach for provenance management in a visualization environment is presented in [Gro04]. Groth et al. record user actions in an interactive visualization environment and present the whole user interaction as a DAG (directed acyclic graph). The user can navigate in this graph and jump back to previous states of the system.

In [Wid05], Widom presents considerations for the Trio-system. Trio is a database system for handling uncertain data and provenance. The formal foundation and implementation details are presented in [ABS+06]. Uncertain data has been intensely studied by the database community for more than two decades, but the combination of uncertainty and provenance introduce new challenges. Trio shows that provenance information can be used to solve some of the problems that arise from introducing uncertain data in a database system. While interesting because of the combination of provenance and uncertainty, the provenance computation of the Trio system based on earlier work for data warehouse views [CWW00, Cui02]. The provenance of the tuples in the database is stored in an auxiliary table. Trio records one level of provenance, meaning only the tuples from which a tuple was directly derived are stored in the provenance relation. A database-wide unique tuple ID is used to identify tuples. Halevy states in [HFM06] that the management of provenance in combination with uncertainty is needed for dealing with data spaces, a concept for next generation data management systems introduced in this publication.

In the GIS research area, the importance of provenance for evaluating the quality of data items has been recognized early on. Most publications from this area focused on the development of metadata standards [HE97] which include provenance information. A well-
designed metadata standard could provide a basis for provenance management system, but the proposed standards are limited to the GIS-domain and cannot be easily generalized. More important the metadata defined by these standards is meant to be provided by a user and may not be appropriate for automatic processing.

In [BCC06], data from various data sources is represented in a tree-structure. In this framework updates, insertions and deletes are copy, paste and insertion operations on these data trees. The authors present a query language that operates on data trees. Buneman et al. used this tree-representation of data for archiving [BKTT02]. They require unique keys for every data object. These keys can be used to associate different versions of a data item in different versions of a database (or data repository).

Provenance is related to data annotation. Annotation systems like DB-Notes [CTV05] and MONDRIAN [GKM05] enable a user to annotate a data item with an arbitrary number of notes. These notes are normally propagated when annotated data is transformed. Using annotations to represent information about a data item is a common approach in life sciences. The possibilities of using annotations to maintain provenance information were first discussed in [BKTT02, BS02]. This approach was also taken in [ZGG+03, MPL+06, CTV05]. The DB-Notes system [CTV05] enable a user to store annotations at the attribute level for data in a relational database. DB-Notes is based on a relational database and queries are represented in pSQL, an SQL-like language including constructs for managing annotations.

Data provenance is also related to temporal data management and versioning. Like in temporal data management, in provenance previous versions of a data item are queried and accessed. So provenance management systems may benefit from existing storage methods and query optimizations for temporal databases. Intelligent archiving techniques [BKTT02] need methods capable of identifying an object in different versions of a database or document. The identification methods used in this context may be also applicable to provenance management.

3 A categorization scheme for conceptual properties of data provenance

In this section we discuss data provenance from a conceptual point of view, extend the terminology introduced in section 1 and define a general categorization scheme for provenance management systems. We define several functionalities a provenance management system can provide and order these functionalities in a hierarchy of categories. The three main categories of our categorization scheme are provenance model, query and manipulation functionality and storage model and recording strategy. We present an overview figure for each main category (figures 1, 2 and 3). Categories and functionalities are represented by boxes and ellipses in these figures.
3.1 Provenance model

The category **provenance model** describes the expressive power of the conceptual model used by a provenance management system to define the provenance of a data item. We define a number of functionalities and categorize a provenance system by means of the functionalities it supports. As stated before, the provenance of a data item can be divided into the two parts *transformation provenance* and *source provenance*. Source provenance is information about the data that was involved in the creation of a data item. There are three conceptual definitions for source provenance. These are *original source*, *contributing source* and *input source*. The input source of a data item includes all data items that were used in the creation of this data item. The *positive contributing source* of a data item includes data items that are essential for the creation of this data item. In formal terms the positive contributing source of a data item X is the union of all minimal subsets of the input source that, when used in the process that lead to the creation of X, would also lead to the creation of X. The original source of a data item consists of all data items which include data that is copied to the resulting data item.

For example, assume we manage the provenance of data in a relational database with two relations $r_1$ and $r_2$ and handle data items at tuple level of detail. When executing the SQL-query

```
SELECT r_1.name FROM r_1, r_2 WHERE r_1.id = r_2.id
```

against the database including relations $r_1$ and $r_2$ the input source of a resulting tuple $t$ includes all tuples in $r_1$ and $r_2$. The positive contributing source of $t$ consists of all tuples $t'$ from relation $r_1$ and $t''$ from relation $r_2$ with $t.name = t'.name$ and $t'.id = t''.id$. At last the original source
of \( t \) includes all tuples \( t' \) from relation \( r_1 \) with \( t.name = t'.name \).

The concepts original source and positive contributing source were first introduced in [BKT01] under names Why- and Where-provenance. Note that the following subset relationship holds:

\[
\text{input source} \supseteq \text{positive contributing source} \supseteq \text{original source}
\]

Some applications would benefit from information about data items that are not existent in the source, but would inhibit the creation of a resulting data item, if they were included in the source. This concept has been first introduced by Widom et. al. in [CWW00]. We use the term negative contributing source for this concept. In contrast to the concept positive contributing source, this definition is not straightforward. It seems feasible to include either all data items that would alone prohibit the creation of the result or include combinations of data items that would prohibit the creation of the result. In most data repositories the amount of data stored in the repository is only a very small fraction of the all possible data that could be stored in the repository. So in the general case, it is not possible to actually store the negative contributing source of a data item.

Additional to the considerations which kind of sources should be included in the source provenance, a provenance management system can record various information about each source data item. A source could be represented as the original data, metadata attached to the source, the source hierarchy structure or a combination of this representations.

A provenance system can either record source data items at one level of detail or be able to handle multiple levels of detail. For example, the source of a tuple data item in relational view could include all tuples from a relation \( r \). If the provenance model handles multiple levels of detail the source could be represented as relation \( r \) instead of representing it by all tuples from \( r \). Managing provenance information at different levels of detail is the more flexible approach and can result in smaller storage overhead of provenance information, but requires a more complex provenance model.

Transformation provenance is information about the transformations that were involved in the creation of a data item. To make a clear separation between a concrete execution of a process and the process itself we use the term transformation class for the first and transformation for the later. Foster et al. use the terms transformation and invocation for these concepts [Fos03]. In our concept a transformation is not limited to be an automatic process, but may be a manual process or a semi-automatic process with user interaction. The transformation provenance of a data item could include metadata like author of the transformation, the user who executed the transformation and the total execution time.

Examples for transformations are SQL statements used to create views in a database, the workflow descriptions of a workflow management system and executable files with command line parameters.

An important part of the provenance model is the world model, which could be either closed or open. In a closed world model the provenance management system controls transformations and data items. Contrary in an open world model the provenance management system has no or only limited control over the executed transformations and data
items. Data items and transformations can be executed, manipulated, created or deleted without notification. From the view of the provenance management system, the world has an uncertain behavior which results in complex provenance recording or make it impossible to record exact provenance information. The closed world and open world model are extremes and there are many possible world models that are neither closed world nor open world models.

Besides the functionalities a provenance management system provides to handle source and transformation provenance, it should be able to recognize if two data items from two different data repositories represent the same real world object. For example the same data item could be stored in many databases or even in a database and as an XML-document. As real world objects tend to change over time, it is important to have mechanisms for checking if two data items are different versions of one real world object. Identification is especially important, when updates to the data repositories are not controlled by the provenance management system. In this case the information about source data items recorded by the system might be incorrect, because these data items were changed or deleted by an update.

The data item equality needed for provenance management systems with open world models is a semantic equality, which has been studied in depth by the data integration community [Ken91, New88, PAGM96, BN05]. Semantic equality in general is not solvable for open world models, but there are several heuristical approaches to this problem (e.g. [Zia99]). The problem of identifying different versions of the same object also applies to archiving and is discussed in [BKT02].

There are several possibilities to identify duplicates. A straightforward approach would be to check if the data item and the duplicate represent exactly the same information. We refer to this approach as value based duplicate identification. If data items have a key property, then another approach is to identify duplicates by their key property. For example, if data items are tuples in a relational database, two tuples could be defined to be duplicates if they have the same attribute values or if they have the same key attribute values. Using the primary key constraints of a relational database for identification could be problematic when no further restrictions are introduced, because the primary key uniqueness is restricted to one relation and primary keys can be changed by updates.

Many data models have an explicit or implicit hierarchical structure. This hierarchy in combination with a key property or value equivalence could be used to identify a data item. For example, if the provenance of tags in XML-documents is recorded, duplicates could be defined by the name of the tag and the position of the tag in the hierarchy of the document.

3.2 Query and manipulation functionalities

To be useful for a real world application, a provenance management system should provide facilities to manipulate and query provenance information and data items. We do not discuss manipulation and querying of data items without integration of provenance infor-
mation, because these query and manipulation facilities have been extensively studied and are well understood.

If a provenance management system handles transformations at various levels of detail, it should provenance mechanisms for merging multiple transformations into one transformation and split a complex transformation into a sequence or graph of simpler transformations. This functionality is similar to the composition of processes implemented by workflow management systems [Moh96]. Provenance data can be used to recreate result data items [Fos03], which cannot be accessed or are expensive to access, by executing the transformations that were used to create the result data item. If a provenance management system is able to compute the inverse of a transformation, then the inversion can be used to recreate source data items from result data items. For example, Widom et al. [CWW00] compute queries for tracing the contributing source tuples for tuples in a materialized view based on the view definition statement.

Split and merge operations could also be applied to the data item dimension. Split divides a higher-level data item into its lower-level parts and merge combines lower-level data items into a higher-level data item. While the split operation is quite clear, the merge operation raises some questions. For example, what is the result of the merge operation on a subset of the lower-level data items that form a higher-level data item, and how can this result be distinguished from the result of a merge operation on the whole set. A provenance management system that records provenance information for different data models should provide facilities for converting the representation of a data item from one data model to another.

So far we here omit the aspect of provenance storage strategy. Provenance information may be attached to the physical representation of a data item or stored in a separated data repository. We discuss this topic in detail in the next subsection. At this point we are only interested in the fact that a provenance management system may support more than one storage strategy and might offer mechanisms for changing the storage strategy for data
Depending on the properties of the provenance model and world model it may be difficult or even impossible to implement the postulated manipulation operations.

### 3.3 Storage and recording

![Diagram](image)

The category storage and recording includes the approaches a provenance management system uses to store provenance information, to record provenance information and to propagate provenance information recorded for source data items.

Storage strategy describes the relationship between the provenance data and the data which is the target of provenance recording. There are three principal storage strategies: the no-coupling, the tight-coupling and the loose-coupling recording strategy. The no-coupling strategy stores provenance information in one or many provenance repositories. These repositories are dedicated to storing only provenance data. The second option, tight-coupling recording, stores provenance directly associated with the data for which provenance is recorded. The loose-coupling strategy uses a mixed storage scheme where provenance and data are stored in one storage system but logically separated.

Most approaches based on annotation use a tight-coupling or loose-coupling strategy by attaching provenance annotations to data items or storing annotations in the same data repository, but separated from the data items. Service based approaches in general record provenance for several data repositories in a distributed environment. These approaches normally deal with a very heterogeneous environment with limited control over the execution of processes and manipulation of data item, which make it difficult to record provenance information. Some control over this environment and especially over the provenance information can be gained by using a closed world model and a no-coupling storage strategy.

There are multiple storage models for storing provenance. In principle every data model could be used to store provenance information, but not every combination of storage model and storage strategy is reasonable.
If provenance is recorded for a transformation which uses source data items with attached provenance information, how is this information propagated to the result data items? The three possible answers to this question are no-propagation, restricted propagation and complete propagation. With no-propagation the provenance of source data items of a transformation is ignored when creating provenance data for result data items of the transformation. Contrary under complete propagation the result data items of a transformation inherit all provenance data from source data items according to the kind of source used in the provenance model. Under restricted propagation a result data item inherits a part of provenance from the source data items, e.g. data provenance that was created during the last \( n \) transformations.

The provenance recording strategy specifies when provenance data is recorded. We consider user controlled recording, eager recording, no recording and system controlled recording. Obviously, with user controlled recording the user decides when and for which data item he likes to record provenance information. Eager recording records provenance simultaneously with every transformation. The no recording approach generates provenance at query time. Under system propagation recording data creation is controlled by the provenance management system. This system could use strategies like record the provenance data once a day or record the provenance after every \( n \) transformations. Eager recording and no recording are related to the eager and lazy provenance tracking approaches introduced in [Tan04].

4 Categorization of existing approaches according to our categorization scheme

In this section we categorize an existing model according to the scheme presented in the last section. A categorization of all existing approaches is beyond the scope of this paper, so we limit the discussion to one example. A long version of this paper including more categorizations is available on the author’s homepage.

4.1 Provenance of views in a data warehouse

<table>
<thead>
<tr>
<th>provenance model</th>
<th>world model</th>
<th>closed world model in [CW00]: extended by openness in [CW01] for transformations</th>
</tr>
</thead>
<tbody>
<tr>
<td>identification</td>
<td>no data item versions supported and no duplicate recognition</td>
<td></td>
</tr>
<tr>
<td>representation of transformations</td>
<td>concrete execution (relational algebra expression) with handling of different levels of detail</td>
<td></td>
</tr>
<tr>
<td>supported transformations</td>
<td>SQL view definitions. In [CW01] extended with general data warehouse transformations (e.g. ETL)</td>
<td></td>
</tr>
<tr>
<td>representation of source data</td>
<td>original source data: database tuples</td>
<td></td>
</tr>
<tr>
<td>source definition</td>
<td>contributing source for views. Only input source for some of the general transformations introduced in [CW01]</td>
<td></td>
</tr>
<tr>
<td>provenance based recreation</td>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

query and manipulation facilities
In [CWW00], Widom et al. study the provenance of materialized views in a data warehouse. They present algorithms to create inverses of a view definition for tracing the positive contributing source of tuples in the view, based on the view definition. Transformations are represented by the relational algebra view definition and are handled at different levels of detail (by defining provenance for the basic relational operators and for the concatenation of these operators thus allowing views based on views). The provenance data is computed lazily and is not stored for further use. In the original paper the closed world model of a data warehouse is used and extended in [CW01] to allow certain openness and new transformations not representable in relational algebra by including general data warehouse transformations.

5 Open Questions and Future Research Directions

Our categorization scheme includes functionalities not jet included in provenance management systems. For example developing a provenance management system for open world models is a challenging problem. Furthermore many of the manipulation facilities present in the scheme are not included in the existing approaches. A formal model designed with the insight gained in this article could be the basis of a provenance management system that handles not only various storage models, but also different types of source and transformation provenance. Source and transformation provenance are not completely independent and it would be interesting to investigate under which circumstances it is possible to convert one into the other and study how much redundancy is introduced by storing source and transformation provenance. It seems also reasonable to investigate which of the functionalities included in our categorization scheme exclude or imply each other.

Some of the problems faced when dealing with provenance are related to data integration problems. For example the concept of semantic identity needed to recognize duplicates or versions of data items in an open world model was studied by various data integration publications. A provenance management system handling different kind of data items stored in distributed repositories needs to integrate this data to gain a unified view on the data. Data integration systems might benefit by including provenance management. For example provenance data could be used to identify duplicate pieces of data or could help a user to assess the quality of integrated data.

It would be interesting to apply concepts developed in the area of temporal database and versioning in to the provenance management of updateable data repositories.

6 Conclusions

We have presented a categorization scheme for different types of provenance and categorized existing approaches according to this scheme. The categorization scheme helps us to gain a systematic
overview of the capabilities and limitations of these approaches. Most categories used in our scheme are based on concepts developed by other researchers, but we investigated new combinations of these concepts and extended some of these concepts with new aspects.

In future work we will investigate which of the functionalities included in our categorization scheme exclude or imply each other. Such an analysis would help us to gain a better understanding of provenance in general. This investigation could be extended to cover investigation of implementation problems and complexity analysis for different functionality combinations.

We will also define a formal language for the management of provenance data. This language should include generation, querying and manipulation of provenance data. Unlike existing approaches, this language should cover not only different data models, but also manage different types of provenance information. It will include language constructs for converting between different data models and kinds of provenance data. We plan to explore the computational complexity and the implementation problems of this language with the goal of creating a prototype implementation. Because of the complexity of the problem, the prototype will be limited to a specific kind of provenance and restricted manipulation options in the beginning.

References


Abstract: Over the recent years, very little effort has been made to give XPath a proper algebraic treatment. The only laudable exception is the Natix Algebra (NAL) which defines the translation of XPath queries into algebraic expressions in a concise way, thereby enabling algebraic optimizations. However, NAL does not capture various promising core XML query evaluation algorithms like, for example, the Holistic Twig Join. By integrating a logical structural join operator, we enable NAL to be compiled into a physical algebra, containing exactly those missing physical operators. We will provide several important query unnesting rules and demonstrate the effectiveness of our approach by an implementation in the XML Transaction Coordinator (XTC)—our prototype of a native XML database system.

1 Introduction

There is one core task, common to almost all XML query languages: the matching of path patterns against XML documents. The performance of an XML query language processor intrinsically depends on its path evaluation engine, because path matching is a frequent and expensive operation. Path matching occurs frequently, because even multiple paths are often defined in a single query. And it is expensive, because path evaluation requires physical access to the document, in contrast to almost all other constructs of an XML query language, which are evaluated on the output generated by path matchings. Despite of the many algebra proposals regarding the standard XML query language XQuery [12, 17, 20], its path-related sublanguage XPath has unfortunately not gained as much attention. However, because of the above reasons, we believe that XPath should be furnished with an algebraic basement, too: It is the core XML data access mechanism in XQuery (and also XSLT) and it is itself a complex language to evaluate, leaving a lot of space for algebraic optimizations. In this paper, we will extend the Natix Algebra (NAL) [3], which is—to our knowledge—the only algebra, specifically dealing with the compilation of XPath.

So, what is missing in NAL? We observed that somewhat in parallel to the progress being made in the XML algebra community, a plethora of core algorithms for XML query eval-

1As you may convince yourself throughout this article.
uation as well as indexing techniques have been published, that qualify as physical XML query operators. Among them, the most prominent representatives are the Structural Join (STJ) [1, 7, 14, 15], the Holistic Twig Join (HTJ) [5, 10], and the various path indexes like, for example, the D(k) index [6]. While being introduced in the context of tree-based algebras [12, 13], very little attempt has been made to integrate these concepts into a tuple-based XML algebra, such as NAL [17]. You may think, why bother, the combination of a tree-based algebra with the holistic twig join works perfectly, so where is the need for a further XML algebra? We believe that the data model of tuple algebras is more general than the one of tree algebras and, therefore, certain XML query language constructs can be handled more suitably. For example, we do not know how a non-tree intermediate result, like pairs of siblings, is represented without introducing an artificial parent node (which has to be handled by subsequent operators). Furthermore, all major RDBMS vendors are currently integrating XML query capabilities into their (tuple-based) relational query engines. For them, the integration of an equally tuple-based XPath/XQuery algebra would be a natural thing to do. That is why we favor tuple algebras and think the integration of the above mentioned physical operators is of great importance.

In this article, we will elaborate on the algebraic treatment of XPath. We will introduce a logical structural join operator into NAL and provide essential rewriting rules to convert an algebraic expression into a format facilitating the mapping onto the existing physical XML operators STJ and HTJ. The extended algebra will be named NAL STJ.

1.1 XML Algebras in the Literature

Although there is—to our knowledge—only one proposal explicitly dealing with the algebraic compilation and optimization of XPath queries [3], we give an overview over existing algebra approaches for XML queries in general and point out their XPath capabilities.

The TAX and TLC algebras [12, 13] evolve from an analogy between relations and trees. In the relational algebra, each operator consumes and produces sets of tuples (relations), whereas sequences of XML data trees are the basic unit of processing in TAX/TLC, i.e., TAX/TLC is a tree-based algebra. A core concept to all operators are pattern trees. They can be used, for example, to define a query tree structure for a selection operator that matches the pattern tree against a document, thereby producing a sequence of so-called witness trees. Each witness tree in the result sequence corresponds to a match. The above mentioned physical algorithms, STJ and HTJ, are core algorithms in the TAX/TLC physical algebra, because they do the job of pattern tree matching. TAX/TLC provides a "natural" way to process XML trees, because it is based on XML trees as intermediate results. However, its expressive power is definitely too limited for the evaluation of XPath queries: only the descendant and child axis are supported for the definition of a pattern tree.

The Natix Algebra (NAL) [17] takes a different approach, because it abstracts from trees as intermediate result structures. NAL operates on sequences of (homogeneous) tuples.

---

2By “physical” we mean that these operators could be part of a physical XML algebra.

3See also [2] for academic research activities in this area.
each tuple consisting of a set of attribute-to-value mappings. Similar to the notion of the evaluation context defined in the W3C Formal Semantics [8], these mappings keep track of the dynamic variable bindings during query processing. Reference [3] describes the translation of an arbitrary XPath expression into NAL. Because our article heavily relies on NAL, we will sufficiently introduce the algebra and its capabilities in the following.

The algebra presented in [20]—called RSF4 algebra in the following—employs a hybrid approach. Its expressions contain both operator types: Tree-based operators are introduced for intermediate XML tree handling and tuple-based operators to control the flow of tuple streams generated by XQuery’s for and let expressions. To ensure the compatibility between these two types, special conversion operators (MapToItem and MapFromItem5) have to be embedded into an algebra expression. This technique avoids tuple flattening which is often required in NAL. Because RSF expressions are generated from the XQuery core representation defined in [8], the whole extent of XPath is covered. However, XPath-specific optimizations, as introduced in this article, have not been published in RSF so far. However, their integration would be possible.

1.2 A Brief Example in NAL

In this section, we will give a brief example in the Natix algebra and point out its strengths and weaknesses. Let us consider the expression \(/a_1::t_1/a_2::t_2\{\text{position()} = \text{last()})\)/a_3::t_3 [a_4 :: t_4] depicted in Figure 1. The evaluation starts with the singleton scan operator (□) which creates a singleton sequence containing an empty tuple. It triggers the map operator (χ) to bind the root node of the queried tree to the c0 attribute of a new tuple. This tuple, in turn, is consumed by the first D-join operator. The D-Join (⊗ or ⋊) in

---

4Named after the last names of the paper’s authors.
5MapToItem converts a sequence of tuples to a sequence of XML trees, while MapFromItem works in the opposite direction.
the textual representation) is similar to XQuery’s for construct: for each tuple \( t \) in the left input sequence, the dependent right expression is evaluated, binding \( t \)'s attributes to free variables in the right expression (here \( c_0 \)). Then, the intermediate result calculated for the dependent sub-expression, is joined with \( t \). In our example, the dependent expression is again a D-Join operator whose left sub-expression is an unnest map operator (\( \Upsilon \)). This operator is a shortcut for a map operator (\( \chi \)) followed by an unnest operator (\( \mu \)). In NAL, \( \Upsilon \) is mainly used for the calculation of path axes. Starting from \( c_0 \) the path expression \( a_1 :: t_1 \) is evaluated to a single sequence (using \( \chi \)) which is immediately unnested (by \( \mu \)). Together with the D-Join, this results in the above mentioned “flattening”.

A predicate is translated into a selection operator (\( \sigma \)), where the predicate’s sub-expression is compiled into \( \sigma \)'s subscript. NAL operators may be arbitrarily nested in this fashion. For each input tuple, the subscript is evaluated. For almost all predicates, certain measures have to be taken to ensure the evaluability of \( \sigma \)'s subscript: In case of a relative path expression, the current context variable \( cn \) has to be provided explicitly. This is accomplished by the two map operators \( \chi_{cn:c_3} \) and \( \chi_{c_0:cn} \), the first one binding \( c_3 \) to the context variable and the second one “transferring” \( cn \) into the variable \( c_0 \) of the local context. For positional predicates, the current context position and the context size have to be calculated. This is the task of the special operators \( \chi_{cp:counter(p_2)+1} \) and \( Tmp^{cs} \). The first operator simply counts the tuples in its input and attaches a new attribute \( cp \), containing the current position, to them. \( Tmp^{cs} \) materializes its input to calculate the total number of tuples in the context, before it attaches attribute \( cs \), containing this number, to each tuple. The aggregation operator \( \mathcal{A} \) evaluates aggregate functions, e.g., \( \min() \), \( \max() \), etc. More sophisticated predicates, for example existential comparisons, are possible, too. Finally, the resulting context node is produced by a map operator, and duplicate elimination (\( \Pi^D \)) is applied to comply with the XPath semantics.

NAL provides a concise algebraic basement for XPath (1.0 [23]) evaluation. The XPath-to-NAL translation process is described in [3] in great detail. Additionally, the authors provided some optimization techniques like stacked translation for outer paths, duplicate-elimination push down, and memoization\(^6\). In [4], certain algebraic equivalences were shown, which enable unnesting of queries with semi-correlated XPath predicates\(^7\).

### 1.3 Problem Statement

In spite of the progress being made in NAL, we believe that there is still room for optimization: Our first observation is that the evaluation of a NAL expression generates almost the same data flow as its equivalent normalized in W3C’s XQuery Core Language. As an example, consider the evaluation of the select operator \( \sigma \) in Figure 1: It is evaluated for each context node provided by the unnest map operator \( \Upsilon_{c_3\times c_2/a_1::c_3} \). This implies node-at-a-time calculation of the path step, embedded in the selection subscript. However, many publications [1, 5, 7] have pointed out that set-at-a-time processing of path steps provides better

---

\(^6\)These optimizations have not been executed on our example, which is presented in the canonical translation.

\(^7\)Queries with semi-correlated predicates have the form \( p = e_1[e_2 \theta e_3] \), where either \( e_2 \) or \( e_3 \) is a path expression depending on \( p \)'s outer—or global—context.
performance in most cases. Another example regarding the generated data flow arises from the order in which the path processing steps are evaluated. Like in XQuery Core, NAL evaluates path steps from left to right. However, as [22] has shown, a reordering of path step evaluations can substantially improve the query processing performance.

As a second point, we observe that the logical-to-physical operator mapping presented in [3] does not take important classes of physical operators into account, like the structural join and the holistic twig join\(^8\). Essentially, these operators provide the above mentioned capability to process path steps in a set-at-a-time manner. There is reasonable doubt that, in the face of complex queries, the algebraic representation can facilitate a mapping onto a physical algebra, containing exactly these operators. We draw this doubt from the fact that nested path expressions are “hidden” in subscripts of selection operators. Furthermore, logically related subexpressions, e.g., the compiled parts of the path steps like \(a_1 :: t_1\), are “scattered” across the operator tree (shown by the encircled areas in Figure 1). Under the assumption that the above query contains only steps referring to the child and descendant axis, a reasonable evaluation approach—at the physical level—would be the application of a single HTJ operator, followed by a subsequent selection. However, from the given representation, it is unclear how the mapping onto this HTJ operator can be accomplished.

### 1.4 Our Contribution

Our overall goal is to integrate the above mentioned important classes of physical evaluation operators like structural join, holistic twig join, and path index access into NAL’s physical algebra. However, as a first step we have to “prepare” NAL at the logical level in a way facilitating this integration. In this article we will

- introduce a *logical* structural join operator to the NAL algebra,
- provide rules to convert a NAL expression from its canonical representation into its NAL\(^{STJ}\) equivalent containing structural joins,
- develop rewriting rules for predicate unnesting, and
- finally show the impact of our approach on the query processing performance in the XML Transaction Coordinator (XTC)—our prototype of a native XML DBMS.

By introducing structural joins, we can abstract from the explicit and implicit node-at-a-time path processing steps (e.g., the D-Join, and the selection due to a path predicate) inherent to the Natix algebra. This abstraction allows us to choose the adequate physical operators for the implementation of a logical plan. From the large set of possibilities for structural join implementation (i.e., stack based [1], hash based [15], index based [7], locking aware [14], etc.), we will gain more flexibility in the logical-to-physical mapping, and therefore extend the search space for cost-based query optimization. Surely, there will

\(^8\)Although we recognize the hint towards that direction given in [18], we did not find any approach that properly introduces structural joins in NAL.
be situations node-at-a-time leads to a better performance than set-at-a-time. However, we think that this decision depends on physical properties and should, therefore, not be decided on a logical level.

Furthermore, our predicate unnesting rules will facilitate the mapping onto more powerful physical operators like the holistic twig join (which can also evaluate and, or, and not predicates) and path-index lookups, because they expose path processing steps hidden in selection subscripts. Additionally, unnesting enables structural join reordering to prise off the inflexible left-to-right path evaluation. We expect our operator plans to be scalable, though consisting of a large number of joins, because, in contrast to the join implementations in the relational algebra, structural joins are evaluable in linear time [1].

In the following, we will not consider questions arising during plan generation, i.e., during the logical-to-physical operator mapping. Specifically, we will neither show, how a holistic twig join can be employed to replace a set of structural join operators, nor how the order of structural joins can be selected [22]. Here, we only want to facilitate the treatment of these important questions by introducing the structural join operator.

The remainder of this article is organized as follows: Sect. 2 provides an overview over the Natix algebra, which we will extend in Sect. 3. The rule-based rewriting of NAL into its extended version is described in Sect. 4, before Sect. 5 introduces the core rule set for query unnesting. Sect. 6 provides several rules for structural join push down. We conclude this article with a quantitative analysis in Sect. 7.

2 NAL in a Nutshell

For your convenience, we repeat the basic definitions from [3]: NAL operates on sequences of homogeneous sets of attribute-value mappings (tuples) \( t \), each \( t \) having the same set of attributes (schema) denoted \( A(t) \). Attribute values may be sequences, thus NAL allows arbitrary nesting. The empty sequence is denoted as \( \varepsilon \) or \( \langle \rangle \). For tuple modification, NAL provides the primitives \( [] \) (tuple construction), \( \circ \) (tuple concatenation), and \( |_{A} \) (attribute projection). The notation \( t.a \) describes the access to tuple \( t \)'s attribute \( a \). \( A(e) \) and \( F(e) \) denote the schema and the set of free variables of an algebra expression \( e \). Applied to sequences, the functions \( e_{1} \oplus e_{2} \), \( \alpha(e) \), and \( \tau(e) \) return the concatenation (\( \oplus \)), the first tuple of the sequence (\( \alpha \)), and the remainder of the sequence (\( \tau \)). If \( e \) is a sequence of non-tuple values, \( e[a] = [a : e(e)] \oplus \tau(e)[a] \) returns a sequence of tuples \([a : e_{i}]\), where \( e_{i} \) is a tuple of \( e \). An overview over all relevant NAL operators can be found in the appendix.

To support the required ordering in XML, all unary operators—except \( \text{Sort} \)—keep the order of their input sequences intact. The binary operators cross product (\( \times \)) and D-Join (\( \bowtie \)) have nested-loop semantics. The projection operator (\( \Pi \)) has two variants for duplicate elimination (\( \Pi^{D} \)) and renaming (\( \Pi_{a'\sim a} \)).
3 Extending NAL to NAL$^{STJ}$

For our NAL extension NAL$^{STJ}$, we introduce some new operator definitions and modify a few existing ones. We want to keep NAL$^{STJ}$ backward compatible, i.e., an expression in NAL shall also be an expression in NAL$^{STJ}$. The new or modified operators are: the structural selection and the structural join, node sequence access, nesting, reverse, group reverse, group sort, and finally sequence-based merge (∪) and intersect (∩).

**Structural Selection.** The structural selection, i.e., the selection of a tuple based on some structural predicate, is embedded by extending the NAL selection operator from Table 3:

$$\sigma_p(s) := \left\{ \begin{array}{l} \alpha(s) \oplus \sigma_p(\tau(s)) : \Psi_p(\alpha(s)) = \text{true} \\ \sigma_p(\tau(s)) : \text{else} \end{array} \right.$$  

where the function $\Psi_p(t)$ evaluates predicate $p$ on tuple $t$. In case, $p = a_i \theta a_j$ is a structural predicate, $\Psi_p$ has the following semantics: Depending on $\theta$, the predicate evaluates the binary structural relation ↑ (is parent of), ↓ (is child of), ⇑ (is ancestor of), / (is ancestor or self of), ⇓ (is descendant of), ⇐ (is preceding sibling of) → (is following sibling of), ← (is preceding of), @ (is attribute of), and ⊲ (is self of). A structural predicate is evaluated to $\Psi_a(t) := t.a_i.a_j$. Note, if we want to express that “b is child of a” we write $b \downarrow a$ and not $a \downarrow b$. The order is important when we define the structural join.

For its evaluation, an XML node identification mechanism (labeling scheme) is beneficial that can decide the relationship in question without a physical node access. All native XML database systems nowadays embody such a mechanism.

In case of all other shapes of the predicate $p$, we refer to the original definition of the selection operator in [3].

**Structural Join.** With the help of the cartesian product (×) and the selection operator ($\sigma_p$), we define the join operator in the classic way:

$$s_1 \Join_p s_2 := \sigma_p(s_1 \times s_2)$$  

This operator becomes a structural join operator when the join predicate checks structural relationships over attributes of the participating tuples. However, some care has to be taken for certain axes that may produce duplicates. Additionally, the question of output order arises\(^9\). For example, when using $\Join_p$ to evaluate the ancestor axis, the output may not reflect the document order (as required by XPath). Therefore, when using the structural join operator, we will keep these aspects in mind. The structural semi-join ($\Join_p$), the structural anti-join ($\Join_p$), and the structural left-outer join ($\Join_p$) are defined accordingly.

Why do we claim this operator to be a *logical* operator? To answer this question, we first have to state that the distinction between logical and physical operators in XML algebras is

\(^9\)Note, that the $\times$ operator on sequences, as defined in [3], returns an ordered result.
not as clear as in the relational world. Because order matters in XML, logical operators are defined in a way, respecting the requirement of order (like $\times$). But then, there is often only one chance to implement a logical operator, because other alternatives do not deliver the correct output order. Therefore, there is often no distinction between a logical operator and its physical implementation. However, for the structural join operator defined above, there are a lot of very efficient physical algorithms present, e.g., stack based [1], hash based [15], index based [6, 7], locking aware [14], etc. We even think that the combination of a D-Join with an unnest map operator is a physical implementation of the structural join defined above. Despite the intrinsic nested loop characteristics, we think our new operator qualifies as a logical one.

Node-Sequence Access. For the access to sequences of nodes having, for example, the same element name, we define the auxiliary function $\phi_p$. For simplicity, its semantics is described in prose: $\phi_p(c)$ is a function depending on the current evaluation context\(^\text{10}\) $c$. It returns all nodes of a document in document order, complying with the predicate. For its evaluation, the function reads the current context node $cn$, defined in the evaluation context, and calculates $cn$’s document root node. Then it scans the document in document order, thereby evaluating predicate $p$ against each visited XML node. All qualifying nodes are returned in one sequence. In the following, $\phi_p$ will be used in combination with the $\Upsilon$ operator. For example, the expression $e = \Upsilon_{c; \text{author}}(\Box)$ returns a sequence with $A(e) = c$ and all $\text{author}$ elements in the current document as values.

Nest. In the following, we will not need the complex grouping capabilities of the general unary/binary grouping operator provided in NAL [17]. A simple nesting operator will do. Nesting is the complementary operator to unnesting. We assume the grouping operator in [17] to be defined on sets (or, more specifically, on vectors) of attributes $A$. Then, nesting is a shorthand for $\nu_{g: A}(e) = \Gamma_{g: A\{id\}}(e)$. If we want to nest by all attributes but the ones given in the vector $A$, we use $\nu_{g: \overline{A}\{id\}}(e)$.

Reverse, Group Reverse, and Group Sort. The reverse operator $R$ simply reverses the order of the tuples in the input sequence. If given an attribute name as subscript, $R_g$ assumes attribute $g$ to be sequence valued. Then, it reverses the order of $g$’s sequence. The group reverse operator $R_{G_{A}}$ first nests its input by the attribute list $A$, reverses the order in each nesting group, and finally unnests the sequence again:

$$R_{G_{A}}(e_1) = \mu_g \circ R_g \circ \nu_{g: A}(e_1)$$

The same can be defined for the sort operator. Similarly to $Sort_{cn}$, the operator $S_{g}$ sorts the sequence valued $g$ in ascending (document) order on the context node (cn). Then group-based sorting can be defined as:

$$S_{G_{A}}(e_1) = \mu_g \circ S_g \circ \nu_{g: A}(e_1)$$

\(^{10}\)Note, in the following, we omit context-parameter $c$ for simplicity
Sequence Merge and Intersect. The operators $\cup$ and $\cap$ are defined as the sequence-based, order preserving, and duplicate eliminating union and intersection on sequences of tuples having the same schema.

4 Introducing the Structural Join into a NAL Expression

In this section, we present a set of rewriting rules which substitute D-Join operators with structural joins. Each rule contains an operator pattern at the left-hand side. The corresponding right-hand side specifies how the operator tree has to be restructured. Note, a direct compilation from XPath to NAL$^{STJ}$ is also possible. However, in this article we chose a given NAL expression as the starting point, because we want to ensure the equality of the resulting NAL$^{STJ}$ expression. Due to space restrictions, we cannot provide any reasoning about the correctness of the following rules. The necessary proofs can be found in the extended version of this paper [16]. After each rule application, the resulting operator tree can still be evaluated, because NAL$^{STJ}$ is an extension of NAL. The introduction of structural joins is guided by the general rule:

$$e_j \left( \Phi \circ Y_{c_j / a; \Delta_i} (\square) \right) = \Phi(e_j \bowtie \theta_{a; e_j} Y_{c_j; \Phi_i} (\square))$$

(1)

At the left-hand side, the outer expression $e_j$ generates a sequence of tuples containing an attribute $c_j$. For each tuple, this attribute is the starting point for the calculation of the axis step in the dependent unnest map expression. $\Phi$ is a function defined by a sequence of already translated algebra operators (i.e., $\Phi$ does not contain any D-Joins). Note, $\Phi$ may not only contain unary operators (as our notation suggests), but also binary ones (like $\bowtie$). However, because we assume $\Phi$ to be already translated, the rewriting depends on the single inner expression $Y_{c_j / e_j / a; \Delta_i}$.

At the right-hand side, expression $e_j$ is shifted into $\Phi$ forming a structural join using the specified axis with a node sequence access $Y_{c_j; \Phi_i}$. This has the effect that $\Phi$ consumes a slightly different input sequence, because it now contains also attributes from $e_j$. While, on the left-hand side, the evaluation contexts are neatly separated, on the right-hand side, they are intermixed. Therefore, this rewriting is only correct for certain $\Phi$. We enumerate the variations of this rule for those $\Phi$, for which the above rule would lead to an incorrect rewriting. In the following cases, $\Phi$ is split into three operators, of which two ($\Phi_1$ and $\Phi_2$) are again functions containing sub-expressions and the third is the operator of interest.

- $\Phi = \Phi_1 \circ Tmp_{c_j} \circ \Phi_2$. This pattern leads to the following right-hand side, where the $Tmp_{c_j}$ operator has the same semantics as in the stacked translation (see [3]):

$$\Phi_1 \circ Tmp_{c_j} \circ \Phi_2(e_j \bowtie \theta_{a; e_j} Y_{c_j; \Phi_i} (\square))$$

(2)

Due to the rewriting, the different evaluation contexts are not separated anymore. The operator has to detect groups of attributes belonging to the same context. In the rule, expression $e_j$ binds attribute $c_j$, thus providing the outer context in which the
structural join is evaluated. \( Tmp^c_j \) detects groups based on \( c_j \), i.e., whenever this attribute changes its value, the start of a new group is indicated. In the following, we will call operators that have been modified in this way group aware.

- \( \Phi = \Phi_1 \circ X_{c:p\cdot\text{counter}(p)}\mathop{+++} \circ \Phi_2 \). For this pattern, we need to make the map operator group aware. Therefore, the expression \( X_{c:p\cdot\text{counter}(p)}\mathop{+++} \) has stacked-translation semantics (as defined in [3]):

\[
\Phi_1 \circ X_{c:p\cdot\text{counter}(p)}\mathop{+++} \circ \Phi_2(e_j \times c_i \theta_{a_j \epsilon_j} Y_{c_i \phi_i} (\square))
\] (3)

Because the order matters for that pattern, we have to be careful to match XPath’s semantics, which requires reverse document order, if a positional predicate is evaluated on a reverse axis. Therefore, if \( a_i \) is a reverse axis, we rewrite to:

\[
\Phi_1 \circ R^G_c \circ X_{c:p\cdot\text{counter}(p)}\mathop{+++} \circ R^G_c \circ \Phi_2(e_j \times c_i \theta_{a_j \epsilon_j} Y_{c_i \phi_i} (\square))
\] (4)

As with the \( Tmp^c_j \) operator, expression \( e_j \) provides the outer context, in which the structural join is evaluated. Therefore, the group reverse operator \( (R^G_c) \) groups by \( c_j \).

Likewise, the group-aware \( \text{counter}() \) function resets its counter, when \( c_j \) changes. Note, we will abbreviate that function by \( \text{ct}() \) in the following.

- \( \Phi = \Phi_1 \circ \mathcal{A}_x : f \circ \Phi_2 \). If the pattern contains an aggregate function, we have to apply nesting first and evaluate the aggregate function on the nested attribute. Afterwards, the nested attribute can be projected out:

\[
\Phi_1 \circ \mathcal{A}_x : f \circ \mathcal{A}_x : f \circ V_{x:e_j} \circ \Phi_2(e_j \times c_i \theta_{a_j \epsilon_j} Y_{c_i \phi_i} (\square))
\] (5)

- \( \Phi = \Phi_1 \circ \text{Sort}_{c:n} \circ \Phi_2 \). Here, a similar situation as in the previous rule can be found. We sort the nested group and unnest it again:

\[
\Phi_1 \circ \mu_g \circ \mathcal{S}_g \circ V_{x:e_j} \circ \Phi_2(e_j \times c_i \theta_{a_j \epsilon_j} Y_{c_i \phi_i} (\square))
\] (6)

For all other shapes of \( \Phi \), especially when \( \Phi \) is the identity function, rule (1) can be applied. Also, when an operator has already been made group aware, as for example the \( Tmp^c_j \) operator, (1) is used. If any \( \Phi \) contains multiple matchings of the given pattern, they are applied in parallel. This typically happens for rules (2) and (3) in case of a positional predicate, i.e., \([\text{position}()] = \text{last}()\).

We conclude this section with the rewriting of a simplified version of the previous example:

\( /\text{child} :: a /\text{child} :: b [\text{position}()] = \text{last}() /\text{child} :: c \) (Figure 2). In the first step, \( e_1 \) and the depending sub-expression can be identified as depicted in Figure 2a. With \( \Phi \) being the identity function, rule (1) can be applied. In Figure 2b, \( \Phi \) contains a structural join, a selection, a \( Tmp^c_j \), and a map operator. Here, rules (2) and (3) are used “simultaneously”. For Figure 2c, rule (1) applies again. Note, the position-handling operators have already been made group aware in the previous step.
5 Query Unnesting

With the introduction of the structural join into a NAL expression, we abandoned the explicit node-at-a-time path processing inherent to the D-Join operator. But still, the implicit node-at-a-time processing resulting from the evaluation of path steps in selections is present. In this section, we will provide a set of unnesting rules to “expose” these hidden path step evaluations. We do not claim to have found all interesting rewritings possible, but we think, we cover the most common cases.

In this section, we will introduce unnesting rules for existential, conjunctive, disjunctive, and negated predicates. Furthermore, we will consider predicates based on aggregate functions. In all nested expressions, we assume relative path expressions to be present\(^{11}\). Our query unnesting strategies are not covered by the rules in [4, 17]. Both contributions do not base their rewritings on the structural join operator.

The General Rules. By an examination of a typical operator tree, you can see that a pair of map operators \(\chi_{cn,cn}, \chi_{cn,cn}^{+}\) often “glues” an outer expression to the subscript of a

---

\(^{11}\)Selections without nested path expressions are considered to be constant or positional.
selection. Due to the XPath-to-NAL compilation, this is always the case when a predicate contains a path expression (for an example, see Figure 1). The inner map is the starting point for a cascade of operators, the first one of which is a structural join (in NAL contains a path expression (for an example, see Figure 1). The inner map is the starting goal is to “extract” the inner path expression and join it with the outer expression. In some cases, we can abandon the select operator completely. In other cases, we have to adjust the subscript to the new situation, using variable references to access necessary information, now produced in the outer expression. In the simple case, when the XPath predicate (and accordingly the selection subscript) contains only one relative path expression, we use the following generic unnesting rule:

\[ \sigma_{\Phi}(\pi_{(\chi_{0}\circ cn(\square))})(\chi_{cn(c_0)}(e_0)) = \prod_{\Pi} \circ \sigma_{\Phi}(s_g) \circ \nu_{g,A[\pi']}((e_0 \ni c_1) \theta_{c_0} \pi') \]  

(7)

In the left-hand side of this rule, you can find the above mentioned pair of map operators: The outer expression \( e_0 \) binds attribute \( c_0 \), which is then mapped onto \( cn \); in the inner expression, \( e_0 \) is reestablished from the context attribute \( cn \). Variable \( \pi \) is a NAL\(^{STJ} \) path expression depending on the context node given by the outer expression, i.e., \( \pi = ((\chi_{0}\circ cn(\square)) \ni c_2) \theta_{c_1} e_1 \ldots \theta_{c_n} e_n) \). \( \Phi \) is—as in the previous rewriting rules—a sequence of NAL\(^{STJ} \) operators, but this time, it may not be the identity function. At the right-hand side we find a modified \( \pi' \). The inner path expression \( \pi \) is extracted and joined with the outer \( e_0 \), using attribute \( c_1 \) of \( \pi \) in the join condition. Note, there is no need for map operators anymore, i.e., \( \pi' \) does not depend on \( \chi_{cn(cn(\square))} \). This means that \( \pi' \) now has the form \( \pi' = (e_1 \ni c_2) \theta_{c_1} e_1 \ldots \theta_{c_n} e_n) \). We denote this circumstance by the omission of the argument of \( \pi' \). Furthermore, in the following, we will simply abbreviate \( \pi((\chi_{0}\circ cn(\square))) \) occurring in a selection subscript by \( \pi^2 \). To handle different evaluation contexts, a nest operator is inserted, which groups by all attributes, except those of \( \pi' \). The selection is now executed on the grouped \( \pi' \), referencing the group by the variable \( s_g \). After the selection, no information about the path \( \pi' \) is required anymore. Therefore, it is projected out. While this rule is directly applicable, there are further refinements for special cases that provide for better performance results.

In contrast to all previous rewriting rules, \( \Phi \) may not be unary anymore, because in one predicate, several path expressions can be evaluated “simultaneously”. This leads to a generic unnesting rule for the complex case, when multiple path expressions are located in a single attribute:

\[ \sigma_{\Phi}(\pi_1 \ldots \pi_n)(\chi_{cn(c_0)}(e_0)) \]

(8)

\[ = \prod_{\Pi} \circ \sigma_{\Phi}(s_{g_1} \ldots s_{g_n}) \circ \nu_{g_1,A[\pi_1]} \circ \ldots \circ \nu_{g_n,A[\pi_n]}((e_0 \ni c_1) \theta_{c_0} \pi_1) \ldots \theta_{c_n} e_n(\pi_n)) \]

Here, \( \Phi \) is \( n \)-ary, depending on a set of path expressions. Because all path expressions are evaluated in the same local context, the depicted nesting is actually possible: no nesting of already nested sequences may occur. The only critical issue arising is the calculation of a nesting, where attributes compared for equality may be sequence valued. This is, however, not a problem of the logical algebra, but has to be solved at the physical level. One strategy, for example, would be to abandon the nest operators and modify the subsequent operators to make them group aware\(^{12} \). Another possible solution is to integrate the generation of nested groups into physical structural join operators, as sketched in [13].

\(^{12}\)This technique has already been applied in the stacked translation, where the \( Tmp^\ast \) operator is converted to a group-aware \( Tmp^\ast \) operator.
Rewriting Conjunctive Predicates. Whenever possible, we normalize the subscripts of selections into a disjunctive form, i.e., \( e_1 \land (e_2 \lor e_3) = (e_1 \land e_2) \lor (e_1 \land e_3) \). We are aware that, by multiplying \( e_1 \), common sub-expressions are introduced. Again, this is not a problem for the logical algebra, but the physical plan generator has to deal with it. Every time we have to introduce common sub-expressions, we give the plan generator a hint to signal their correspondence.

The first rewriting handles conjunctive expressions. For them, we rewrite the query using the well-known equivalence:

\[
\sigma_{e_2 \land e_3}(e_1) = \sigma_{e_2} \circ \sigma_{e_3}(e_1) = \sigma_{e_3} \circ \sigma_{e_2}(e_1)
\]  

(9)

Rewriting Disjunctive Predicates. Disjunctive predicates may be handled similarly to conjunctive ones using the sequence merge operator:

\[
\sigma_{e_2 \lor e_3}(e_1) = \sigma_{e_2}(e_1) \lor \sigma_{e_3}(e_1) = \sigma_{e_3}(e_1) \lor \sigma_{e_2}(e_1)
\]  

(10)

Again, this rewriting requires special care from the plan generator to handle the multiplied occurrences of expression \( e_1 \). When sub-expressions of the disjunction are aggregated using the \( \text{exists()} \) function, they can be extracted by using left-outer joins:

\[
\sigma_{\exists_{\alpha_1}(\pi_x) \lor \exists_{\alpha_2}(\pi_x)}(\chi_{\pi_0}(e_0)) = \Pi^D_{A(e_0)} \circ \sigma_{(A(\pi') \neq e)}(e_0 \exists_{\pi_1 \theta_{e_0} \pi'})
\]  

(11)

The notation \( A(\pi') \neq e \) essentially has the meaning \( \forall a \in A(\pi') : a \neq e \), i.e., \( \pi' \) has provided a join partner in the left outer join. In all other cases, when multiple path expressions in a general disjunction may occur, the query can be rewritten as:

\[
\sigma_{\Phi_1(\pi_1') \lor \Phi_2(\pi_2')}(\chi_{\pi_0}(e_0)) = \Pi_{A(e_0)} \circ \sigma_{(\Phi_1(S_{\pi_1}) \lor \Phi_2(S_{\pi_2}))}(e_0 \exists_{\pi_1 \theta_{e_0} \pi_1'} \exists_{\pi_2 \theta_{e_0} \pi_2'})
\]  

(12)

In the following, every time a path expression participates in a disjunction, we use a left outer join operator instead of a full join. This guarantees, that we do not accidentally “throw away” intermediate results. For example, in the expression \( a[b \lor c] \) we may not use an ordinary join between \( a \) and \( b \), because then we would miss all \( a \) elements which should be part of the final result due to \( c \).

Unnesting Existential Predicates. Sometimes plain path predicates like in \( a[b/c] \) occur. In NAL\(^{STJ} \), those expressions are compiled to an aggregation in combination with an \( \text{exists} \) in the selection subscript. They can be unnested with the following rule, introducing a semi-join operator:

\[
\sigma_{\exists_{\alpha}(\pi_x)}(\chi_{\pi_0}(e_0)) = e_0 \bowtie_{\pi_0} \pi'
\]  

(13)
Note, on the right-hand side, \( \pi' \) is evaluated first, before the structural join is computed. Essentially this means, that \( \pi' \) is not evaluated in the context of \( \epsilon_1 \) anymore. This could be problematic, if \( \pi' \) returns a large number of intermediate tuples. Another solution is viable as well, where path expression \( \pi \) is exposed:

\[
\sigma_{\exists e_{\text{exist}}(\pi \cdot \chi_{e \in c_0}(e_0))} = \Pi_A(e_0)( (e_0 \Join e_1 \theta_{c_0} e_1) \cdots \Join e_{c_n} \theta_{c_n} e_n )
\]

(14)

In the case of a negated path predicate, e. g., \( \text{a}[\text{not}(b/c)] \), we use an anti-join operator:

\[
\sigma_{\neg \exists e_{\text{exist}}(\pi \cdot \chi_{e \in c_0}(e_0))} = e_0 \join_{\epsilon_1 \theta_{c_0}} \pi'
\]

(15)

Unnesting Path Comparison Expressions. In the NAL compilation process, predicates of the form \([e_1 \theta e_2]\) are translated into an \( \exists_{\text{exist}} \) predicate. Therefore, with the first rule above, we can also unnest predicates that contain a comparison of a path with a constant (simple path comparison expression). For example, the query \( \text{a}[[b > 3]] \) can be translated and unnested into the NAL \( \text{STJ} \) expression\(^{13}\) \( \Pi_D^2(\chi_{e \in c_1}(\chi_{e \in c_2}(\chi_{e \in c_3}(\chi_{e \in c_4}(e_0))))) \). However, because \( \Phi \) is unary, this rewriting rule does not provide any help in case of complex path comparison expressions like \( \text{a}[[b/text() = c/text()]] \). In such a case, the following unnesting rule can be applied.

\[
\sigma_{\exists e_{\text{exist}}(\Phi_0(\pi_1, \pi_2))} = \Pi_{A(e_0)} \circ \sigma_{\exists(e_1 \theta e_2)}( (e_0 \Join e_1 \theta_{c_0} \pi'_1) \Join e_2 \theta_{c_0} \pi'_2 )
\]

(16)

In this rule \( \Phi_0 \) is the compilation of the existential comparison as introduced in [3]. For example \( \pi_1 = \pi_2 \) would be compiled into \( \exists_{\text{exist}} \pi_1 \times \pi_2 \). Rule (13) is promising, because it may be implemented very efficiently. At the right-hand side, the selection operator simply compares two attributes. This comparison has non-existential semantics, in contrast the existential semantics on the left-hand side. The generated tuple stream is in document order. Therefore, the duplicate elimination operator is simply a buffered filter with a buffer size of one tuple. This is also true for the duplicate elimination in rule (14).

Unnesting Predicates with Aggregate Functions. If the nested sub-expression contains an aggregate function, e. g., as in \( \text{a}[[\text{count}(b) = 3]] \), we can unnest this query using a group-by in combination with the aggregate function:

\[
\sigma_{\Phi(\pi_1)}(\chi_{e \in c_0}(e_0)) = \Pi_A(e_0) \circ \sigma_{\Phi(\pi_1)}(\chi_{e \in c_0}(e_0)) \circ \Gamma_{\exists e_{\text{count}}(e_0) = f}(e_0 \Join e_1 \theta_{c_0} \pi')
\]

(17)

An Unnesting Example. We will conclude the discussion of query unnesting with an example. To save space, this example is presented using formulas. Consider the XPath expression \( /\text{desc} :: \text{a}[[\text{child} :: c = "foo" \lor \text{count}(\text{desc} :: \text{b}) > 3]] \). The nested NAL \( \text{STJ} \) query is:

\[
\sigma_{(\text{child} :: c)}(\chi_{e \in c_1}(e_0)) \quad \text{where} \quad e_0 = \chi_{e \in \text{root}(\text{cn})}(\square) \Join e_1 \theta_{c_0} \chi_{e \in \text{child}(\square)}(\square)
\]

\[
s_1 = \chi_{e \in \text{exist}}(\chi_{e \in \text{cn}}(\square) \Join e_2 \theta_{c_0} \chi_{e \in \text{child}(\square)}(\square)) = "foo"
\]

\[
s_2 = \chi_{e \in \text{count}}(\chi_{e \in \text{cn}}(\square) \Join e_3 \theta_{c_0} \chi_{e \in \text{child}(\square)}(\square)) > 3
\]

\(^{13}\)Because 3 is a constant, we do not compile it using an aggregation, e. g., \( \exists_{\text{max}}(\text{b}) \), as suggested in [3].
In the first step, we use Rule (11) to extract the exists() part of the disjunction:

$$\Pi_{\varphi \neq e} \circ \sigma_{\varphi \neq e} \circ \chi_{\text{count}}(e_1) \quad \text{where} \quad e_1 = (e_0) \land \gamma_{\chi_{\text{count}}(\emptyset)} \land \text{“foo”}$$

$$s_2 = \chi_{\text{count}}(\emptyset) \land \text{“foo”}$$

In the second step, we use Rule (17) to extract the aggregate function from the disjunction:

$$\Pi_{\varphi \neq e} \circ \sigma_{\varphi \neq e} \circ \chi_{\text{count}}(e_2) \quad \text{where} \quad e_2 = (e_1) \land \gamma_{\chi_{\text{count}}(\emptyset)} \land \text{“foo”}$$

Finally, a slight optimization regarding expression $e_1$ can be pointed out: $\gamma_{\chi_{\text{count}}(\emptyset)} \land \text{“foo”}$ can simply be integrated into the node sequence access. In a physical algebra, this type of access could be supported by an index.

6 Pushing Down Structural Joins

The mapping of a logical algebra expression to its corresponding physical one is out of the scope of this paper. However, when thinking about this mapping, two interesting questions arise: How can a logical expression be “prepared” to facilitate the logical-to-physical algebra mapping and how can the problem of structural join order selection be tackled? We think the answer to these questions lies in a special operator tree format, where the tuple-generating structural joins are located at the bottom of the tree, and filtering/selection operators occur as inner nodes. In this representation, logically related path processing operators are situated close to each other. Because no selections or other operators interfere, it is easy to determine the different parts to be mapped onto a HTJ join operator, a path index access, or onto the STJ operator. This operator tree format can be generated by lifting non-structural join operators out of either side of a structural join operator. For example, in Figure 2d, from the left side of the final structural join, the selection, $\text{Temp}$, and $\chi$ operators could be lifted, pushing down the structural join to the bottom of the tree. In Table 1 we provide rewriting rules to accomplish such restructurings. We are aware that these rules have an immediate impact on the costs of the query, because the evaluation

<table>
<thead>
<tr>
<th>Operator</th>
<th>Rule</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_p$ (Selection)</td>
<td>$\sigma_p(e_1) \land \text{test} \land e_2 = \sigma_p(e_1 \land \text{test}, e_2) \land F(p) \land A(e_1) = \emptyset$</td>
<td>$e_1 \in A$</td>
</tr>
<tr>
<td>$\Pi_A$ (Projection)</td>
<td>$\Pi_A(e_1) \land \text{test} \land e_2 = \Pi_A(e_2) \land \text{test}$</td>
<td>$e_2$ duplicate free</td>
</tr>
<tr>
<td>$\Pi_D$ (Dup. Elim.)</td>
<td>$\Pi_D(e_1) \land \text{test} \land e_2 = \Pi_D(e_2) \land \text{test}$</td>
<td>$A \land A(e_2) = \emptyset$</td>
</tr>
<tr>
<td>$\Pi_P$ (Projection)</td>
<td>$\Pi_P(e_1) \land \text{test} \land e_2 = \Pi_P(e_2) \land \text{test}$</td>
<td>$e_1 \in A$</td>
</tr>
<tr>
<td>$\Gamma_{\chi \land A, f}$ (Group)</td>
<td>$\Gamma_{\chi \land A, f}(e_1) \land \text{test} \land e_2 = \Gamma_{\chi \land A, f}(e_1 \land \text{test}, e_2)$</td>
<td>$e_1 \in A$</td>
</tr>
<tr>
<td>$\nu_g$ (Constr.)</td>
<td>$\nu_g(e_1) \land \text{test} \land e_2 = \nu_g(e_2) \land \text{test}$</td>
<td>$e_1 \in A$</td>
</tr>
<tr>
<td>$\mu_h$ (Unnest)</td>
<td>$\mu_h(e_1) \land \text{test} \land e_2 = \mu_h(e_2) \land \text{test}$</td>
<td>$e_1 \in A$</td>
</tr>
</tbody>
</table>
of selections—minimizing the intermediate result size—is deferred. However, using the same set of rules, these selections may be pushed back into their original places, after the logical-to-physical mapping has been performed.

Because these equivalences may be read from either side, they also provide a way to push down non-structural operators. Again, we do not claim to have found all interesting rewritings possible here. In addition to the rules depicted in Table 1, we have found rules to push down a join over the special operators $tmp_{ij}$ and $\chi_{c,xt}(p)++$. However, their discussion is beyond the scope of this paper.

7 Quantitative Results

To substantiate our findings, we compared the different evaluation strategies by a one-to-one comparison on a single-user system. We implemented the operators of the NAL$^\text{STJ}$ algebra in the XTC system. Because we wanted to keep the comparison between a pure NAL expression and the NAL$^\text{STJ}$ variants of a query simple and, because we do not elaborate on a sophisticated logical-to-physical algebra mapping in this paper, we just used the algorithm presented in [1] for the implementation of the structural join.

System Testbed. XTC is one of the few native database systems, providing fine-grained transaction isolation over shared XML documents. In XTC, each XML node has a unique stable path labeling identifier (SPLID [11]). We refined the ORDPATH [19] concept for the implementation of SPLIDs. For document storage, each node is mapped onto a record, containing the SPLID and the encoded node data. All records of a document are stored in a B*-Tree, comprising the document container.

Furthermore, the element index provides for fast access to elements with the same element name (see Figure 3). It is a two-way index, consisting of a name directory (B-Tree) and a set of node-reference indexes. Given a context node $cn$, the element index can be used to calculate the sequence of all elements having a specific name on a specific axis. Such queries are simply translated to range queries over a particular node-reference index. This is exactly, how we implemented the evaluation of the $\Upsilon$ operator. XPath predicates subject to the value content of XML nodes are evaluated on the document index.

Query Workload. The query workload depicted in Table 2 was run on four XMark documents of size 120 KB, 1.2 MB, 12 MB, and 112 MB (factors 0.001, 0.01, 0.1, 1). To compare the raw performance of the given strategies, we switched off isolation mechanisms in XTC, thus, no locking overhead occurs. Each query was compiled into the pure NAL stacked translation and into its (optimized) unnested equivalent in NAL$^\text{STJ}$. To address various XPath use cases, we tested the following types of queries: a purely
structural query, a query relying on position, a content-based query, and a query with aggregations. For the structural query, the NAL expression does not examine all dependent paths in the path predicate. When the first matching path is found, the evaluation of the predicate is accomplished.

Results. Our tests were carried out on an Intel XEON computer (four 1.5 GHz CPUs, 2 GB main memory, 300 GB external memory, Java Sun JDK 1.5.0) as the XDBMS server machine and a PC (1.4 GHz Pentium IV CPU, 512 MB main memory, JDK 1.5.0) as the client, connected via 100 MBit ethernet to the server. All tests were issued on a hot DB buffer of 250 16KB-sized pages.

Our first observation is that the figures of all queries look very similar. On the small document, both NAL and NAL\textsuperscript{STJ} show the same performance. However, as the documents and the result sizes grow larger, the NAL\textsuperscript{STJ} optimized expressions are roughly one magnitude

<table>
<thead>
<tr>
<th>No</th>
<th>Query</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>//closed_auction/annotation/description</td>
<td>parlist/listitem/text/keyword</td>
</tr>
<tr>
<td>Q2</td>
<td>//open_auctions/open_auction/bidder/position() = last() ∨ position() = 1</td>
<td>positional</td>
</tr>
<tr>
<td>Q3</td>
<td>//item[//date = “20/07/2000” ∧ /payment = “Creditcard”]</td>
<td>content based</td>
</tr>
<tr>
<td>Q4</td>
<td>//item[count(.//text/</td>
<td>bold</td>
</tr>
</tbody>
</table>

Figure 4: Queries Q1, Q2, Q3, and Q4
faster (note, we used the log scale on the x-axis and the y-axis). The only exception is the content-based query Q3. Furthermore, we notice that both strategies scale with respect to the size of the input document.

The major explanation for the above effects is the relation between node-at-a-time path processing (D-Joins in NAL) and set-at-a-time path processing (structural joins in NAL\textsuperscript{STJ}). For example in NAL, query Q4 is evaluated by accessing all \textit{item} elements and, for each such element, evaluating the predicate. This implies a repeated access to the element index to scan the depending predicate paths. In contrast to this, set-at-a-time requires only few element index scans which are carried out in a sequential fashion. On small documents where only a few intermediate tuples occur, the distinction between the two processing styles does not carry much weight. However, when the element index has to be accessed over and over again, e.g., due to a large input in a selection predicate, access costs explode.

The problem with query Q3 is that, for NAL\textsuperscript{STJ}, Q3 also requires node-at-a-time processing to evaluate the content predicate. This is due to the lack of a content index in the XTC system. If we could access text nodes carrying the same content as easily as element nodes with the same element name, then it would also be possible to evaluate the equality predicate using a structural join.

We are aware that all presented queries could be evaluated faster, if suitable measures on the mapping from the logical to the physical algebra were taken. For example, Q1 could be answered more easily with the help of a structural index, even if only a sub-path of the query could be evaluated by that index. For queries with positional predicates, special evaluation algorithms resembling structural joins have been proposed [24]. In query Q3, a text index, as sketched above, would be very beneficial. A structural join reordering could take the selectivity of the text predicate into account and start the evaluation by the computation of a structural join between the \textit{date} elements and the value \textquote{07/05/2005}. However, to keep the two strategies comparable, we contented ourselves with the simple mapping sketched above.

8 Conclusions

To the best of our knowledge, this is the first article dealing with the introduction of the structural join operator into a tuple-based XPath algebra. With our contribution, we hope we can bridge the gap between the many promising algebra proposals on one side and the equally many proposals on evaluation algorithms (physical operators) for XML queries on the other side. We are aware that this is only an initial step towards the integration of these valuable concepts, because many problems regarding the logical-to-physical algebra mapping are still left out, e.g., join reordering, cost-based optimization, etc.

With the structural join, it is now possible to free an operator plan from implicit (selections) and explicit (D-Join) node-at-a-time processing steps. Note, this is accomplished at the logical level only; a physical implementation may freely choose to implement a structural join in a node-at-a-time manner [14], nevertheless. Even hash-based strategies may be applied [15]. But the decision to do so depends on physical issues and should not be determined at the logical level.
Finally, even with the given simple mapping from a logical algebra expression to a physical one (taking only the algorithm from [1] into account), we gained an order of magnitude in the performance of query evaluation. And more sophisticated mappings are still to come, from which we hope to gain further improvements.

Acknowledgements. I would like to thank Theo Härder, Jose de Aguiar Moraes Filho and the anonymous referees for their valuable comments on this paper. The support of Andreas Bühmann while formatting the final version is appreciated.

References


260
A NAL Overview

Table 3: Relevant NAL Operators taken from [3]

<table>
<thead>
<tr>
<th>Operator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>( \sigma_{p}(e) := \begin{cases} a(e) \circ \sigma_{p}(\tau(e)) &amp; : p(a(e)) = \text{true} \ \sigma_{k}(\tau(e)) &amp; : \text{else} \end{cases} )</td>
</tr>
<tr>
<td>Projection</td>
<td>( \Pi_{s}(e) := a(e)</td>
</tr>
<tr>
<td>Map</td>
<td>( \chi_{e_{1} \times e_{2}} := (a(e_{1})) \circ \chi_{e_{2}} \oplus (\tau(e_{1})) \circ \chi_{e_{2}} )</td>
</tr>
<tr>
<td>Cross Product</td>
<td>( e_{1} \circ e_{2} := (a(e_{1})) \circ e_{2} \oplus (\tau(e_{1})) \circ e_{2} )</td>
</tr>
<tr>
<td>D-Join</td>
<td>( e_{1} \circ e_{2} := a(e_{1}) \times \tau(e_{1}) \oplus (\tau(e_{1})) \times e_{2} )</td>
</tr>
<tr>
<td>Product</td>
<td>( \eta_{1} \circ \eta_{2} := (\eta_{1} \circ a(e_{2})) \oplus (\eta_{1} \circ \tau(e_{2})) )</td>
</tr>
<tr>
<td>Semi-Join</td>
<td>( e_{1} \circ e_{2} := \begin{cases} a(e) \circ (\tau(e_{1}) \times \tau(e_{2})) &amp; : \exists x \in e_{2} : p(a(e_{1}) \times x) \ \tau(e_{1}) \times \tau_{e_{2}} &amp; : \text{else} \end{cases} )</td>
</tr>
<tr>
<td>Anti-Join</td>
<td>( e_{1} \circ e_{2} := \begin{cases} a(e) \circ (\tau(e_{1}) \times \tau(e_{2})) &amp; : \exists x \in e_{2} : p(a(e_{1}) \times x) \ \tau(e_{1}) \times \tau_{e_{2}} &amp; : \text{else} \end{cases} )</td>
</tr>
<tr>
<td>Unnesting</td>
<td>( \mu_{s}(e) := (a(e)) \times e_{2} \oplus \mu_{s}(\tau(e)) )</td>
</tr>
<tr>
<td>Unnest-Map</td>
<td>( \Gamma_{\epsilon_{2}}(e_{1}) := \mu_{s}(\chi_{\epsilon_{2}}(e_{1})) )</td>
</tr>
<tr>
<td>Binary Grouping</td>
<td>( e_{1} \Gamma_{\epsilon_{A \epsilon_{B} \epsilon_{C} \epsilon_{2}}} := a(e_{1}) \circ (g : G(a(e_{1}))) \oplus (\tau(e_{1})) \Gamma_{\epsilon_{A \epsilon_{B} \epsilon_{C} \epsilon_{2}}} )</td>
</tr>
<tr>
<td></td>
<td>( \text{where } G(s) := f(a_{1}, \theta_{1}(e_{2})) )</td>
</tr>
<tr>
<td>Unary Grouping</td>
<td>( \Gamma_{\epsilon_{A \epsilon_{B} \epsilon_{C} \epsilon_{2}}}(e_{1}) := \Pi_{\epsilon_{A \epsilon_{B} \epsilon_{C} \epsilon_{2}}}(\Pi_{\epsilon_{B \epsilon_{C} \epsilon_{2}}}(\Gamma_{\epsilon_{A \epsilon_{B} \epsilon_{C} \epsilon_{2}}} e_{1})) )</td>
</tr>
<tr>
<td>Aggregation</td>
<td>( \mathbf{A}<em>{e</em>{1}}(e) := [a : f(e)] )</td>
</tr>
<tr>
<td>Sort</td>
<td>( \text{Sort}<em>{s}(e</em>{1}) := \text{Sort}<em>{s}(\mathbf{A}</em>{e_{1}}(a(e), e_{1} \circ \tau(e))) \oplus a(e) \oplus \text{Sort}<em>{s}(\mathbf{A}</em>{e_{1}}(a(e), e_{1} \circ \tau(e))) )</td>
</tr>
<tr>
<td>Singleton Scan</td>
<td>( \square := [1] )</td>
</tr>
</tbody>
</table>
Anreizmechanismen für Peer-to-Peer Web Crawling unter Berücksichtigung bösertiger Teilnehmer

Holger Steinhaus
Institut für technische und betriebliche Informationssysteme
Otto-von-Guericke-Universität Magdeburg
39106 Magdeburg
hsteinha@cs.uni-magdeburg.de

Klemens Böhm, Stephan Schosser
Institut für Programmstrukturen und Datenorganisation
Universität Karlsruhe (TH)
76131 Karlsruhe
{boehm,schosser}@ipd.uni-karlsruhe.de


1 Einleitung

1.1 Problembeschreibung


\(^1\) Ein Vorabexperiment zur Analyse der Webstruktur wurde auf einer Breite von 12 Mio. gecrawlnen Webseiten (ca. 0.1% der geschätzten Größe [GS05] des Webs im Jahr 2005) durchgeführt. Das Speichern der Linkstruktur der besuchten Seiten (ohne deren Inhalt) führte zu einer Datenbank mit 1 Mrd. Tupeln, die etwa 80 GB an permanentem Speicherplatz benötigt.

1.2 Related Work


1.3 Annahmen

Die Effektivität unserer Lösung ist an einige Annahmen gebunden, die im Folgenden beschrieben werden:


Diskrete Nutzenfunktion. Der Nutzen eines Teilnehmers ist 1, wenn für den betreffenden Teilnehmer Anfragen an das System erlaubt sind, sonst 0. Der Nutzen hängt somit nicht von der Zahl der erlaubten Anfragen ab. Diese Annahme entspricht einem Flatrate-Abrechnungsmodell, das in vielen Bereichen erfolgreich eingesetzt wird.

2 Verteiltes Webcrawling


Unser Ansatz geht von einer partitionsbasierten Verteilung der Arbeitslast aus, die keine solche Zentralinstanz benötigt. Die Partitionierung wird mit Hilfe einer Hashfunktion erreicht, die URLs von Webseiten auf einen numerischen Bereich fester Länge abbildet\(^2\). Anhand dieses Wertes kann eine verteilte Hashtabelle (DHT) aufgebaut werden, beispielsweise unter Verwendung von Protokollen wie CAN [R+01] oder Chord [S+01]. Abgesehen von der allen Peers bekannten Hashfunktion ist keine darüber hinausgehende Koordination erforderlich. Jeder Teilnehmer kann lokal entscheiden, ob eine bestimmte URL zu seiner Partition gehört. URLs, die nicht zur lokalen Partition gehören, können mit Hilfe

\(^2\)Wir verwenden dazu eine kryptographische Hashfunktion (MD5), die aufgrund ihres Entwurfszwecks auch sehr ähnliche Eingangswerte sehr gleichmäßig über den gesamten möglichen Wertebereich streut.
des DHT-Protokolls transparent auf den entsprechenden entfernten Teilnehmer abgebildet werden.

Die Partitionierung stellt weiterhin sicher, das einzelne Seiten nicht mehrmals gcrawlt werden, ohne dabei aufwändige verteilte Sperrmechanismen anwenden zu müssen. Die Vermeidung redundanten Crawls ist wichtig, da nicht nur Ressourcen des Crawlers gebunden werden, sondern auch die des gcrawlt Webserver. Folglich betrachten viele Serverbetreiber redundantes Crawling als ’unfreundlich’ und sperren solche Crawler mittels technischer Maßnahmen künftig aus.

3 Verhalten von Teilnehmern

3.1 bösertige vs. passive Teilnehmer


- Er übermittelt regelmäßig und wahrheitsgemäß Feedback über Erfahrungen mit anderen Teilnehmern.
- Er beteiligt sich in der Pflege eines Reputation Repository.
- Er beteiligt sich an der Erkennung und Sanktionierung nicht-kooperativer Teilnehmer.
- Er beantwortet Anfragen von dazu berechtigten kooperativen Teilnehmern.


**Anforderungen.** Zusammenfassend müssen die beabsichtigten Mechanismen die folgenden Anforderungen erfüllen:

- Free-Riding darf keine dominante Strategie sein. Ein sich nicht beteiligender Peer darf keinen Nutzen aus der Arbeit Anderer ziehen können.
- Um Angriffe bösertiger Teilnehmer abwehren zu können, muss das System in der Lage sein, Angreifer auszuschließen.
- Nicht crawlinge Teilnehmer werden nicht ausgeschlossen, da sie dem System keinen Schaden zufügen.

### 3.2 Identifikation und Behandlung bösertiger Peers


Auf einem höheren Abstraktionsniveau betrachtet, besteht der Grund für die Teilnahme eines kooperativen Peers am hier beschriebenen P2P-System in der Erzielung von Nutzen, der jedoch an der Existenz des Systems gebunden ist. Ein kooperativer Peer wird also sicher keine Anstrengungen unternehmen, die Existenz des Systems zu zerstören, so lange dafür keine Anreize bestehen. Feedback über andere Teilnehmer, egal ob negativ

Reputation. Grundsätzlich existieren verschiedene Ansätze, Feedback in verteilten Umgebungen zu sammeln und in möglichst manipulationssicherer Weise zu einem Reputationswert zu aggregieren [KSGM03, AD01]. Im Folgenden gehen wir von der Existenz einer solchen Technologie aus. Die Reputation eines Teilnehmers $i$ zum Zeitpunkt $t$ ergibt sich folgendermaßen:

$$R_t^{(i)} = \min\{R_{t-1}^{(i)} + \sum_j fb_j, R_{\text{max}}\}$$


Umgang mit neuen Teilnehmern. Von besonderer Bedeutung ist der Umgang mit neu hinzu kommenden Teilnehmern. Im hier vorgestellten Ansatz wird ihnen eine Reputation von $R_{\text{initial}}$ zugewiesen. Diese sollte der folgenden Bedingung genügen: $R_{\text{min}} < R_{\text{initial}} \leq R_{\text{max}}$. Die Wahl der initialen Reputation stellt dabei die Einstellung des Systems gegenüber neuen Peers dar. Ein hoher Wert führt zu einem System, was jeden neuen Peer als kooperativ betrachtet, solange keine Beschwerden über ihn eingehen. Die Wahl von $R_{\text{initial}}$ hat jedoch keine Auswirkungen auf die Beurteilung der Crawlingleistung eines Teilnehmers und dessen Möglichkeit, auf Nutzdaten zuzugreifen. Die Wahl eines hohen Wertes für $R_{\text{initial}}$ ist damit möglich, ohne Free Riddern die Tür zu öffnen. Jedoch erhöht ein zu hoher Wert die Chancen eines Angreifers, Schaden im System anzurichten; Es muss mehr negatives Feedback eingehen, bis der Angreifer erkannt und ausgeschlossen wird. Im Gegensatz dazu führt die Wahl einer initialen Reputation von $R_{\text{initial}} = R_{\text{min}} + \epsilon$

3.3 Erkennung passiven Verhaltens

Die Erkennung passiver Teilnehmer ist nicht so einfach, wie das auf den ersten Blick erscheinen mag: Da jeder Teilnehmer laut Annahme seine gesammelten Daten selbst speichert, gibt es keinen allgemeinen Weg, die Aktivität der einzelnen Teilnehmer zu beobachten. Die erbrachte Leistung ist jedoch ausgezeichnet definiert, wenn sie größer als die $s$-fache durchschnittliche Beteiligung aller kooperativen Peers ist. Um einen Crawlingfortschritt zu gewährleisten, genügt es, Werte von $s$ um 1.0 zu wählen. Unter der Bedingung, dass es wenigstens einen ständig crawlingen kooperativen Peer gibt, wird damit die durchschnittliche Leistung im System stets steigen, was wiederum andere Teilnehmer zum Crawlen motiviert. Jeder Teilnehmer kann die durchschnittliche Leistung recht einfach lokal beobachten: Jeder anfragende Teilnehmer muss seine bislang erbrachte Leistung mitteilen. Experimente haben gezeigt, dass, trotz der Zeit die einzelnen Teilnehmer ein recht zutreffendes Bild der durchschnittlichen Leistung durch gleitende Mittelwertbildung aufbauen können.


---

$^3$Das kleine Inkrement $\epsilon$ verhindert den sofortigen Ausschluss eines neu beitretenden Teilnehmers
1. **function verifyCrawlContrib( remotePeer, clContrib ) : boolean**
2. begin
3. Partition rpart = remotePeer.getPart()
4. Partition spart = randomSubPart(rpart, clContrib);
5. linkData = remotePeer.getSample(spart);
6. realC = countLinks(linkData);
7. if (realC * sizeof(rpart) / sizeof(spart) < clContrib) return FALSE;
8. return verifySample(linkData)
9. end

Abbildung 1: Der Verifikationsalgorithmus


Wir schlagen daher folgende Heuristik vor:

$$P_{ver} = \max \left( P_0 - \frac{R - R_{\text{min}}}{R_{\text{max}} - R_{\text{min}}} + 1; 1 \right)$$


### 3.4 Anfragebearbeitung

Teilnehmer ziehen ihren Nutzen aus dem System, indem sie Anfragen an Andere absetzen können. Der genaue Inhalt ist applikationsspezifisch, im Beispiel einer P2P-Suchmaschine wird eine solche Anfrage vermutlich aus Suchworten und eventuell weiteren Attributen
(wie z.B. Zeitraum der letzten Änderung, Sprache, Größe) bestehen. Die Anfragedbearbei-
tung besteht aus einem Sender- und einem Empfängerteil. Der Sender, der am Absetzen
eines Requests interessiert ist, muss eine hinreichende Beteiligung (siehe Abschnitt 3.3
als Parameter des Requests übermitteln. Ist die Beteiligung nicht hinreichend, wird ein
kooperativer Teilnehmer die Antwort verweigern. Abhängig vom Erhalt der Antwort gibt
der Absender des Requests entsprechendes Feedback.

Der Empfänger des Requests vergleicht die enthaltene behauptete Crawlingleistung des
Absenders mit seiner Schätzung der durchschnittlichen Leistung. Anschließend ermit-
telt er die Reputation des Anfragenden und startet mit einer gewissen Wahrscheinlichkeit
\( P_{ver}(R) \) eine Verifikation. Abhängig von deren Ergebnis übermittelt er entsprechendes
Feedback. Wenn die behauptete Crawlingleistung sich als wahr herausgestellt hat und sie
hinreichend ist, wird der Request beantwortet.

4 Evaluation

Abschnitt 3.1 listet die wichtigsten Typen unerwünschten Verhaltens auf. Um ein Modell
als Basis für unsere Experimente zu entwickeln, haben wir typische Muster von Benutzer-
fehlverhalten in heutigen Websystemen betrachtet. Dabei traten zwei Hauptbedrohungen
zu Tage: Free Riding und Denial-of-Service-Attacken. Die folgenden Experimente sollen
zeigen, dass die bislang beschriebenen Experimente damit umgehen können.

Zu diesem Zweck benötigen wir eine Art Worst-Case-Verhalten eines maximal destruk-
tiven Peers. Von besonderer Bedeutung ist dabei die Frage, ob ein solcher Peer crawlen
sollte. Falls er das tun würde, hat ein solcher bösnartiger Teilnehmer nahezu identische
Kosten wie ein kooperativer, da wir davon ausgehen, dass bei einem effizienten System-
entwurf die Aufwendungen für das Crawling alle Anderen bei weitem übersteigen. Zum
Angriff eines großen Systems aus vielen tausend Teilnehmern muss ein Angreifer damit
Ressourcen in einer ähnlichen Größenordnung wie der bereits im System vorhandenen
Kapazität zur Verfügung stellen. Verzichtet ein Angreifer jedoch auf das Crawling, würde
er einen erheblichen Ressourcenvorteil erzielen bzw. die vorhandenen erheblich destruktivi-
er einsetzen können. Diese Variante erscheint uns erheblich gefährlicher, so dass wir uns
in unserem Worst-Case-Modell darauf konzentrieren.

Das beabsichtigte Verhaltensmodell beruht auf der folgenden Grundidee: Ein bösnartiger
Peer crawlt nicht. Trotz allem behauptet er natürlich, gecrawlt zu haben. In der Folge wird
er jedoch keine Chance haben, eine Verifikation der Crawlingleistung zu bestehen. Um
maximalen Schaden anzurichten, gibt ein solcher Peer negatives Feedback über alle ko-
operativen Teilnehmer, zusätzlich ‚beschützt‘ er andere bösnartige Peers, indem er positives
Feedback übe sie abgibt (Verschwörungsprinzip).
4.1 Experimentierumgebung


Abbildung 2: Simulationsablauf

Abbildung 3: Neue Peers treten einem System aus kooperativen Teilnehmern bei
4.2 Sporadische Free-Rider


4.3 Koordinierter Angriff


Abbildung 4: Variation des bösartigen Anteils
Abbildung 5: Anzahl der Verifikationen vs. \(R_{\text{initial}}\)
Abbildung 6: Peeranzahl vs. \(R_{\text{initial}}\)

Als Nächstes betrachten wir die Auswirkungen der Wahl der initialen Reputation. Im Gegensatz zum letzten Experiment wird dabei der Anteil der bösartigen Teilnehmer fix auf 10% gesetzt, die initiale Reputation variiert dagegen. Abbildung 5 stellt die Zahl der Verifikationen als Funktion der initialen Reputation dar. In den ersten Zyklen wird die Verifikationsanzahl stark durch \(P_{\text{ver}}\) bestimmt. (vgl. Abschnitt 3.3). Die Dauer bis zur Erkennung bösgartiger Teilnehmer beeinflusst die weitere Entwicklung der Verifikationskosten. Da die Anzahl der bösgartigen Teilnehmer für kleine initiale Reputationen schneller abnimmt, nimmt die Zahl der Verifikationen hier ebenfalls schneller ab. Die absolute Anzahl ist minimal für hohe Werte von \(R_{\text{initial}}\).

als starkes anfängliches Misstrauen interpretieren kann, erlaubt den schnellen Ausschluss von vielen Teilnehmern - darunter auch kooperative. Aus Sicht dieses Experiments sollte \( R_{\text{initial}} \) so hoch wie möglich gewählt werden. Jedoch ist auch diese Wahl nicht optimal, da sie gleichzeitig bösartigen Teilnehmern ermöglicht, länger im System zu verbleiben und somit die Langzeitstabilität im Fall eines kontinuierlichen Zustroms (vgl. erstes Experiment) gefährden kann. Aufgrund des flachen Gradienten im Bereich von -0.2 bis 1 erscheint ein Wert im Intervall \((-0.2, 0)\) ein guter Kompromiss zu sein. Der in den anderen Experimenten verwendete Wert von \( R_{\text{initial}} = 0 \) liegt in diesem Bereich.

5 Schlussfolgerungen


Literatur


[GS05] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In WWW ’05: Special interest tracks and posters of the 14th international conference on World Wide Web, New York, NY, USA, 2005.


YAWN: A Semantically Annotated Wikipedia XML Corpus

Ralf Schenkel Fabian Suchanek Gjergji Kasneci
Max-Planck-Institut für Informatik, Saarbrücken, Germany
{schenkel,suchanek,kasneci}@mpi-inf.mpg.de

Abstract: The paper presents YAWN, a system to convert the well-known and widely used Wikipedia collection into an XML corpus with semantically rich, self-explaining tags. We introduce algorithms to annotate pages and links with concepts from the WordNet thesaurus. This annotation process exploits categorical information in Wikipedia, which is a high-quality, manually assigned source of information, extracts additional information from lists, and utilizes the invocations of templates with named parameters. We give examples how such annotations can be exploited for high-precision queries.

1 Introduction

1.1 Motivation

Much of the existing work on structure-aware XML retrieval [AY +02, S +05, Sch02] has anticipated the existence of a huge number of heterogeneous XML documents with descriptive (i.e., self-explaining and semantically rich) tags. This has always been predicted as the natural consequence of the flexibility and variety of XML, where every author of a Web page can invent a different schema for her data on the Web. Given a semantically rich structural query like //person[about(//work,physics) and about(//born,Germany)]¹ (i.e., find people who work in physics and were born in Germany), such search engines would either consider structural similarity metrics of the query and documents [AY +02, Sch02] or semantic similarity of the tags used in the query and in the documents, using an ontology as background knowledge [S +05].

However, this revolution still has to happen. XML, even though widely used nowadays, usually is either generated from a structured database or used to store textual information with some structure. In the latter case, documents are content-rich, but tags are generic, while in the former case, there may be meaningful tag names, but there is often not much textual content. Therefore, today’s typical XML search engines either ignore structural information completely, focussing on keyword queries alone, or deal with semantically weak structures as in //article[about(.,XML)]//section[about(./paragraph,retrieval)].

¹This query is formulated in NEXI, the query language of the INEX benchmark (http://inex.is.informatik.uni-duisburg.de/2006/), but could be expressed similarly in XPath with FullText extensions.
This paper bridges the gap between semantics-aware XML search engines and real world data by adding semantics to XML data, extending the ideas already used in SphereSearch \cite{G05} and, more recently, by Chu-Carroll et al. \cite{CC+06}. Based on the well-known and widely used Wikipedia collection, this paper presents YAWN\footnote{Yet Another Wikipedia Annotation project}, showing how to convert Wikipedia pages to XML that is annotated with concepts from WordNet, resulting in a huge annotated XML corpus with semantically rich tags. We exploit three sources of semantics: categorical information that has been added to most pages by the author (like “Albert Einstein is a physicist”), lists of similar pages, which are a common concept in Wikipedia (like lists of actors, songs, and companies), and invocations of predefined templates with human-readable parameters that encode factual information.

1.2 Related Work

There are a number of database-oriented XML benchmarks such as XMark \cite{S+02} and XMach \cite{BR01} that focus on performance aspects of XML databases. They usually provide no meaningful content and are therefore not suited for information retrieval. The INEX community has produced two XML benchmark collections, namely the IEEE collection and the Wikipedia collection \cite{DG06}, which combine huge corpora with queries, lists of relevant results, and a methodology to evaluate the quality of search results. However, these corpora do not include heterogeneous and semantically rich tags. The INEX Heterogeneous Track has started to collect different XML collections, but there are no self-explaining tags yet. A first step towards richly annotated data was made in the INEX Multimedia Track with the Lonely Planet collection\footnote{http://inex.is.informatik.uni-duisburg.de/2005/tracks/media/index.html}.

There have been several attempts to define a standard XML format for documents stored in Wikis, the most recent ones being the Wikipedia DTD\footnote{http://meta.wikimedia.org/wiki/Wikipedia_DTD(last change APR-09-06)} that proposes a set of tags similar to HTML, and DookBook XML Export\footnote{http://meta.wikimedia.org/wiki/DocBook_XML_export(last change JUN-19-06)} that defines XML tags for a subset of the Wiki markup. However, to the best of our knowledge, none of these covers the complete feature set of Wiki markup and is publicly available.

The Semantic Web community has recently launched a number of projects to add semantics to Wikis \cite{Aum05,BG06,Sou05,V+06} which typically aim at adding semantic information at design time. In contrast, our approach exploits information that is already present in Wikipedia pages, without the need for any user interaction.

Information extraction from text and HTML data is an area with intensive work. Agichtein \cite{Agi05} gives a survey of information extraction techniques with a focus on scalability in large collections with millions of documents. Approaches in the literature mostly either follow a rule-based paradigm \cite{AGM03,CMM02,G+04}, or employ learning techniques and/or linguistic methods \cite{AFG03,CS04,Cun02}. Our algorithms, unsupervised and statistical in their nature, fit in the second class. The list extraction method used in Section 3.3 is similar to list extraction methods used in other information extraction sys-
tems like KnowItAll [E‘05] and SCRAP [FFT05]. Rule-based information Extraction from XML documents has been considered by Abolhassani et al. [AFG03]; in contrast, we follow a purely automatic approach. Annotated XML collections and their use for information retrieval were considered by Graupmann et al. [G‘05] and Chu-Carroll et al. [CC‘06]; the techniques presented in this paper perfectly integrate with these works.

2 Converting Wiki Markup to XML

2.1 Wiki Markup

With more than 1.4 million articles as of October 2006, Wikipedia is the largest general purpose encyclopedia that is freely available. The content of pages in Wikipedia, like in other Wikis, is formulated in Wiki markup, a combination of standard HTML tagging with specific constructs for structuring, tables, links etc. There is no formal specification of the language and its semantics yet. Figure 1 shows a simple example for a document in this language. Important building blocks of the Wiki markup language include structuring text by defining several levels of sections, different levels of emphasis for text parts, bulleted and numbered lists of different nesting depths, tables with rows, columns, headers and captions, links within the Wikipedia collection and to the Web, and inlined images.

==Introduction==

‘Wiki markup’ is used in [[Wikipedia]].

==Language Components==
* tables
* lists
* and a lot more

==See also==
[http://www.wikipedia.org]

Figure 1: An example Wiki Markup page

In addition, Wiki markup can be arbitrarily mixed with HTML tags (which is most often used for layout purposes), and some Wiki markup symbols (like images and tables) may contain additional layout hints. Wiki markup also provides a template language where invocations of a template TEMP (in the form {TEMP}) are replaced with the definition of this template. Template invocations may also include values for parameters that then replace the parameter in the template’s definition. A Wiki2HTML converter (written in PHP) generates HTML pages (that neither have semantically rich tags nor are guaranteed to be well-formed XML) from the Wiki markup input.

The different components of Wiki markup can be combined almost arbitrarily. However, this huge flexibility is at the same time a big problem, as authors often tend to overstrain the features. As an example, many authors make frequent use of tables for layout, resulting in

6http://www.wikipedia.org/
a deep nesting of tables and (sub-)sections. Secondly, the fault-tolerant converter does not encourage correct markup; while this is adequate to create HTML, this makes it difficult to create well-formed XML.

2.2 Generating XML

This section shows how we generate XML from the Wiki markup of the Wikipedia pages. We focus on the content of the pages, accepting that some layout information is lost in this process. The complete textual content of Wikipedia is available as a huge XML file (as of April 2006, this was about 6GB), which contains one element for each Wikipedia page with some meta information and its content in Wiki markup.

Our Wiki2XML converter runs in two phases, where each phase corresponds to a SAX-style scan of the input XML document. In the first phase, the existing pages are collected and redirections (i.e., pages that are simply links to other pages) are resolved. In the second phase, an XML document is generated for each page, which consists of three parts: (1) the preamble that sets the character encoding and includes a pointer to an XSLT for presenting the page, (2) the article element with its header child, which in turn has children that specify meta data like the page title and id, the last revision, and the categories of this page, and (3) the body child of the article element that contains the XML representation of the page's content. The conversion of the Wiki markup to XML is done as follows:

- Sections, subsections etc. are converted to section, subsection etc. elements, each with an st child that contains the section title.
- Both numbered and bulleted lists are converted to list elements with entry children corresponding to the different list entries. Nested lists are represented by subentry, subsubentry etc. tags.
- Tables are represented by a table tag. Inside this tag, there is one row element for each row of the table, which in turn has one col tag for each column. For header rows, the tag header is used.
- Links to other pages in Wikipedia are converted to link elements that correspond to an XLink to the target page’s XML version. Links to web pages are converted to weblink tags with an XLink to the link target.
- Links to images are converted to image elements that correspond to an XLink to the image file located in a directory derived from the image name’s MD5 hash.
- Markup for emphasis is converted to b and it elements.

Figure 2 shows the XML generated for the example Wiki markup from Figure 1. The exact conversion of Wiki markup to well-formed XML is more complex (and in fact sometimes impossible without human interaction) if the markup is mixed with arbitrary HTML tags. This is especially a problem with some tables that are defined in a mixture of HTML and
Wiki markup, but also simple tags like <br> render a generated document malformed. To solve this problem, we eliminate all HTML tags from the Wiki markup in a preprocessing step. As those tags are typically used only for layout purposes, we do not lose any semantics, but can generate much better XML. Additionally, as the original Wiki2HTML converter is quite tolerant to syntax errors, there is a substantial number of Wikipedia pages with syntactic problems7. Each document is tested for well-formedness after it was generated, making sure that the collection contains only syntactically correct documents.

<?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="application/xml" href="../../wikipedia.xslt"?>
<article xmlns:xlink="http://www.w3.org/1999/xlink/">
  <header>
    <title>Wiki markup</title>
    <id>42</id>
    <revision>
      <timestamp>2006-10-05 14:22</timestamp>
    </revision>
    <categories>
      <category>Markup languages</category>
    </categories>
  </header>
  <body>
    <section>
      <st>Introduction</st>
      <p><b>Wiki markup</b> is used in <link xlink:href="../Wi/Wikipedia.xml" xlink:type="simple">Wikipedia</link>.</p>
    </section>
    <section>
      <st>Language Components</st>
      <list>
        <entry>tables</entry>
        <entry>lists</entry>
        <entry>and a lot more</entry>
      </list>
    </section>
    <section>
      <st>See also</st>
    </section>
  </body>
</article>

Figure 2: Generated XML for the Wiki Markup from Figure 1

The XML dialect we use is similar to the one used by the INEX Wikipedia collection. However, we have better support for some details, including nesting of sections (in INEX, all nesting levels of sections are mapped to sec elements). Additionally, as the INEX

7The Wiki Syntax Project (http://en.wikipedia.org/wiki/Wikipedia:WikiProject_Wiki_Syntax) lists 8,400 for the April 21, 2005 dump, including 50 defective section headings, 80 HTML tags, 250 tables, 600 double quotes, 950 triple quotes, and 3400 square brackets.
3 Semantic Annotation of Wikipedia Pages

We aim at finding high-quality semantic annotations for Wikipedia pages by a combination of exploiting manually created, high-quality information from categories that are assigned to most pages, and deriving additional information from highly structured documents such as lists (of persons, locations, etc.). To make the results applicable for a large suite of applications, we find annotations within the scope of a predefined ontology; we use WordNet [Fel98], the currently most extensive and widely used general-purpose thesaurus for the English language, but the results are transferable to any hierarchical ontology. The ontology provides us with a standard vocabulary for the annotations than can also be exploited for querying the annotated documents. Additionally, as our annotation algorithms are heuristic, we explicitly maintain an estimated confidence in the annotations.

3.1 Overview of WordNet

WordNet [Fel98] is a fairly comprehensive common-sense thesaurus carefully handcrafted by cognitive scientists. WordNet distinguishes between words as literally appearing in texts and the actual word senses, the concepts behind words. As of the current version 2.1, WordNet contains 81,426 synsets for 117,097 unique nouns. Often a single word has multiple senses, each of which comes with an estimation of its commonality and a brief description and is also characterized by a set of synonyms, words with the same sense, called synsets in WordNet. In this paper, we use the term concept for word senses, hence each concept corresponds to exactly one synset. WordNet provides relationships between concepts like hypernyms (i.e., broader senses), hyponyms (i.e., more narrow senses), and holonym (i.e., part of) relationships; for this paper, we focus on hypernym relationships.

Conceptually, the hypernym relationship in WordNet spans a directed acyclic graph with a single virtual source node ‘ROOT’ (that we introduced to get a connected graph) and seven first-level basic synsets (entity, state, abstraction, event, act, group, possession) that are children of the source node. Figure 3 shows an excerpt of that graph. For each concept in WordNet, there exists at least one, but usually several distinct root-to-concept paths (like for the concept ‘singer’ in the excerpt).

3.2 Exploiting Categories

The majority of Wikipedia pages is assigned to one or multiple categories. The page about Albert Einstein, for example, is in the categories German language philosophers, Swiss physicists, and 34 more. Not all categories, however, imply that the entity
described on the Wikipedia page is an instance of some concept. Some categories serve administrative purposes (like Articles with unsourced statements), others yield non-conceptual information (like 1879 births) and again others indicate merely thematic vicinity (like Physics). The administrative and non-conceptual categories are very few (less than a dozen) and can be excluded by hand. To distinguish the conceptual categories from the thematic ones, we employ a shallow linguistic parsing of the category name. For example, a name like Naturalized citizens of the United States is broken into a pre-modifier (Naturalized), a head (citizens) and a post-modifier (of the United States). Heuristically, we found that if the head of the category name is a plural word, the category is most likely a conceptual category. We used the Pling-Stemmer [S+06] to reliably identify and stem plural words. This gives us a (possibly empty) set of conceptual categories for each Wikipedia page.

As we want to use WordNet as foundation for our annotations, every conceptual Wikipedia category has to be linked to a corresponding WordNet concept. We experimented with different heuristics, including context-aware [S+05] and compactness-based [M+05] methods, and discovered that the simplest heuristics yields the correct link in the overwhelming majority of cases: We determine the WordNet concepts that the head of the category name refers to and link the category to the most common concept among them.

### 3.3 Exploiting Lists

Wikipedia contains many lists, which are an extensive, manually created and therefore high-quality source of information. In the Wikipedia Snapshot from April 2006 that we used for our experiments, there are 18,436 different lists. We consider the XML versions of the Wikipedia lists, i.e., the output of the XML generation process described in Section 2; this allows a better handling of structure in the lists. As an example, Figure 4 shows
an excerpt of the list of Germans from Wikipedia. To uniquely identify the elements of such a list, we assign a unique XPath expression to each element that consists only of name tests, child axes and position predicates. In the example, the link element that is a child of the second entry element in Figure 4; it is identified by the XPath expression /article[1]/body[1]/section[1]/list[1]/entry[2]/link[1]. Note that Wikipedia lists are not always designed as nested Wiki markup lists; there are many lists that are in fact tables. Our algorithm is not limited to Wiki markup lists and supports any type of regular structure.

It is evident that the example list is well structured, and it would be easy for a human to find out that all links point to pages about actors or, more generally, persons. A manual approach to exploit lists would require that a user identifies patterns in a list (either explicitly by an XPath expression or by highlighting them in an interface) and assigns them to a WordNet concept like ‘actor’. However, while such a manual approach could be useful for small subsets of all lists, this tedious task does not scale to the whole Wikipedia collection.

3.3.1 Automatic Grouping of XPath Expressions

We propose an automated algorithm that exploits the fact that many pages have already been annotated with concepts derived from their categories, and only some pages are left to annotate. The algorithm, a variant of previously proposed algorithms for list extraction like [FFT05], proceeds in three steps: (1) it identifies parts of the list that are structurally similar (so-called group candidates), (2) it selects those group candidates where a large fraction of links points to pages with coherent annotations (so-called groups), and (3) it finds annotations that are common among them, and heuristically assigns these annotations to all pages in the group. In the example, if all but the third link point to pages that are labeled as ‘actor’, the algorithm would assign that concept to the page of ‘Moritz Bleibtreu’ as well.
In a preprocessing step, we first temporarily extend the annotations of all pages that are linked in the list. For each concept annotated to a page, we add all concepts on the root-to-concept paths of this concept in WordNet to the annotations of a page. This allows us to identify different annotations that are similar at a higher level of abstraction in WordNet, like 'actor' and 'singer', which are both subconcepts of 'performer'.

Group candidates are identified by grouping elements with similar XPaths together. We say that two elements have a similar XPath if both paths have the same tag sequence and differ only in a single position. We label each group candidate with an XPath pattern that has the same tag sequence and the same positions as the elements in the group candidate, but a wildcard '*' at the position where the elements differ. As we maintain annotations only for pages (which are identified by link elements in the lists), it is sufficient to consider only group candidates where the last tag is link. We identify each XPath in a group candidate with the page its link element points to. In the example, the group candidate including the three link elements has the label /article[1]/body[1]/section[1]/list[1]/entry[*]/link[1]. We eliminate group candidates that are too small, i.e., consist of less than 5 elements.

To determine if a group candidate with $n$ elements is a proper group, we count, for each concept $c$, the number $f_c$ of times where $c$ occurs as an annotation in the group. We accept the group if there is at least one concept where $f_c/n \geq \delta$, i.e., where the concept occurs in the annotations of at least a (configurable) fraction of all pages. Note that $n$ includes pages without any annotations. Setting $\delta$ close to 1.0 gives a higher annotation quality, but may reduce recall, whereas setting $\delta$ close to 0 incurs a high danger of wrong annotations. In our experiments, $\delta = 0.75$ yielded good results. Each such concept $c$ is then assigned to all pages in the group that are not already annotated with $c$; the confidence of the annotation is set to $f_c/n$.

3.3.2 Outlier Detection

Even though most lists have a regular structure, sometimes outliers occur, i.e., small glitches in an otherwise perfectly regular structure. As an example, consider the excerpt of a list of songs shown in Figure 5. While, for most entries, the first link points to the singer’s page and the second to the song’s, this regularity is broken for one song that has two singers (shown in boldface). If we apply our algorithm in this setting, 'David Bowie’ would be accidentally annotated as 'song'. Similar outliers can be caused by omitting links to pages that do not yet exist in Wikipedia (like some unknown singer) or by mistakes of the page editor. We apply a simple heuristics to detect such outliers that works as follows. First, we manually define the compatibility of a carefully selected set of base concepts towards the top of the WordNet DAG; these concepts are shown in grey in Figure 3. Compatible sets of base concepts are \{living\thing, causal\agent, group\} and \{whole, thing\}, all other combinations of base concepts are incompatible. Whenever a page $p$ should be annotated with a new concept $c$, all base concepts $B(c)$ on paths from $c$ toward the root node are computed and compared with the base concepts that have been assigned to $p$ earlier in the process. The new concept is assigned to $p$ if and only if each base concept in $B(c)$ is compatible with each already assigned base concept of $p$. This heuristic cleaning
eliminates most erroneous annotations.

3.4 Adding Semantic Tags to Pages

Annotations characterize a Wikipedia page and therefore should be stored with the page, enabling queries of the form “find pages of singers that...”. We therefore add tags that correspond to the annotations for a page right after the article element (see Figure 6). The tag names are derived from the WordNet synset that correspond to the annotated concept. The tags are augmented with the confidence, the ID of the WordNet concept, and the source of the annotation. In a NEXI-style query language, the example query fragment would be posed as //singer[about(.,...)], exploiting the new annotation.

At the same time, the annotations of a page are also an important source of information in other pages that link to the page; this can be exploited for queries like “find concerts where the band Queen played”. To support this, we add the same tags also to links to the page in
other pages (see Figure 7). Once we have such an annotation of links, the example query fragment could be formulated as `//concert[about(//band,'Queen')].`, exploiting the fact that any link to the Queen page will be annotated as `band` (as opposed to links to the Queen Mary ship or Queen Elizabeth II of England).

```
<group confidence="1.0" wordnetid="26729" source="categories">
  <artist confidence="0.75" wordnetid="9187509" source="3 lists">
    <link xlink:href="../Qu/Queen+(band).xml" xlink:type="simple">
      Queen
    </link>
  </artist>
</group>
```

Figure 7: Semantically annotated version of a link to the 'Queen' page

4 Exploiting Implicit Semantics of Template Invocations

A rich source of semantics that is already included in Wikipedia are template invocations. However, unlike in the approaches presented in the previous sections, we exploit template invocations not for annotating a Wikipedia page as a whole, but for annotating pieces of information on that page. Templates are often used to generate a standard layout for structured information that is common to many pages. As an example, Figure 8 shows the invocation of the `Infobox_band` template that generates a table with some standard information on a musical band; this information is provided as parameters to the template. Similar templates exist for persons, countries, companies, rivers, software, and many more.

```
{{Infobox_band |
  band_name = Queen |
  image = [[Image:Queen.png|250px|right]] |
  years_active = 1971 - Present |
  status = Active |
  country = [[United Kingdom]]
}}
```

Figure 8: Example call of the `Infobox_band` template

For most templates, the name of a parameter is a clear indication of its semantics. We therefore exploit template invocations in a Wikipedia page to enrich the generated XML document with semantic annotations based on the template parameters. To do so, we try to map each parameter name to a WordNet concept, using the heuristics explained in Section 3.2. We then generate an element with the same name as the template. For each parameter, this element has a child element with the parameter name as name and the current parameter value as value. Figure 9 shows the XML that is generated for the example template invocation from Figure 8.

In contrast to the INEX Wikipedia collection that simply represents the template invocations and their parameters with a generic tag `<template>`, we believe that our approach is much more in the spirit of XML, allowing more natural XPath- and NEXI-style queries
such as //article[about(.,‘band’) and contains(//country,’USA’) and contains(//status,’active’). Note that such queries can also be posed and answered if a user does not exactly know the schema, by relaxing tag names and other structural query conditions [AY+02, S+05, Sch02], or possibly by support of a DTD-aware graphical interface [vZ+06].

5 Applications

5.1 Concept-Based Information retrieval

An important application of annotations is concept-based retrieval. Graupmann et al. [G+05] have shown that annotating important classes of information like persons, locations, and dates can help to improve result quality, especially precision of results. As YAWN annotates with a huge set of diverse concepts from WordNet, it seems likely that these annotations can lead to further enhancements for the retrieval.

We have not yet done a thorough evaluation of the quality of annotations in general and their impact on result quality. However, to give a first impression of the use and effectiveness of annotations, we made some preliminary experiments with our XML search engine TopX [TSW05] on a small, annotated Wikipedia fragment, with NEXI-style [TS04] structured queries. We converted the first 10,000 Wikipedia documents (excluding redirections that contain only a pointer to another document) from the April 2, 2006 dump file into our XML format with semantic annotations, using the techniques presented in the sections before. The conversion failed for 49 documents, usually due to syntax problems in the input Wiki markup or unusual combinations of tables and sections. In our preliminary implementation, this took less than one hour on a standard notebook (the whole collection was converted in about 36 hours on a dual Xeon server machine).

We consider three types of queries: (1) queries that exploit only the annotation of complete
pages, optionally with content constraints, (2) queries that exploit the annotation of pages and links, optionally with content constraints, and (3) queries that additionally exploit annotations derived from template invocations. We collected 5 to 10 queries of each type and compared the performance of TopX with annotation-aware queries to plain keyword queries. Table 1 summarizes the results for the average precision@10 of the three query classes with and without annotation awareness. Note that we did not measure recall; we expect that some results will not be found due to errors in the annotation process. We now discuss some anecdotal results for the three query types.

<table>
<thead>
<tr>
<th>Query type</th>
<th>Precision@10[annotations]</th>
<th>Precision@10[keywords]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (pages)</td>
<td>0.85</td>
<td>0.24</td>
</tr>
<tr>
<td>2 (pages+links)</td>
<td>0.96</td>
<td>0.14</td>
</tr>
<tr>
<td>3 (pages+links+templates)</td>
<td>1.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 1: Experimental results with TopX

A simple example for the first query type are list-style queries like 'find scientists that won the Nobel prize' that would be formulated as keyword query scientists nobel prize without annotation awareness. If we exploit annotations, we can reformulate the query as //scientist[about(.,Nobel prize)], yielding a lot less nonrelevant results. For the example collection, TopX achieves a precision@10 of 0.2 for the keyword query, compared to 0.8 for the annotation-aware query. Here, the additional annotation serves as a prefilter for pages that simply mention the Nobel prize, but are not about scientists (like the page about Jimmy Carter who won the peace Nobel prize).

A typical example that shows the usefulness of the second query type is the query for musicians who have performed a song where 'space' occurs in the title, which could be formulated as //musician[about(//song,space)]. Without the annotation of the link, a search engine would find any occurrence of the term within the page, not only in the context of a song. As a result, the precision of the annotation query is perfect (1.0), whereas the corresponding keyword query finds no relevant results within the top 10.

The last query type is most powerful as it can exploit all three types of annotations. As an example, consider the query 'find mayors of towns in the (German state) Hesse'. While a keyword-based query has no chance to retrieve relevant results, the annotation-aware query //town[about(//state,Hesse)]//mayor that exploits tags introduced by the template Infobox_Town_DE yields only relevant results in the test collection. However, as not all city pages may use such a template, the recall is probably not perfect. To increase recall (at the cost of precision), using ontological tag expansion and structural similarity measures in the engine could help; this is subject of future work.

5.2 Other Applications

There are many more applications for an annotated XML corpus beyond the obvious information retrieval case sketched in the previous subsection, for example to help with
clustering and classification of Wikipedia pages. The annotations can further be exploited for query formulation, by letting a user pick concepts from WordNet and combine them to form a structured query, similar to the DTD-aware query formulation presented in [vZ + 06]. The diverse structure introduced by the annotations will most certainly be a rich source for structural feedback [ST06] that exploits the structure of relevant results to generate more precise queries. And, finally, being the first real-life heterogeneous XML collection with both rich tags and content, we think that our new collection can serve as a large-scale benchmark for systems that exploit semantic annotation for retrieval, classification, clustering, etc, extending existing benchmarks with structural diversity.

6 Conclusions and Outlook

This paper presented YAWN, a project to create an XML version of Wikipedia with semantic information. We showed how to extract semantics from categories, lists, and template invocations, yielding a huge XML corpus annotated with semantically rich tags.

For future work, we plan to extensively evaluate the quality of the annotations and their effect for information retrieval, possibly within the INEX benchmark, and to consider some of the applications sketched in the previous section. Besides that, we will examine how we can integrate the collected information into our ontology and exploit it for other data sources, too. Finally, we plan to offer the annotated XML collection for public download.

References


From Personal Desktops to Personal Dataspaces:
A Report on Building the iMeMex Personal Dataspace Management System*

Jens-Peter Dittrich Lukas Blunschi Markus Färber
Olivier René Girard Shant Kirakos Karakashian Marcos Antonio Vaz Salles
ETH Zurich, Switzerland
dbis.ethz.ch | iMeMex.org

Abstract: We propose a new system that is able to handle the entire Personal Dataspace of a user. A Personal Dataspace includes all data pertaining to a user on all his disks and on remote servers such as network drives, email and web servers. This data is represented by a heterogeneous mix of files, emails, bookmarks, music, pictures, calendar data, personal information streams and so on. State-of-the-art tools such as desktop search engines and desktop operating systems (including the upcoming Vista) are not enough as they neither solve the problem of physical personal information independence (where is my data) nor format and data model independence (how is it stored and which application do I have to use in order to access that data). Our work builds on the visions presented in [DSKB05], which calls for a single system to manage the personal information jungle, and [FHM05], which advocates dataspaces as a new abstraction for information management. In contrast to [FHM05] this paper presents a concrete implementation of a Personal DataSpace Management System (PDSMS) termed iMeMex: integrated memex. We discuss the core architecture of iMeMex and services offered by our system. As we will show, a PDSMS can be seen as a system that occupies the middleground between a search engine, a database management system, and a traditional information integration system. A PDSMS has to bridge these separate worlds and requires: (1) no full control on data, i.e., data may be accessed bypassing the interfaces of a PDSMS, (2) simple keyword search on all data available in a dataspace without performing any semantic data integration, (3) rich querying able to mix structural, attribute, and content predicates, (4) pay-as-you-go integration capabilities, (5) the ability to define arbitrary logical views on all data, (6) durability and consistency guarantees to avoid loss of data assigned to a dataspace, and (7) update capabilities. iMeMex is the first implementation of a PDSMS we are aware of. This paper presents the architecture of iMeMex and reports on the current state of the iMeMex research project at ETH Zurich.

*This work is partially supported by the Swiss National Science Foundation (SNF) under contract 200021-112115.
1 Introduction

Dataspaces have recently been identified as a new agenda for information management [FHM05, HFM06, Mai06, HRO06]. In a nutshell, a dataspaces management system is a new kind of information managing architecture that allows to manage all data pertaining to a certain organization, task, or person. In sharp contrast to existing information integration architectures a dataspaces management system is a data-coexistence approach: it does not require semantic integration investments before services on the data are provided. For example, keyword searches should be supported at any time on all data (schema later or schema never). The existing dataspaces can then be gradually enhanced by defining semantic connections between different components of a dataspaces in a pay-as-you-go fashion [FHM05]. An important use-case of dataspaces are personal dataspaces, i.e., all electronic items that belong to a single person.

This paper aims to give an overview on the iMeMex project which was started two years ago at ETH Zurich and strives to build the first publicly available Personal DataSpace Management System (PDSMS). The current version of our software was shown in a demo at CIDR 2007 [BDG07]. iMeMex is supported by the Swiss National Science Foundation (SNF) and includes a senior researcher, two Ph.D. students, currently three M.Sc. students as well as several short-term project work students (fourteen have completed their work so far). iMeMex is open source since December 15, 2006 (Apache 2.0 License) and a first developer version of the iMeMex server is available from our web-site http://www.imemex.org.

1.1 Background

Personal Information Jungle. In 1945, Bush [Bus45] presented a vision of a personal information management (PIM) system named memex. That vision has deeply influenced several progresses in computing. Part of that vision led to the development of the Personal Computer in the 1980ies. It also led to the development of hypertext and the World Wide Web in the 1990ies. Since then, several projects have attempted to implement other memex-like functionalities [FG96, Be05, CRDS06, KBH05]. In addition, PIM regained interest in the database research community [HED03, DH05a]. Moreover, it was identified as an important topic in the Lowell Report [AAB03], discussed in a VLDB panel [KWF03], and became topic of both SIGMOD 2005 keynotes [Bel05, Mit05]. Furthermore, it was discussed in an NSF-sponsored workshop [JB05] and debated in a SIGIR 2006 PIM workshop [PIM].

Even though considerable progress has been made in the PIM area over the last decades, we argue that a satisfactory solution has still not yet been brought forward to many central issues related to PIM. This is illustrated in the following.

1.2 The iMeMex Vision

PIM today: Assume that Mr. John Average owns a set of devices including a laptop, a desktop, a cellular, and a digital camera. His personal files are spread out among those de-
vices and include music, pictures, pdf files, emails, and office documents. Today, Mr. Average has to copy files from one device to the other, he has to download data to his desktop to see the pictures he shot, he has to upload pictures to sites like flickr.com or picasa.com to share them with his family. He has to make sure to regularly backup data from the different devices in order to be able to retrieve them in case of a device failure. Further, he uses two different modes of searching: a local desktop search engine enabling search on his local devices and a web search engine enabling search on public web-sites. Mr. Average may organize his files by placing them in folder hierarchies. However, the files and data items stored on his different devices are not related to each other.

**iMeMex vision of 2010:** Mr. John Average still owns several nifty devices with growing processing and storage capabilities. Instead of handling ‘devices’ he assigns all his data to a logical *dataspace* named John’s space. Whenever he listens to a piece of music, takes a picture, gets an email, etc., those items are assigned to John’s space. His dataspace management system takes care of the low-level issues including replicating data among devices, enabling search and querying across devices. Whenever John Average wants to share data, he simply creates a subdataspace like John’s space:pictures and selects a list of people who may see that data, e.g., his family or friends. There is no need to ‘upload’ or ‘download’ data: John’s family and friends will just see John’s pictures without requiring to access web-servers or messing around with files. The boundary between the Web and the different operating systems running on his local devices is gone. However, John Average still owns his data: all master copies of his data are physically stored on devices he owns. Searching his dataspace is not restricted to certain devices (like the local desktop), but includes all devices containing data assigned to his dataspace. Other than simple keyword queries, structural queries similar to NEXI [TS04] are enabled. John Average may also search and query the dataspaces of his friends and his family. The search granularity is fine-granular ‘resource views’ [DS06] and not files. Other than just searching or querying, John Average may also use iMeMex to integrate the information available in his dataspace or his friends’ dataspaces in a pay-as-you-go fashion. Therefore, his dataspace management system analyzes the data and proposes relationships among data items. It enhances his dataspace over time and helps to turn a set of unrelated data items into integrated information. Finally, Mr. Average may also update data using iMeMex. However, he may still update his data using any of his applications, bypassing iMeMex.

2 Related Work

Figure 1 displays the design space of existing solutions for information management. The horizontal axis displays requirements for semantic integration, while the vertical axis, in contrast to [FHM05], displays the degree of update guarantees provided by different systems.

On the lower left corner of the design space we find **DBMSs**, which require high semantic integration efforts (upfront investment for schemas), but provide strong update guarantees (ACID). Examples of systems that attempted to apply DBMS technology to personal information include MyLifeBits [Bel05] as well as Rufus [SLS+93]. However, these solutions
incurs high costs for semantic integration and require full control of the data. In sharp
counter to that, a PDSMS does not require schema-first modeling.

In the upper right of Figure 1, strictly opposed to DBMSs, we find Desktop Search Engines (DSE). They do neither require semantic integration, nor full
control of the data. The downside, however, is that a DSE does neither provide any up-
date guarantees nor does it allow to include complex structural constraints, e.g., queries
like //mail/*[from="Jens Dittrich"]). Moreover a DSE does not provide any semantic
information integration capabilities. Examples of DSEs are Google Desktop [GDS], Ap-
ple Spotlight [AMS], Beagle [Bea], and Phlat [CRDS06].

Figure 1 also shows Traditional Information Integration Systems in the middle-left: these systems require high semantic integration investments and vary in terms of their
update guarantees. Several traditional information integration systems have used the rela-
tional model to provide a global, data source independent view of data [LRO96, ACHK94].
The Information Manifold [LRO96], for example, is based on the idea of a global schema
on top of which the data sources may be expressed as relational views (LAV). Semi-
structured approaches, first introduced by TSIMMIS [PGMW95], have been proposed
as an alternative to self-describe data and thus eliminate the need to provide pre-defined
schemas. The integrated view in TSIMMIS is expressed as a set of views on the data
sources (GAV). A more flexible approach, GLAV, is proposed in [FLM99]. The differ-
ences between LAV, GAV and GLAV are further discussed in [Len02]. One deficiency
of classical data integration solutions is the need for high upfront effort to semantically
integrate all source schemas and provide a global mediated schema. Only after this startup
cost, queries may be answered using reformulation algorithms [Hal01]. Quality-driven
information integration [NLF99] enriches query processing in a mediation system by as-
sociating quality criteria to data sources and ranking query plans based on their quality
factors. However, not only schema integration must still be performed, but also col-
lection of quality information for each of the data sources. Peer-to-peer data manage-
ment [TH04, HHL+03, NOTZ03] tries to alleviate this problem by not requiring semantic
integration to be performed against all peer schemas. Nevertheless, the data on a new
peer is only available after schema integration is performed with respect to at least one of
the other peers in the network. In contrast to all of these approaches, dataspace manage-
ment systems demand basic querying on all data from the start, with the possibility to add

Figure 1: Design space of state-of-the-art information manage-
ment systems: PDSMSs fill the space between existing specialized systems.

295
semantic information in a “pay-as-you-go” fashion.

In Figure 1, the corner on the upper left is occupied by **Data Warehouses**: these systems are optimized for read-only access (read-mostly systems). Furthermore, data warehouses require very high semantic integration efforts (integration of multiple schemas). Some PIM systems, such as SEMEX [DH05a] and Haystack [KBH+05], extend data warehouse and information integration technology. They extract information from desktop data sources into a repository and represent that information in a domain model (ontology). The domain model is a high-level mediated schema over the data sources. These systems focus on creating a queryable, however non-updatable, view on the user’s personal information.

The corner on the lower right of Figure 1 is occupied by **Versioning Systems** (e.g., Subversion [Sub], Perforce [Per]). These systems provide strong update guarantees (ACID) but do not perform any semantic integration.

**File systems** occupy the region on the middle-right, providing weaker update guarantees than versioning systems (e.g., recovery on metadata for journaling file systems but no recovery on file content). There are two major approaches to implementing file systems:

1. The first strategy tries to implement file systems using heavyweight DBMS technology. This idea was already proposed and implemented as early as in Exodus [CDRS86] and the Shore [CDF+94] object-oriented storage manager. Shore represented a merger of object-oriented database and file system technologies. It provided a tree-structured, Unix-like namespace in which all persistent objects were reachable from a distinguished root object. The latter approaches influenced many other research projects, however, they never found acceptance in the operating systems community. Recently, Microsoft WinFS [WFS] attempted to continue that tradition by implementing a file system using a relational DBMS. However, that project was dropped in 2006. Another approach striving to implement a file system using a DBMS is [HGS07] by representing file and folder objects as a huge XML document and storing it in a commercial XML database.

2. The second strategy to implement file systems is to code them from scratch using only a few lightweight DB techniques such as B+-trees and simplified recovery techniques (journaling). Today, this strategy is followed by almost all popular file systems including HPFS, XFS, ZFS, EXT3, ReiserFS, and NTFS.

Other approaches that do not try to reimplement current file systems but rather enrich file system using XML technologies are Hubble [LHHB05], and the first version of iMeMex [DSKB05]. In this initial version of iMeMex we used XML and XQuery to join files (e.g., an Excel sheet and a text file) into a new file (e.g., a Word document). The join result appeared as a virtual file on a network share.

In Figure 1 the upcoming operating system Windows Vista is also displayed as it provides some basic information managing capabilities (dotted box on the upper right corner), covering functionalities currently offered by file systems and DSEs. However, as Figure 1 shows, Windows Vista covers only a small fraction of the design space covered by Personal Dataspace Management Systems such as iMeMex.

1 The downloadable beta as well as all other preliminary information about WinFS were recently removed from its web-site [WFS].
In summary, Figure 1 shows that a huge design space between the different extremes (sitting in the corners and along the margins) is not covered by current information management solutions. However, in order to be able to manage the entire dataspace of a user that space has to be covered. PDSMSs fill that space. These systems cover the entire design space of information systems requiring medium to low semantic integration efforts. In particular, PDSMSs occupy the middle-ground between a read-only DSE (drawbacks: no update guarantees, no information integration, neither physical nor logical data independence), a write-optimized DBMS (drawbacks: schema-first, full-control on data required), and a traditional information integration system (drawback: schema-first).

3 iMeMex Core Architecture

In this section, we discuss the core architecture of the iMeMex PDSMS. iMeMex is based on a layered architecture which is described in Section 3.1. Following that, Sections 3.2 and 3.3 discuss important services that are provided by the different layers.

3.1 Logical Layers

The DSSP vision of Franklin et.al. [FHM05] defines a dataspace as a set of participants (or data sources) and relationships among the participants. We term the set of data sources Data Source Layer. Although Franklin et al. [FHM05] present services that should be provided by a DSSP, little is said on how a DSSP would provide those services on top of the Data Source Layer. In fact, in the current state-of-the-art for personal information management, applications (e.g., search&browse, email, Office tools, etc.) access the Data Source Layer (e.g., file systems) directly. This comes at the cost of physical data dependence, including system dependence. This situation is depicted on the left of Figure 2.

To remedy that situation, we argue that what is missing is a logical layer between the applications and the Data Source Layer that provides services on the dataspace. We propose to add the iMeMex PDSMS as that intermediate logical layer. It is depicted on the right of Figure 2. iMeMex abstracts from the underlying subsystems, from data formats, and from devices, providing a coherent view to all applications. iMeMex, however, does not have full control of the data as it is the case with DBMSs. Thus, applications may also access the data sources bypassing iMeMex, e.g., email or office applications do not have to be rewritten to interact with iMeMex: they work directly with the data sources. Other applications, however, may be rewritten to directly operate on iMeMex, e.g., explorer and tcsh.
In the following, we discuss the characteristics of each layer of the iMeMex PDSMS as well as of the layers with which it interacts. All of these layers are shown on the right of Figure 2.

**Data Source Layer.** This layer represents all subsystems managed by the PDSMS. A subsystem that participates on the dataspace may offer either an API that enables full access to the data on that subsystem, access through querying only, or a hybrid of these two options. Thus, the PDSMS must be aware of data vs. query shipping trade-offs [Kos00] to enable efficient query processing.

**Physical Data Independence Layer (PHIL).** This layer is responsible for resolving the data model, access protocol, and format dependence existing on the data sources participating in the dataspace. PHIL offers unified services such as data model integration and indexing and replication. We provide more details on these services in Section 3.2.

**Logical Data Independence Layer (LIL).** This layer provides view definition and query processing capabilities on top of PHIL. LIL offers services such as result caching, view materialization and dataspace navigation for views defined on top of the data unified by PHIL. We discuss important aspects of these services in Section 3.3.

**Application Layer.** This layer represents the applications built on top of the iMeMex PDSMS. As a PDSMS does not obtain full control of the data, applications may choose to either benefit from the services offered by the PDSMS or access the underlying data sources and use specialized APIs. To enable legacy applications to directly interface with the PDSMS, a PDSMS may offer a mechanism for integrating seamlessly into the host operating system, as demonstrated in [DSKB05].

### 3.2 PHIL Services

The primary goal of PHIL is to provide physical data independence. Thus, PHIL unifies data reachable in distinct physical storage devices, access protocols and data formats. We present the main services offered by PHIL below.

**Data Model Integration.** Data model integration refers to the representation of all data available in the data source specific data models using a common model: the iMeMex Data Model (iDM) [DS06]. In a nutshell, iDM models each piece of personal information by fine-grained logical entities. These entities may describe files, structural elements inside files, tuples, data streams, XML, or any other piece of information available on
the data sources. These logical entities are linked together in a graph and logically represent the entire personal dataspace of a given user. The details of our data model are beyond the scope of this paper (please see [DS06]). One aspect of this work is that we favor a clear separation of the logical data model describing the structural properties of data (flat, relational, tree-structured, graph-structured) and the different possible physical data representations (binary, Tables, Object graphs, XML). In the remainder of this paper we use the terms resource view and resource view graph to refer to a logical piece of information (e.g., an email message, a section in a document, a document, an RSS news entry, etc.), and a graph of logical pieces of information, respectively. Please note, that the iMeMex approach is in sharp contrast to semantic integration, in which expensive up-front investments have to be made in schema mapping, in order to make the system useful. We follow a pay-as-you-go philosophy [FHM05], offering basic services on the whole dataspace regardless of how semantically integrated the data is. We are currently developing a powerful framework for pay-as-you-go integration on top of our data model.

**Indexing and Replication.** Given a logical data model to represent all of one’s personal information, the next research challenge is how to support efficient querying of that representation. One may consider a pure mediation approach, in which all queries are decomposed and pushed down to the data sources. Though this strategy may be acceptable for local data sources, it may incur long delays when remote data sources are considered. In order to offer maximum flexibility, PHIL offers a hybrid approach. Our approach is based on a tunable mechanism to bridge warehousing and mediation. For example, we may choose to replicate relationships among resource views that come from remote data sources, but neither index nor replicate their content. In this situation, relationship navigation among resource views can be accelerated by efficient local access to the replica structures, while retrieval of resource view content will incur costly access to (possibly remote) data sources.

### 3.3 LIL Services

The primary goal of LIL is to provide logical data independence. LIL enables posing complex queries on the resource view graph [DS06] offered by PHIL. We discuss the services provided by LIL in the following paragraph.

**Personal Dataspace Search&Query Language.** LIL processes expressions written in a new search&query language for schema-agnostic querying of a resource view graph: the iMeMex Query Language (iQL). In our current implementation, the syntax of iQL is a mix between typical search engine keyword expressions and XPath navigational restrictions. The semantics of our language are, however, different from those of XPath and XQuery. Our language’s goal is to enable querying of a resource view graph that has not necessarily been submitted to semantic integration. Therefore, as in content and structure search languages (e.g. NEXI [TS04]), our goal is to account for impreciseness in query semantics. For example, by default, when an attribute name is specified (e.g. size>10K), we should not require exact matches on the (implicit or explicit) schema for that attribute, but rather return fuzzy, ranked results that best match the specified conditions (e.g. size,
This allows us to define malleable schemas as in [DH05b]. A PDSMS, however, is not restricted to search. Other important features of iQL are the definition of extensible algebraic operations such as joins and grouping (see [DS06]).

**Result Caching.** The caching of query results is used to speed up the computation of views. iMeMex’s approach to query processing is based on lazy evaluation: whenever matching results are present in the Data Source Layer, PHIL, and/or LIL, then these results should be retrieved from the highest of those layers. However, in this scenario, the freshness of the data may be lower at higher levels in the query processing stack. As a consequence, query processing must take QoS concerns (e.g., freshness) into consideration. Our goal is to deliver stale results quickly and then update the result list as fresh data is delivered from the data sources.

**Dataspace Navigation.** Users of information systems typically do not start with a precise query specification, but rather develop one in the course of querying and observing results. We call the process of refining query conditions based on a previous definition of the query dataspace navigation. It is a common pattern in the exploration of personal information but also data warehousing [DKK05]. In general, if any given set of views were previously computed and had their results cached at LIL, the research challenge is to detect whether a new query may be answered using those views [Hal01]. In difference to [Hal01], these techniques have to work on arbitrary content represented as a resource view graph [DS06].

### 4 iMeMex Features

#### 4.1 Current Features

In this section we present current features of our system as of December 2006 (v 0.42.0).

1. **The iMeMex server** is implemented in Java 5 and is platform independent. It currently consists of approximately 50,000 Lines of Code and 530 classes.
2. **iMeMex** is based on a service-oriented architecture as defined by the OSGi framework (similar to Eclipse). This means that services, e.g., data source plugins or content converters, can be exchanged at runtime. Our server may be run with two different OSGi implementations: Equinox [Equ] or Oscar [Osc].
3. **All data is represented using the iMeMex Data Model** [DS06].
4. **Our query parser supports an initial version of iQL.** Our language iQL supports a mix of keyword and structural search expressions.
5. **We provide a rule based query processor** that is able to operate in three different querying modes: warehousing (only local indexes and replicas are queried), mediation (local indexes are ignored, queries are shipped to the data sources), and hybrid (combination of the former methods).
6. **We provide several different indexing strategies implemented on top of a relational Java database (Apache Derby [Der]) and a full-text search engine (Apache Lucene [Luc]),** The relational portions of resource views are vertically decomposed [CK85] [ASX01].
to provide better response times. Our primary target is to develop indexes that operate on external memory. However, some of our index structures are main memory resident. Which indexes to use is fully configurable.

7. Scalability: our current version is able to handle up to 25 GB of indexed data (net size, excluding image or music content) on a single iMeMex instance. The biggest file indexed was 7 GB.

8. We have implemented wrapper plugins for the following data sources:
   1. File systems (platform independent: works for Windows, Linux and MAC OS X)
   2. Network shares (SMB)
   3. Email servers (IMAP)
   4. Databases (JDBC, tested with MySQL and Oracle)
   5. Web documents (RSS/ATOM, i.e., any XML data that is accessible by a URI)

9. We provide content converters for lATEX, Bibtex, XML (SAX-based), and PDF.

10. iMeMex provides two important interfaces:
   1. A text console that allows to perform all administration tasks and also allows to query the server.
   2. An HTTP server supporting three different data delivery modes: HTML, XML, and binary. We are also developing an iMeMex client that accesses the iMeMex server through the HTTP interface. The current version of that client was presented at CIDR 2007 [BDG]07.

11. The iMeMex server is open source (Apache 2.0 License) since December 2006 and a first version of our server can be downloaded from http://www.imemex.org or http://imemex.sourceforge.net.

4.2 Upcoming Features

We are planning to provide the following features with upcoming releases of our software:

1. Pay-as-you-go information integration based on a new declarative framework
2. OS integration for file events (Mac and Windows, using native libraries and C++)
3. Materialized views
4. Cost-based query optimization
5. Integration of updates from data sources
6. Data replication and sharing framework
7. Support for larger datasets > 25 GB, scaling beyond 1 TB using distributed instances.

Please see our web-site for an updated list of supported and upcoming features.
5 Use Cases

In the following, we show two use-cases of our system and how they are supported by the different layers: QoS driven query processing (Section 5.2), and distributed dataspaces (Section 5.3).

5.1 Query Processing Architecture

The design approach we follow in iMeMex is to implement stacked query processors (see Figure 3). The organization of data managing functionality in a processing stack is related to the vision of RISC-style data management components [CW00]. In contrast to [CW00], which focuses on fine-grained data managers for building DBMSs, we advocate a coarse-grained approach that only requires three different query processors. Still, we clearly favor the use of multiple query processors than to use only a single query processor. One reason is that using three query processors provides a clear separation of concerns which in turn facilitates query processing.

5.2 Use Case: QoS Driven Query Answering

Queries posed to iMeMex may be processed by different strategies according to the following QoS parameters:

**Exact vs. Relaxed Matches:** queries may request the system to produce either exact or relaxed matches. When relaxed matches are requested, then queries may be decomposed into many relaxed sub-queries (as, for example, in [AY04]). The results from these sub-queries will be merged and ranked by the query processors according to the relevance to the query.

**Freshness vs. Response Time:** queries may request fresh/high cost results, low cost/stale results or best-effort results (i.e. low cost or stale results first, followed by fresh or high cost results). For instance, a stale version of a resource view may be quickly returned from a local cache, an up-to-date version of that same resource view may be returned from a remote data source only after some time.

Depending on the QoS parameters for a given query, each query processor assembles a plan that accesses data on the same layer and/or splits and ships a query to the next query processor in the stack. To illustrate that concept, consider the following query which requests all papers in the dataspace au-
thored by Mike Franklin:

**Q1:** \(\text{mimetype}=(\text{PDF or PS}) \text{ and } \text{author}\sim "\text{Mike Franklin}"\)

In this example, our personal dataspace includes files in the local file system and emails on the IMAP server. Assume that query

**Q2:** \(\text{mimetype}=(\text{PDF or DOC})\)

was submitted to the system 10 minutes before Q1 and, consequently, its result is cached at LIL. For simplicity, consider that PHIL was not updated since query Q2. Figure 4 shows the query plan produced by stacked query processors with relaxed matches and best-effort QoS specifications. The plan on Figure 4 contains three sub-plans, each of them assembled by a different query processor in the stack.

QP\(_{iQL}\) starts the planning process and detects that a subset of Q1’s results is cached at LIL. This is a consequence of the previous execution of Q2. Since QP\(_{iQL}\) has the result for PDF mime types cached, QP\(_{iQL}\) rewrites Q1 into two sub-queries:

**Q1.1:** \(\text{mimetype}=(\text{PDF}) \text{ and } \text{author}\sim "\text{Mike Franklin}"\)

**Q1.2:** ([mimetype=PS and author\sim "Mike Franklin"]

\text{ or } ([\text{mimetype}=(\text{PDF}) \text{ and } \text{author}\sim "\text{Mike Franklin}"

\text{ and modified > now - 10min})]

The sub-query Q1.1 is planned by QP\(_{iQL}\) and its plan is depicted in Figure 4 in the left region. QP\(_{iQL}\) plans to scan the cache, filter all resource views that have mime type PDF, and then to filter&rank all resource views having either associated tuples with name \textit{author} and value “Mike Franklin” or content matching “Mike Franklin”. In addition, QP\(_{iQL}\) plans to union&rank the results from Q1.1’s sub-plan with the results of Q1.2. Note that Q1.2, which requests data modified within the last 10 min, is not planned by QP\(_{iQL}\), but rather shipped to the next query processor in the stack, namely, QP\(_{iDM}\).

QP\(_{iDM}\) verifies that, since the indexes and replicas at PHIL were not updated for the last 10 minutes, the up-to-date data has to be requested from the next query processor in the stack. We assume that QP\(_{iDM}\) was configured to represent, in its indexes and replicas, all...
data in the data sources and therefore may partially answer Q1.2. Thus, QP\textsubscript{IDM} splits query Q1.2 into two sub-queries:

Q1.2.1: \texttt{mimetype=PS and author\sim"Mike Franklin"}
Q1.2.2: \texttt{mimetype=(PDF or PS) and author\sim"Mike Franklin"

and \texttt{modified > now - 10min}}

The sub-query Q1.2.1 is planned by QP\textsubscript{IDM} and accesses the indexes and replicas in PHIL. The plan for sub-query Q1.2.1 is shown in the middle of Figure 4. It accesses the text indexes for matches of “Mike Franklin” and accesses the tuple index for matches of the tuple \texttt{author="Mike Franklin"}. Then it unions and ranks the results from the two index access operators. Afterwards, the plan intersects and ranks the results from the union with the results obtained by accessing the tuple index for matches of the condition \texttt{mimetype=PS}. The final operation in the plan is to apply union and ranking to the results obtained from the intersection with the results of sub-query Q1.2.2. Note that sub-query Q1.2.2 is shipped to the next query processor, QP\textsubscript{DS}.

QP\textsubscript{DS} is the bottom query processor in iMeMex’s query processor stack. The plan it assembles for Q1.2.2 is depicted on the right of Figure 4. The leaves of the query execution plan constructed by QP\textsubscript{DS} are queries to the data sources. These queries are expressed in the languages specific to the data sources’ query interfaces. The leaf on one branch of the plan scans the file system for all PDF or PS files modified in the last 10 minutes. The leaf of the other branch of that plan submits a query to the IMAP server requesting all mails that were received in the last 10 minutes. The results from the IMAP server then pass through a filter that keeps all attachments that are PDF or PS. As Figure 5 shows, the results on both branches are filtered and ranked according to the condition \texttt{author="Mike Franklin"} and content matching "Mike Franklin", before feeding them to a union operator that unites the two branches and also ranks the results, before passing them to QP\textsubscript{IDM}. Each ranking operator ranks according to specific criteria.

An alternative plan would be to retrieve all data from the IMAP server and the local file system that was modified within the last 10 minutes and use that data to simultaneously update the indexes and replicas of the PHIL.
Execution Time vs. Result quality

Traditional query processors perform cost-based query optimization by estimating the cost of a certain plan. Cost typically only includes an estimate on the execution time. In $iMeMex$ we plan to take a different approach: we want to assess plans using a trade-off function that combines the estimated execution cost with the estimated query result quality. A trade-off function scores a plan and looks as follows:

$$\text{Score(Plan)} = \frac{1}{a+b} \left[ a \times \left(1 - \frac{\text{estimated execution time}}{\text{max execution time}}\right) + b \times \frac{\# \text{ results}}{\text{max number results}}\right].$$

In this function $a$ and $b$ are weights that determine how much an accurate result has to be favored over a quick result. Figure 5 shows an experiment using a trade-off cost function for three different query modes ($a = 3$, $b = 4$). The horizontal axis depicts the percentage of updates occurring on the data sources. The vertical axis depicts the score as computed by the trade-off function. We executed the query in three different planning modes: (1) DWH: data warehousing mode (only local stale indexes are used by $iMeMex$, no updates on data sources are considered); (2) MED (local indexes are ignored by $iMeMex$, the query is fully shipped to the data sources); and (3) HYB (local indexes are exploited by $iMeMex$ but merged with updates from data sources). Figure 5 shows that for very low update rates a DWH-like plan gives the best results w.r.t. the scoring function, for higher update rates (up to 50% in the figure) hybrid query plans attain the highest scores. Pure mediation plans achieve only a constant low score for all update rates as they require costly query processing on the source systems.

5.3 Use Case: Distributed Dataspaces

This section briefly illustrates the distributed query processing capabilities of $iMeMex$. All data sources that are part of a dataspace managed by $iMeMex$ may be distributed among several machines. For instance, one $iMeMex$ instance may manage $N$ data sources. However, an $iMeMex$ instance, say $B$, may also play the role of a data source. For instance, an $iMeMex$ instance $A$ may observe $iMeMex$ instance $B$ as a data source. Then, when a query is received by $A$, that query has to be shipped to $B$ which in turn ships it to
its data sources. In distributed networks of iMeMex instances, query planning has to be aware of possible overlap of query results, e.g., two iMeMex data sources may deliver the same results. Further, infinite query shipping loops have to be avoided, i.e., two iMeMex instances may register each other as a data source.

We document the current capabilities of our system by showing a scalability experiment. Figure 6 shows an experiment evaluating the scaleup of a distributed dataspace scenario. In this experiment we installed ten iMeMex instances \(I_1, \ldots, I_{10}\) on ten different machines. Those ten iMeMex instances were registered as data sources to another iMeMex instance \(I_{11}\). Instances \(I_1, \ldots, I_{10}\) contained identical data sets. The figure displays the query execution time as a function of the number of iMeMex instances appearing in the dataspace. The execution time was measured at \(I_{11}\). Due to space constraints we display two representative queries: query Q1 is a keyword query while Q2 is a path query. The results show that iMeMex scales linearly with the number of iMeMex instances.

6 Conclusions

This paper has advocated the design of a single system to master the personal information jungle [DSKB05]. We have proposed Personal Dataspace Management Systems (PDSMS) as a unified solution to manage the entire personal dataspace. This paper reported the current state of the iMeMex project. Our system iMeMex introduces a logical layer on top of the data sources that provides full physical and logical personal information independence. Moreover, iMeMex provides seamless transition between warehousing and information integration. Our PDSMS is based on stacked query processors. We have discussed some of the advantages and challenges of implementing this type of architecture using two use cases. iMeMex is open source (Apache 2.0 License) since December 2006 and an initial version (v0.42.0) of our server can be downloaded from http://www.imemex.org or http://imemex.sourceforge.net. As part of future work we will improve the existing features and provide new features as presented in Section 4.

Acknowledgements First of all, we would like to thank the anonymous referees for their helpful comments. Furthermore, we would like to thank all M.Sc. students who did their semester project in the iMeMex project (currently about fourteen). All of them contributed to the project and shaped it to its current state.

References


Melting Pot XML
Bringing File Systems and Databases One Step Closer

Alexander Holupirek, Christian Grün, Marc H. Scholl
Databases and Information Systems Group
University of Konstanz
<firstname>.<lastname>@uni-konstanz.de

Abstract: Ever-growing data volumes demand for storage systems beyond current file systems abilities, particularly, a powerful querying capability. With the rise of XML, the database community has been challenged by semi-structured data processing, enhancing their field of activity. Since file systems are structured hierarchically they can be mapped to XML and as such stored in and queried by an XML-aware database. We provide an evaluation of a state-of-the-art XML-aware database implementing a file system.

1 Introduction

We generally face the fact that the amount of data stored in file systems on personal computers is steadily growing. This comes as no real surprise since—against current opinion—data gets copied from old machines to new ones instead of being curated, archived and purged from the working system. This may be considered a bad habit, but it surely is a side effect of storage capabilities increasing at low cost, and thus cannot be condemned. Jim Gray et al. pointed out that a “decade ago, 100 GB was considered a huge database. Today it is about 1/2 of a disk drive and is quite manageable. […] so it is both economical and desirable to bring the old data forward and store it on newer technology.” [GST+ 02]. Hence file systems contain a significant amount of text documents, images, and multimedia files. While the mere storage is an easy-to-manage task, convenient access to and information retrieval from huge amounts of data is crucial to leverage the stored information. Current file systems and their proven, but basic interface (VFS) support neither.

Challenge. Donald Norman coined the phrase “Attractive things work better” [Nor04]. While Norman’s statement in the first place aims at pushing aesthetics and attractiveness into user interfaces, it suits well for any human-centered design approach. Without usability, joy of use cannot evolve. Ease of use, on the other hand, is crucial and for a data storage system it is determined by the ability to search/find and access/use stored data. In fact, the challenge we face now (and will have to even more in the future) is to enhance storage systems in a way that users can make full use of their data. Finding relevant content in this ever growing amount of data is a major hassle. File systems still focus on mere storage and tend to be conservative regarding feature enhancements [ZN00]. Consequently, they do not offer solutions to this demanding task. A user’s demand, however, can be derived
from the popularity of industrial products such as Apple’s Spotlight or Google’s Desktop Search. As these products make intensive use of database technology and since important features (such as index structures or part of transaction management: journaling/recovery) have already been ported from databases to modern file systems, it comes as no surprise that leading researchers, like Jim Gray—speaking of a “file system/database détente” at USENIX FAST 2005 [Gra05]—see both worlds colliding. In this paper we will evaluate the potential of a state-of-the-art XML-aware database in the field of file system processing.

Outline. We start with a discussion of related work. In Section 3 we will represent information contained in files and file systems in XML. As such it can be stored in and retrieved by an XML-aware database. To actually operate on the file system representation, a mapping of commonly used file system operations to XPath/XQuery (Section 4) is proposed. We have chosen X-Hive/DB as an evaluation candidate and will report about its performance in Section 5. Since we found that it yields promising results we will discuss our next steps towards merging file system and database technology and finally conclude.

2 Related Work

Various ideas have been proposed for including file contents into information systems. One of the earliest attempts, the Semantic File System [GJSO91], extracted attribute-value pairs for specific file types via so-called transducers. Content queries could be formulated by entering directory paths and extending them with AND combined query terms. The result was a virtual path, resembling a default directory path and including symbolic links to the result documents. While SFS offered only limited retrieval functionality and ways of representing the query results, it has influenced numerous future file system projects, including Shore [CDF+94], HAC [GM99] or the recently discarded WinFS from Microsoft.

An interesting approach to bring XML and file systems together was presented by IBM’s XMLFS [AFMM02]. The underlying prototype implementation offered access to XML documents via an NFS server, and a simple path language allowed querying tags and text nodes across several documents. Nevertheless, the project was not extended to full XPath/XQuery support, and document storage was apparently limited to XML instances.

IBM’s Virtual XML Garden [RMS06] and the draft of File System XML (FSX) [Wil06] share the common idea to have a unified view over heterogenous data sources. Since file systems are structured hierarchically they can easily be mapped to an XML structure as sketched in [Wil06]. Together with the idea to let the file system immerse into the file [Lau98], i.e., the internal structure of a file is no longer a black box to the system, but is integrated in the generic view and can thus be used to navigate into the file itself, these provide the basis for the construction of our test data.

iMeMex [Dit06] is, according to the authors, the first Personal DataSpace Management System (PDSMS). It aims at providing a software platform to facilitate a heterogeneous and distributed mix of personal information. Since “XML is not enough” [Dit06] to model “the total of all personal information pertaining to a certain person” [Dit06], the creation
of an own iMeMex Data Model [DS06] is proposed. It copes with various data sources and also the information contained in files and file systems can be expressed in a unified way. Furthermore, a new search and query language is proposed. A publicly available version of the iMeMex system was rescheduled for December 2006.

**Position.** Our long-term research project is focused on the question to what extent we can use semi-structured database techniques to implement file systems and enhance them with a standardized and widely accepted query interface. As such we represent both data and metadata in the lingua franca of the Internet, i.e., XML, and deal with it using its related tools XQuery/XPath.

## 3 Mapping a File System to XML

**Naïve mapping of a directory structure.** A straight-forward approach to map a simple exemplary directory structure to XML is depicted in Figure 1. While this representation is simple, it keeps the proven hierarchical structure alive.

Directories are either empty or recursively consist of other directories and/or files. All defined types and declared elements belong to a target namespace—prefix `fs`—to identify them as file system entities. Expressed in XML Schema, the definition thus yields:

```xml
<complexType name="directoryType">
  <sequence>
    <element name="directory" type="fs:directoryType" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
    <element name="file" type="fs:fileType" minOccurs="0" maxOccurs="unbounded" nillable="true"/>
  </sequence>
</complexType>
```

We intentionally ignore the UNIX principle that “Everything is a file”, but follow our paradigm to make information explicit. Thus we distinguish between named pipes, block and character devices, sockets as well as hard and symbolic links. As a consequence “Ev-
everything is an element” in our mapping. Strictly speaking, the term directory tree is deceptive when links are involved. The more precise representation of a directory hierarchy would be a directed graph.

**Directory hierarchy as a directed graph.** Enhancing the exemplary directory hierarchy with some hard and symbolic links leads to Figure 2. Symbolic links may point to files (devices etc.) or directories, build cycles, or may even be broken, i.e., point to non-existing file system entries. At least three ways exist to model these conditions in XML, either by using the XLink/XPointer Language [W3C01], [W3C02], by using XML 1.0 ID, IDREF, IDREFS attributes [W3C06a] or by enforcing a constraint using the key and keyref elements provided by XML Schema. We will omit the details here.

In general the file system mappings are built in a depth-first preorder tree traversal, starting from the topmost directory. Whenever a symbolic link is encountered, its target is resolved and referenced. Broken/dangling symbolic links are labeled with attribute @fs:valid="false". Additionally, the attribute @fs:readlink stores the output of the homonymous UNIX system command.

**Inclusion of content.** A central issue of the mapping is the inclusion of textual contents into the XML representation to allow the full range of XQuery retrieval features, defined by the emerging XQuery Fulltext specification [W3C06c]. The mapping process itself is pretty straightforward: all content is enclosed in <content> and resides in a mime-type specific namespace, e.g. <txt:content>.

We also expect the inclusion of additive structural information to be performed for any
meaningful file type, and we demonstrate the idea by applying it to musical metadata as discussed in [Pac05]. The following describes a music title using a simplified MPEG-7 markup (a “real life”, and thus rather lengthy, mapping of ID3 information is defined by MPEG-7 [ISO04]):

```
<file fs:name="Contrapunctus 9 a 4 alla Duodecima.mp3" ...>
  <file:fs:suffix="mp3" fs:type="audio/mpeg">
    <mp3:content mp3:track="9/11" mp3:version="id3v2: simplified">
      <mp3:title>Contrapunctus 9 a 4 alla Duodecima</mp3:title>
      <mp3:albumtitle>Die Kunst der Fuge</mp3:albumtitle>
      <mp3:comment>BWV 182</mp3:comment>
      <mp3:creator>
        <mp3:role mp3:type="artist">
          <mp3:name>Robert Hill</mp3:name>
        </mp3:role>
        <mp3:role mp3:type="composer">
          <mp3:name>Johann Sebastian Bach</mp3:name>
        </mp3:role>
      </mp3:creator>
      <mp3:recordingyear>1970</mp3:recordingyear>
      <mp3:genre>Classical</mp3:genre>
    </mp3:content>
  </file:fs:suffix>
</file>
```

To summarize this section: Although the proposed mapping is straightforward it (a) fulfills our paradigm to externalize formerly hidden information and (b) keeps the well-known and proven directory hierarchy alive. By stepwise porting a UNIX file system to XML, we can observe that mapping is possible without loss of information. Quite the contrary, the leverage of tacit information, formerly encapsulated in various formats, leads to a standardized and easily accessible representation. This provides a basis to operate on file system data with database technology, and we are now able to query the data itself and to apply information retrieval techniques, such as fuzzy information retrieval or relaxed structural queries.

4 Basic Operations on File Systems using XQuery et al.

To actually work on the proposed mapping as a user would work on a file system, basic operations can be expressed in XPath, XQuery, and XQuery Update. The following chapter will introduce selected, frequently used UNIX commands (that operate on file systems) and their translation to the XML domain. Together with the previously proposed mapping, they provide a basis for the evaluation in Chapter 5. As in the previous chapter we will stick with UNIX-based operating systems and their commands but claim that other environments have similar operations.

Path names and path expressions. Navigation in file systems and navigation in XML documents have quite a lot in common: paths play an important role. For XML, path expressions are the core construct of XPath; they represent a fundamental part of XQuery. For file systems, path names are—since their introduction in the PDP-11 system—the natural way to address files. In both worlds paths consist of a sequence of steps, syntactically
Traversing the directory hierarchy. Most basic operations, such as changing the directory (chdir/cd) or listing the directory contents (ls), are completely based on path names. In the following, UNIX commands such as cd $fs$ (operating on the file system) are in the following mapped to xqfs:cd(\(\rho\)) (operating on the XML document).

Navigate to root. Because we allow several file system instances to be stored in one XML document, xqfs:fsRoot(\(\cdot\)) allows to select and determine a root directory. Its behavior is comparable to XQuery’s fn:root() function, but returns the topmost directory node of the file system instead of the root node in the same tree. If the argument contains a file system identifier of type anyURI or any file system node, the according root directory node is returned.

XQuery function xqfs:cd(\(\cdot\)), in its different flavors, maps the functionality of the homonymous operating system command. Traversing the directory hierarchy basically means applying XPath expressions on element(dir) nodes. It is thus sufficient to restrict the operations to element(dir) nodes. This is reflected by the signatures of the implementing functions. Their return value and their effective arguments are of type element(dir).

If a file system path name $\rho_{fs}$ is accepted as string argument it is instantly converted to its element(dir) counterpart. Therefore the xqfs:cd(\(\cdot\)) function always evaluates path expressions encapsulated in functions. Sometimes path names are converted to path expressions as a preprocessing step.

xqfs:cd(\(\cdot\), $\rho_{fs}$) switches the context to path name $\rho_{fs}$. A path name is expected in file system-like notation as string and converted to an XPath expression, according to Table 1. If $\rho_{fs}$ is an absolute path, the current context node ‘.’ is used to find the file system root.

---

Table 1: File system path names to XPath/XQuery path expressions

<table>
<thead>
<tr>
<th>path names</th>
<th>path expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>self::fs:dir</td>
</tr>
<tr>
<td>..</td>
<td>parent::fs:dir</td>
</tr>
<tr>
<td>$\delta_0$/.../$\delta_n$</td>
<td>child::fs:dir[@fs:name=&quot;$\delta_0$&quot;]/.../child::fs:dir[@fs:name=&quot;$\delta_n$&quot;]</td>
</tr>
<tr>
<td>/...</td>
<td>xqfs:fsRoot()/...</td>
</tr>
<tr>
<td>.../$f$</td>
<td>.../child::fs::*[@fs:name=&quot;$f$&quot;]</td>
</tr>
</tbody>
</table>

separated by a slash (‘/’): $s_0/s_1/.../s_n$. Each step $s_1...s_n$ operates on the result of its previous step $s_{i-1}$. Depending on the type of the path (absolute or relative), the origin for the first step $s_0$ differs. For absolute paths it is the topmost directory and the topmost node of an XML document, respectively. In the relative case it is the current working directory and the current context sequence (cs). Absolute path names are notated with a leading ‘/’. A special marker for relative path names may be omitted. However, a relative path name $\delta_0$/.../$\delta_n$/f with directory names ($\delta_i$) and a device/socket/file ($f$) is equivalent to ./$\delta_0$/.../$\delta_n$/f, where ‘.’ denotes the current working directory. Given the proposed mapping, file system path names ($\rho_{fs}$) translate to path expressions ($\rho_{xq}$) as shown in Table 1.
Simplified, the context sequence is a set of items returned by a previous evaluation, i.e., the result of an expression. In XQuery part of the dynamic evaluation context is called the focus, consisting of three items: the context item (‘.’), i.e., the item currently being processed, the context position, and the context size [W3C06b]. Thus, for an expression (e) that operates on a given context sequence cs/e, expression e will be evaluated with ‘.’ (the current context item) set to each item in the context sequence.

In a nutshell. Since path names are a simple sequence of child and parent steps in a hierarchy, they are ’naturally’ supported by XPath. Constructions like the definition of default prefix paths, stored in a user’s environment by the operating system, are inherently given in XPath. Basically, they provide the parallel lookup of directories. Moreover, a sequence of qualifying directories may be selected in XPath by a single expression and used as input for the next step. In contrast to a static string, containing, e.g., directories with executable files, the relevant directories may be returned and dynamically adopted by a single XQuery expression. Thus, for XPath it is quite simple to actually switch to multiple directories in one step and list all bin directories in the file system:

```
> doc("mappedfs.xml")/descendant-or-self::fs:dir[@fs:name="bin"]/xqfs:ls(.)
```

With XQuery, the result set may easily be filtered, according to access control lists, file permissions and the like.

Conversion from path names to path expressions and vice versa. Since conversions between path names (pn) and path expressions (pe) provide a bridge between both worlds, two dedicated functions—xqfs:pn2pe(·) and xqfs:pe2pn(·)—deal with this task. xqfs:pn2pe(·) expects a UNIX file system path name as string and converts it to an equivalent XPath expression as seen in Table 1. xqfs:pe2pn(·) returns the preserved UNIX file system path name for arbitrary nodes in the mapping.

```
declare function xqfs:pe2pn($f as element()+) as xs:string* {
    for $e in $f
        return fn:string-join(
            for $v in $e/ancestor-or-self::*
                return if ($v/@fs:root) then '' else fn:data($v/@fs:name)
        , '/')
};
```

The function is equivalent to the UNIX command `pwd`. Passing the current context item will exactly behave as expected:
Most functions make direct or indirect use of these conversion methods. This is pretty obvious as the functions provide an interface for legacy applications. Path names are passed in a familiar manner as input to the XML data store, and the result is returned in the same way. For instance, the search for a music album containing the word ‘Friede’ returns a set of path names which can then be processed by any available application.

In the following, we will briefly look into a few more commands as they are used in the evaluation in Section 5. We will take a look on how the data store may be modified and how existing content can be searched.

**Modify and search the data store.** Of course, it is essential to add new, modify existing and remove obsolete content in a storage system. A bunch of UNIX commands is dealing with such issues, e.g., rm, rmdir, touch, mkdir, and, consequently, all other applications that modify existing content. Most commands offer several options to specify exactly how they are supposed to operate. Some of them extend the area of operation, such as flag \(-r\), that usually instructs the command to operate recursively on descendant entries. Commands such as \([\text{touch} \rho_f, \ldots]\) or \([\text{rm} -rf \rho_f, \ldots]\) are relatively simple to express. It is sufficient to resolve the targets specified by the list of path names and modify/remove the entries. However, such operations have side effects, and XQuery, as a declarative language, has no means for it. At the time of writing, update functions—such as remove, insert, replace, and rename—are not yet part of the XQuery 1.0 Recommendation. The first drafts of the XQuery Update Facility [W3C06d] have been published. Currently, state-of-the-art databases support updates through either proprietary extension of XQuery or implementations of an older XUpdate draft [LM00] which is not maintained since 2001. For the implementation we therefore use functions in the xhive namespace, that do data modifications as a side effect and return an empty sequence.

Commands such as \([\text{rmdir} [-p] \rho_f, \ldots]\) need some preprocessing as they impose constraints on their targets. For instance, a directory will only be deleted if it is empty. Option \(-p\) treats each argument as a path name of which all empty components will be removed, starting with the last component (man \(1p\ rmdir\)). A recursive function would be a straight-forward approach to implement such behavior in XQuery. The approach we choose later in our evaluation is to follow the path bottom up and to check each node to just contain a single (empty) directory. By such, a single (the topmost qualifying) directory node is returned which is removed together with its (empty) directory descendants.
Summary. In this section we provided some examples for the implementation of commonly used UNIX commands by XPath/XQuery operations. All discussed (and some more) functions are combined in an XQuery library module (fsops.xql). Shortly summarized, most operations resolve path names to path expressions as a first step. The resulting context sequence is mostly narrowed down to elements of a specific type. According to that type, it is declaratively described how to list/remove/sort, i.e., process it. Because of the declarative nature of XQuery, most implementations of UNIX commands are of great simplicity. This may overstate the issue, but the representation of data in XML allows for declarative programming in a formerly purely imperative environment. While this is convenient to implement the upcoming ad-hoc evaluation, we will investigate if it is still feasible to work interactively with the proposed data store.

5 Evaluation of File System Mappings on X-Hive/DB

Testing our approach on a general-purpose XML-aware database, the upcoming evaluation is focused on the question: Can we work interactively on XML documents representing file systems and their contents? Basic behavior patterns that occur when working with file systems are simulated. Since the navigation along the directory hierarchy is a crucial task for various commands, it is evaluated in the first place. The second experiment aims at modifying the data storage, therefore entries are added to and removed from the storage. The last test is focused on search and retrieval functionality.

Description of the test environment. All tests were performed on a 64-bit system with a 2.2 GHz Opteron processor, 16 GB RAM and SuSE Linux Professional 10.0 with kernel version 2.6.13-15.8-smp as operating system. Two separate discs were used in our setup (each formatted with ext2). The first contains the system data (OS and the database), the second the input data and the internal representations of the shredded documents. The query results are written to the second disc. X-Hive/DB 7.3.1 [XH06] is chosen as test candidate for two reasons: (a) It has proven to be one of the best available, commercial solutions for persistent XML processing [BGvK’06], (b) it is the most complete system presently available. It provides element, value, path, and even fulltext indices as well as update functionality. A JAVA-based benchmarking framework PERFIDIX, developed within our research project [GHK’06], is used to facilitate a consistent evaluation of all tests. The framework was initially inspired by the unit testing tool JUNIT [GB06]; it allows to repeatedly measure execution times and other events of interest. The results are aggregated, and average, minimum, maximum, and confidence intervals are collected for each run of the benchmark. PERFIDIX executes a specified query for a dedicated number of times, i.e., number of runs (#runs) in the following tables. Each run is divided into three steps: Step one creates a new session, connects to the database server and registers the session. Step two triggers the execution of the query (incl. commit). The execution time of this step is actually measured. Step three disconnects from the database server and terminates the session. The overall procedure for each benchmark is a) start database server, b) load library module fsops.xql, c) execute benchmark (the three steps described above) n-times and finally d) stop database server.

317
Table 2: Path queries for test scenario I with number of path steps

<table>
<thead>
<tr>
<th>no</th>
<th>query</th>
<th>steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td><code>doc('mappedfs.struct.xml')/xqfs:cd(., './home/holu')</code></td>
<td>2</td>
</tr>
<tr>
<td>q2</td>
<td><code>doc('mappedfs.xml')/xqfs:cd(., './home/holu')</code></td>
<td>2</td>
</tr>
<tr>
<td>q3</td>
<td><code>doc('phobos04.xml')/xqfs:cd(., './home/holupire')</code></td>
<td>2</td>
</tr>
<tr>
<td>q4</td>
<td><code>doc('mappedfs.struct.xml')/xqfs:cd(., './usr/share/doc/rfc/.../tar')</code></td>
<td>8</td>
</tr>
<tr>
<td>q5</td>
<td><code>doc('mappedfs.xml')/xqfs:cd(., './usr/share/doc/rfc/.../tar')</code></td>
<td>8</td>
</tr>
<tr>
<td>q6</td>
<td><code>doc('phobos04.xml')/xqfs:cd(., './home/cebron/.../unikn/knie/')</code></td>
<td>8</td>
</tr>
<tr>
<td>q7</td>
<td><code>doc('phobos04.xml')/xqfs:cd(., './home/cebron/.../tmp/props/')</code></td>
<td>19</td>
</tr>
</tbody>
</table>

Table 3: Path queries on file system mappings (without index in sec.)

<table>
<thead>
<tr>
<th>no</th>
<th>docsize</th>
<th>min</th>
<th>max</th>
<th>avg</th>
<th>stddev</th>
<th>#runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>7M</td>
<td>0.008</td>
<td>0.139</td>
<td>0.014</td>
<td>0.010</td>
<td>1000</td>
</tr>
<tr>
<td>q2</td>
<td>230M</td>
<td>0.009</td>
<td>0.189</td>
<td>0.014</td>
<td>0.012</td>
<td>1000</td>
</tr>
<tr>
<td>q3</td>
<td>8600M</td>
<td>0.009</td>
<td>0.185</td>
<td>0.016</td>
<td>0.013</td>
<td>1000</td>
</tr>
<tr>
<td>q4</td>
<td>7M</td>
<td>0.024</td>
<td>0.292</td>
<td>0.031</td>
<td>0.015</td>
<td>1000</td>
</tr>
<tr>
<td>q5</td>
<td>230M</td>
<td>0.034</td>
<td>0.264</td>
<td>0.041</td>
<td>0.015</td>
<td>1000</td>
</tr>
<tr>
<td>q6</td>
<td>8600M</td>
<td>0.011</td>
<td>0.340</td>
<td>0.017</td>
<td>0.016</td>
<td>1000</td>
</tr>
<tr>
<td>q7</td>
<td>8600M</td>
<td>0.014</td>
<td>0.261</td>
<td>0.022</td>
<td>0.018</td>
<td>1000</td>
</tr>
</tbody>
</table>

Query scenario I: The directory hierarchy

Task description. Navigating the directory hierarchy is essential for almost all file system tasks. Queries 1–7 perform a simple traversal down the directory hierarchy, which is done by the `cd` command. This includes the translation from path names to path expression and the evaluation of the latter. The mappings reveal a maximum depth of 8 for the `mappedfs` documents and a maximum depth of 19 for the research server (`phobos04.xml`). For each mapping, a path of length 2 and a path of length 8 is evaluated. For the third mapping an additional descent to the deepest directory is performed. The serialization, i.e., the output of the resulting nodes, is not measured since only the costs of the traversal are relevant. The test is carried out twice: Phase one operates on a vanilla database instance, phase two uses a value index on `fs:dir/@fs:name`. Queries and results are combined in Tables 2–4.

Result interpretation. At first we observe that the average runtime is always close to the minimum. This can be derived from the benchmark’s layout. The internal log of PERFIDIX revealed that the first run is always the slowest. Considering the fact that the database server is shut down only between the different queries and not between each run, one can make an educated guess that this is due to cache influence. As the operating system also caches, a hot cache evaluation is nevertheless appropriate. Independent from cache influence, the first important result is that the amount of stored content does not interfere with a high-performance path traversal. Execution time for path navigation are in the same range.
Table 4: Path queries with value index on \texttt{fs:dir}@\texttt{fs:name} (in sec.)

<table>
<thead>
<tr>
<th>no</th>
<th>docsize</th>
<th>min</th>
<th>max</th>
<th>avg</th>
<th>stddev</th>
<th>#runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>7M</td>
<td>0.009</td>
<td>0.100</td>
<td>0.014</td>
<td>0.010</td>
<td>1000</td>
</tr>
<tr>
<td>q2</td>
<td>230M</td>
<td>0.009</td>
<td>0.175</td>
<td>0.015</td>
<td>0.011</td>
<td>1000</td>
</tr>
<tr>
<td>q3</td>
<td>8600M</td>
<td>0.009</td>
<td>0.131</td>
<td>0.014</td>
<td>0.009</td>
<td>1000</td>
</tr>
<tr>
<td>q4</td>
<td>7M</td>
<td>0.011</td>
<td>0.180</td>
<td>0.018</td>
<td>0.012</td>
<td>1000</td>
</tr>
<tr>
<td>q5</td>
<td>230M</td>
<td>0.009</td>
<td>0.162</td>
<td>0.016</td>
<td>0.010</td>
<td>1000</td>
</tr>
<tr>
<td>q6</td>
<td>8600M</td>
<td>0.024</td>
<td>0.387</td>
<td>0.032</td>
<td>0.019</td>
<td>1000</td>
</tr>
<tr>
<td>q7</td>
<td>8600M</td>
<td>0.191</td>
<td>1.134</td>
<td>0.207</td>
<td>0.035</td>
<td>1000</td>
</tr>
</tbody>
</table>

for all three mappings, although the document size for each mapping is different. The outliers in q4 and q5 (avg column) of Table 3 are the consequence of bigger intermediate result sets during the applied steps, as the path goes along the rfc directory with approx. 4400 entries. This conclusion is supported by the fact that these queries are the only ones that really profit from the applied value index (see Table 4). For all short paths there is no real difference, but for longer paths with small intermediate result sets, the value index has a negative effect on the performance. This is due to the fact that the query is transformed to explicit child and parent steps along the path. Since the result set of each single step is relatively small, the additional lookup in the index does not pay off.

**Query scenario II: Modification and inspection of hierarchy and content**

*Add new directories.* Queries 8–10 create a new directory structure in the home directory through function `xqfs:mkdir-p('jokes/rfc/1st april')`. A check for the existence of the directory was removed. This allows the same ‘directory’ to exist a 1000 times beneath a common parent. Of course, this is not according to the POSIX specification, but suits well to test inserts into the directory hierarchy. The tests are performed with and without a value index (`fs:dir/@fs:name`) on the directory hierarchy. The construction time for the new directories yields a maximum value of \(\sim 300\) ms and the average is in the range of \(\sim 20\) ms for both variants.

*Add new content.* A single directory `/home/holu/jokes/rfc/1st april` is created. Queries 11–13 store document RFC3092 (“The Etymology of Foo”) in this directory. While the other queries are run 1000 times externally by PERFIDIX, this query is invoked only once and performs the loop inside the query itself. This allows to store the same file with different names (“1-RFC3092” to “1000-RFC3092”). The insertion of 1000 text files, each of size 28K, takes \(\sim 10\) sec for all three mappings. Though, if a fulltext index is supplied for all `<txt:content>`, the insertion time climbs up to \(\sim 6\) min! for `mappedfs.xml`.

*List a directory.* The previously stored files are listed by queries 14–16 through function `xqfs:ls('/home/holu/jokes/rfc/1st april/')`. The time for serialization is taken into account. For 1000 runs the average value is \(\sim 31\) ms.

*Remove content from storage.* Queries 17–19 call the function `xqfs:rm-rf('/home/ /holu/jokes')`. All previously stored RFCs and the directory structure are removed from the storage, which takes \(\sim 3\) sec for each mapping without full-text index.
Intermediate summary. The same conclusions can be derived as in the first experiment. The cache influence has a major effect, but, generally, the size of the corresponding mapping has no influence on the performed operations as they all operate locally. Additionally, the operations revealed quite a promising performance, in the sense that it is possible to operate interactively on the file system mappings.

Query scenario III: Searching for content

Exact search for a file name. Queries 20–22 define a search for an exact file name, returning the absolute pathname(s). Serialization time is included.

```plaintext
> doc(...)/xqfs:pwd(//fs:file[@fs:name = 'ssh'])
Query returned 6 results:
/etc/default/ssh
/etc/pan_d/ssh
/etc/init.d/ssh
/usr/bin/ssh
/usr/lib/apt/methods/ssh
/usr/ssh
```

Search for partial string. Queries 23 and 24 return album titles with the word 'Friede' as shown in Table 6.

```plaintext
> doc('mappedfs.xml')/xqfs:locate(., //mp3:albumtitle[contains(text(), 'Friede')])
Query returned 10 results:
/usr/local/share/music/BWV116 'Du Friedefürst.../...Christ.mp3
...;
/usr/local/share/music/BWV158 'Der Friede.../...Osterlamm.mp3
```

Fulltext search. A fulltext index is applied on txt:content with phrase support, insensitive search and removal of English stop words. Only mappedfs.xml is measured and

### Table 5: Exact search for a file name (with and without index on fs:file/@fs:name in sec.)

<table>
<thead>
<tr>
<th>no</th>
<th>docsize</th>
<th>min</th>
<th>max</th>
<th>avg</th>
<th>stddev</th>
<th>#runs</th>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>q20</td>
<td>7M</td>
<td>0.139</td>
<td>1.028</td>
<td>0.147</td>
<td>0.030</td>
<td>1000</td>
<td>-</td>
</tr>
<tr>
<td>q21</td>
<td>230M</td>
<td>0.198</td>
<td>1.175</td>
<td>0.210</td>
<td>0.034</td>
<td>1000</td>
<td>-</td>
</tr>
<tr>
<td>q22</td>
<td>8600M</td>
<td>2.461</td>
<td>18.935</td>
<td>2.537</td>
<td>0.520</td>
<td>1000</td>
<td>-</td>
</tr>
<tr>
<td>q20</td>
<td>7M</td>
<td>0.008</td>
<td>0.352</td>
<td>0.013</td>
<td>0.013</td>
<td>1000</td>
<td>+</td>
</tr>
<tr>
<td>q21</td>
<td>230M</td>
<td>0.009</td>
<td>0.388</td>
<td>0.014</td>
<td>0.017</td>
<td>1000</td>
<td>+</td>
</tr>
<tr>
<td>q22</td>
<td>8600M</td>
<td>0.010</td>
<td>0.325</td>
<td>0.016</td>
<td>0.017</td>
<td>1000</td>
<td>+</td>
</tr>
</tbody>
</table>

### Table 6: Search for album title to contain 'Friede' in ID3 information (in sec.)

<table>
<thead>
<tr>
<th>no</th>
<th>docsize</th>
<th>min</th>
<th>max</th>
<th>avg</th>
<th>stddev</th>
<th>#runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>q23</td>
<td>230M</td>
<td>0.171</td>
<td>1.621</td>
<td>0.182</td>
<td>0.049</td>
<td>1000</td>
</tr>
<tr>
<td>q24</td>
<td>8600M</td>
<td>1.791</td>
<td>3.454</td>
<td>1.868</td>
<td>0.060</td>
<td>1000</td>
</tr>
</tbody>
</table>
revealed the expected fast access rates for phrase, wildcard, and boolean queries. The more interesting results, however, are the failures: For the 8.6G mapping it was not possible to even build the fulltext index, as a requested array size exceeded the available memory. Ranking of end results is not yet possible and together with the immense slow down of update operations the result is contrary to our expectations. Since the fulltext index is largely based on the open-source project Lucene [Luc06], it seems that the integration and adoption to a semi-structured database has not yet been pushed to its limits.

Intermediate summary. To maintain an index for the directory hierarchy, \texttt{fs:dir/@fs:name} and \texttt{fs:file/@fs:name} is a justifiable approach. There is a certain trade-off for very explicit path traversals with small intermediate results, but such queries could be rewritten to make better usage of the index. The number of files and directories are of relatively small size, maintaining the index is thus feasible. While the fulltext index shows an excellent retrieval performance, it is apparently still unsuitable for large documents, and the results might indicate that X-HiVE does not seem to have focused on fulltext updates yet. This is surely a field for further research, since two cutting-edge issues (fulltext retrieval in XML and update functionality) are involved. Still, the speed of fulltext updates in major relational databases as well as in desktop search engines supports our assumption that fulltext indexing can yield promising performance results.

The discussed evaluation shows first, specific results for X-HiVE. Although being a generic XML processor, many of the results show interactive response times.

6 Future Work and Conclusion

Work in progress. Currently, there are two closely related projects, i.e., IDEFIX and BASEX [GHK+06] in our department, where we try to combine file system and database technology. IDEFIX, a block-oriented persistent XML storage layer, is prepared to serve as a back-end for the open-source XQuery compiler PATHFINDER [PF06]. IDEFIX evaluates the relational algebra emitted by PATHFINDER for its native XML encoding and serializes the final query results. The BASEX FILE SYSTEM (BXFS), a user-space file system, will serve as second step towards operating system integration. BASEX uses a native tree encoding (derived from [Gru02]) to store data and is able to process basic file system commands. A graphical user interface to visually explore and query the (file system) data will be presented at the demonstration panel of the conference [GHS07].

Contribution. In the scope of this publication, we provided a proof-of-concept. The question whether a state-of-the-art XML-aware database management system is capable to process file system operations as well as demanded query functionality on file system data represented as XML has been evaluated. We found that navigation along the directory/content structure is independent of the amount of stored content in the representation. Basic file system commands, as well as content retrieval, can be performed in interactive time on the constructed file system mappings with a general-purpose XML-aware database.

As final conclusion it can be stated that traditional file systems are, of course, not obsolete
in terms of mere storage. As soon as the demand for querying and retrieval preponderates the processing of file systems, using semi-structured database techniques to enhance file system capabilities is a clear option.

References


Algebraic Query Optimization for Distributed Top-k Queries

Thomas Neumann, Sebastian Michel
Max-Planck-Institut für Informatik
Saarbrücken, Germany
{neumann, smichel}@mpi-inf.mpg.de

Abstract: Distributed top-k query processing is increasingly becoming an essential functionality in a large number of emerging application classes. This paper addresses the efficient algebraic optimization of top-k queries in wide-area distributed data repositories where the index lists for the attribute values (or text terms) of a query are distributed across a number of data peers and the computational costs include network latency, bandwidth consumption, and local peer work. We use a dynamic programming approach to find the optimal execution plan using compact data synopses for selectivity estimation that is the basis for our cost model. The optimized query is executed in a hierarchical way involving a small and fixed number of communication phases. We have performed experiments on real web data that show the benefits of distributed top-k query optimization both in network resource consumption and query response time.

1 Introduction

Top-k query processing has received great attention for a variety of emerging application classes including multimedia similarity search [GM04, NR99, Fag99, GBK00, BGRS99, NCS+01], preference queries over product catalogs and other Internet sources [CK98, MBG04, BGM02], ranked retrieval of semistructured (XML) documents [GSBS03, TSW05], and data aggregation for “heavy hitters” in network monitoring and other sensor-network applications [CKMS04, CJSS03]. Such queries typically need to aggregate, score, and rank results from different index lists or other forms of data sources. Most of the prior work concentrates on centralized settings where all index lists reside on the same server or server farm. The case where the sources are distributed across a wide area network and the optimization of communication costs is crucial has received only little attention, the most noteworthy exceptions being the TPUT algorithm [CW04], our own prior work on KLEE [MTW05], and the work by [MBG04] and [ZYVG+1] both of which build on the family of threshold algorithms (TA) that seems predominant in centralized top-k querying [FLN03, GBK00, NR99].

For the centralized case, the need for choosing good query execution plans from a large space of possible plans has been recognized, and research has started towards a better understanding of algebraic and cost-based optimization of top-k queries. The most prominent, recent example of this line of work is the RankSQL framework [ISA+04, LCIS05], which considers binary trees of outer joins with score aggregation and ranking, so-called “rank joins”. The TA family, on the other hand, is a fixed execution strategy with a single m-ary join over precomputed index lists where m is the number of attributes or keywords in the score-aggregation function. For the distributed setting, only fixed algorithms have...
been proposed without consideration to flexible, data- and/or workload-dependent, construction of different execution plans. For example, TPUT and KLEE, the best known algorithms, first fetch the best \(k\) data items from each of the underlying \(m\) index lists or data sources, compute the aggregated score \(s\) of the rank-\(k\) item in this intermediate result, and then issue score-range queries against the different sources to fetch all items with per-attribute scores above \(s/m\). Despite some additional optimizations for better pruning of result candidates, this approach does not explore at all the space of forming different execution trees of joining and aggregating scored items in a flexible manner with attention to networking and local computation costs.

The current paper is the first work that addresses this issue. In contrast to [LCIS05] we consider distributed execution and variable-arity trees. In contrast to [CW04, MTW05] we consider different execution plans, based on estimates of data characteristics, and aim to find the cost-optimal execution plan by algebraic rewriting and a novel way of dynamic-programming-style search for a cost minimum. Compared to the ample work on distributed join queries [MTO99, Kos00], our method is ranking-aware and aims to terminate the query execution as early as possible once the top-\(k\) results can be safely identified. More specifically, this paper makes the following research contributions:

- a new model for describing, systematically enumerating, and algebraically rewriting the feasible execution plans in a large space of plans for distributed top-\(k\) queries over wide-area networks,
- a new form of dynamic-programming-style optimization for finding the cost-minimal execution plan with cost estimation based on score-distribution statistics,
- a new way of estimating and aggregating statistical score-value distributions, using Poisson mixtures or equi-depth histogram convolutions, to assess the costs of an execution plan, with the salient property that the statistical synopses themselves are very small and thus network-friendly in the optimization phase,
- comprehensive performance studies on large, real-world web data collections, showing that significant performance benefits can be achieved with acceptable overhead of the optimization step itself.

The paper is organized as follows. Section 2 introduces our computational model. Section 3 discusses relevant work on distributed top-\(k\) querying and on query optimization in general. Section 4 presents the query optimization techniques together with the actual query processing algorithms. Section 5 presents the cost model and shows how we use distributed statistics to characterize the input data and estimate selectivities. Section 6 presents an experimental evaluation that shows the impact of query optimization on distributed top-\(k\) queries. Section 7 concludes the paper and presents ongoing and future work.

2 Computational Model

We consider a distributed system with \(N\) peers, \(P_j, j = 1, \ldots, N\), that are connected, e.g., by a distributed hash table or some overlay network. Data items are either documents such as Web pages or structured data items such as movie descriptions. Each data item has associated with it a set of descriptors, text terms or attribute values, and there is a precomputed score for each pair of data item and descriptor. The inverted index list for one descriptor is the list of data items in which the descriptor appears sorted in descending order of scores. These index lists are the distribution granularity of the distributed system. Each index list is assigned to one peer (or, if we wish to repli-
cate it, to multiple peers). In the following we use only IR-style terminology, speaking of "terms" and "documents", for simplicity. Each peer $P_j$ stores one index list, $I_j(t)$, over a term $t$. $I_j(t)$ consists of a number of (docID, score) pairs, where score is a real number in $(0, 1]$ reflecting the significance of the document with docID for term $t$. Each index list is assumed to be sorted in descending order according to score. In general, score(docID) reflects the score associated with docID in an index list, e.g., a $tf \times idf$-style $(term - frequency \times inverse - document - frequency)$ or language-model-based measure derived from term frequency statistics. A query, $q(T, k)$, initiated at a peer $P_{init}$, consists of a nonempty set of terms, $T = \{t_1, t_2, ..., t_t\}$, and an integer $k$. Assuming the existence of a set of, say $m$, peers having the most relevant index lists for the terms in $T$, with $m = t$, our task is to devise efficient methods for $P_{init}$ to access these distributed index lists at the $m$ peers, so as to produce the list of (the IDs of) the top-$k$ documents for the term set $T$. The top-$k$ result is the sorted list in descending order of TotalScore which consists of pairs (docID, TotalScore), where TotalScore for a document with ID docID is a monotonic aggregation of the scores of this document in all $m$ index lists. For the sake of concreteness, we will use summation for score aggregation, but weighted sums and other monotonic functions are supported, too. In case an index list does not contain a particular docID, its score for docID is set to zero, when calculating its TotalScore. Note that $P_{init}$ serves as a coordinator only for the given query; different queries are usually coordinated by different peers. However, we will see later that $P_{init}$ can forward the coordination task to another peer that is involved in the query if this is promising to decrease the overall query execution cost. A naive solution would be to have all $m$ cohort peers (i.e. peers that maintain an index list for a specific keyword) send the complete index lists to $P_{init}$ and then execute a centralized TA-style method on the copied lists at $P_{init}$. This approach is unacceptable in a P2P system for its waste of network bandwidth resulting from transferring complete index lists. An alternative approach would be to execute TA at $P_{init}$ and access the remote index lists one entry at a time as needed. This method is equally undesirable for it incurs many small messages and needs a number of message rounds that is equal to the maximum index-scan depth among the participating peers. Even when messages are batched (e.g., with 100 successive index entries in a single message), the total latency of many message rounds renders this approach unattractive.

3 Related Work

Top-k query processing has received much attention in a variety of settings such as similarity search on multimedia data [GM04, NR99, Fag99, GBK00, BGRS99, NCS+01], ranked retrieval on text and semi-structured documents in digital libraries and on the Web [AdKM01, LS03, TWS04, KKNR04, BJR03, PZSD96, YSMQ], network and stream monitoring [BO03, CW04] collaborative recommendation and preference queries on e-commerce product catalogs [YPM03, MBG04, GBK01, CWH02], and ranking of SQL-style query results on structured data sources in general [ACDG03, CDHW, BCG02]. Among the ample work on top-k query processing, the TA family of algorithms for monotonic score aggregation [FLN03, GBK00, NR99] stands out as an extremely efficient and highly versatile method. TA-sorted (aka. NRA) variants process the (docID, score) entries of the relevant index lists in descending order of score values, using a simple round-robin scheduling strategy and making only sequential accesses on the index lists. TA-sorted maintains a priority queue of candidates and a current set of top-$k$ results, both in memory. The algorithm maintains with each candidate or current top-$k$ document $d$ a score interval,
with a lower bound worstscore(d) and an upper bound bestscore(d) for the true global score of d. The worstscore is the sum of all local scores that have been observed for d during the index scans. The bestscore is the sum of the worstscore and the last score values seen in all those lists where d has not yet been encountered. We denote the latter values by high(i) for the ith index list; they are upper bounds for the best possible score in the still unvisited tails of the index lists. The current top-k are those documents with the k highest worstscores. A candidate d for which bestscore(d) < min_k can be safely dismissed, where min_k denotes the worstscore of the rank-k document in the current top-k. The algorithm terminates when the candidate queue is empty (and a virtual document that has not yet been seen in any index list and has a bestscore ≤ \sum_{i=1}^m high(i) can not qualify for the top-k either). For approximating a top-k result with low error probability [TWS04], the conservative bestscores, with high(i) values assumed for unknown scores, can be substituted by quantiles of the score distribution in the unvisited tails of the index lists. Technically, this amounts to estimating the convolution of the unknown scores of a candidate. A candidate d can be dismissed if the probability that its bestscore can still exceed the min_k value drops below some threshold: \Pr[worstscore(d) + \sum S(i) > min_k] < \varepsilon, where the S(i) are random variables for unknown scores and the sum ranges over all i in which d has not yet been encountered.

The first distributed TA-style algorithm has been presented in [BGM02, MBG04]. The emphasis of that work was on top-k queries over Internet data sources for recommendation services (e.g., restaurant ratings, street finders). Because of functional limitations and specific costs of data sources, the approach used a hybrid algorithm that allowed both sorted and random access but tried to avoid random accesses. Scheduling strategies for random accesses to resolve expensive predicates were addressed also in [CwH02]. In our widely distributed setting, none of these scheduling methods are relevant for they still incur an unbounded number of message rounds. The method in [SMW+W03] addresses P2P-style distributed top-k queries but considers only the case of two index lists distributed over two peers. Its key idea is to allow the two cohort peers to directly exchange score and candidate information rather than communicating only via the query initiator. Unfortunately, it is unclear and left as an open issue how to generalize to more than two peers. The recent work by [BNST05] addresses the optimization of communication costs in P2P networks. However, the emphasis is on appropriate topologies for overlay networks. The paper develops efficient routing methods among super-peers in a hypercube topology. TPUT [CW04] executes TA in three phases: 1) fetch the k best (DocID, Score) entries from each cohort peer and compute the min_k score (i.e., the score of the item currently at rank k) using zero-score values for all missing scores; 2) ask each of the m cohort peers for entries with Score > min_k/m, then compute a better min_k value and eliminate candidates whose bestscore is not higher than min_k; 3) fetch the still missing scores for the remaining candidates, asking the cohorts to do random accesses. [YLG05] presents a modification of TPUT that adapts the min_k/m threshold to the score distributions’ peculiarities. KLEE [MTW05] is an approximate version of TPUT that leverages Bloom filter and histograms to prune / filter unpromising documents. [DKR04] considers hierarchical in-network data aggregation where the query hierarchy is given by the network topology. [ZYVG^+] presents a threshold algorithm for distributed sensor networks, here, again, the hierarchy is predetermined by the network. [CW04] shortly mentions a hierarchical version of TPUT but does not consider optimized query plans.

Query processing in general is a well studied standard technique. We employ an optimization algorithm that uses a search space exploration technique similar to the Volcano
Figure 1: Two equivalent execution plans

approach [GM93]: The search space is explored in a top-down manner by splitting the problem into smaller problems and solving these recursively, memoization prevents duplicate work. The other standard techniques include bottom-up dynamic programming [HFLP89] and transformative optimization [Gra95]. However, only a few papers consider the peculiarities of top-\( k \) queries during the optimization phase and treat the actual top-\( k \) processing as orthogonal to the main query optimization step. One notable exception is the RankSQL proposal [LCIS05], which integrates the decision about rank computation and usage into the query optimization. It extends the relation algebra towards a rank-aware algebra using an extended search space that includes the available rank aware functions. In contrast to their work, we assume that the basic rank (score) is already known. The problem that arises when dealing with score aggregations is not sufficiently covered in [LCIS05] as the paper is focused on the optimal computation of rank functions. Although they do recognize the problem they only consider scores that follow normal distributions [ISA + 04], or they use sampling [LCIS05], which is, from our point of view, prohibitively expensive in a distributed setting.

4 Algorithms

In this paper we use an algorithm that, besides the actual query optimization, essentially consists of the 3-phase algorithm TPUT in an hierarchical environment as shortly mentioned in [CW04]. The query initiator \( P_{init} \) retrieves the top-\( k \) documents from the involved cohort peers and calculates a top-\( k \) estimate by aggregating the scores for the particular documents. \( \text{min}_k \) denotes the partial score of the document at rank \( k \). In a flat structure, \( P_{init} \) would send \( \text{min}_k \) as a threshold to the cohort peers that return all documents above this threshold. In a hierarchical environment, however, the threshold for a particular cohort is based recursively on the threshold of the parent cohort and the number of siblings. For instance, consider the example in Figure 1: In the left query plan, the query initiator sends \( 0.00028 \) (i.e. \( \text{min}_3 \)) to the cohort peers, this refers to an estimated \( \text{min}_k \) of \( 0.00084 \). In the left query plan the \( P_{init} \) estimates the same \( \text{min}_k \) but sends the threshold \( 0.0004 \) (i.e. \( \frac{\text{min}_k}{2} \)) to its children. The child at the left receives the threshold \( 0.0004 \) and forwards the new threshold \( \frac{0.0004}{2} \) to its children. For deeper hierarchies this procedure is applied recursively.
4.1 Overview

We now assume that a query with \( m \) terms is started at a peer \( P_{\text{init}} \), which has to produce the top-\( k \) results, and that the \((\text{docId}, \text{score})\) lists are stored at \( m \) cohort peers. Note that these peers need not be disjoint, as one peer can handle multiple terms, and \( P_{\text{init}} \) can also be one of the cohort peers. To model this in the algorithms, we introduce the concept of sites: The basic assumption is that if two index lists are at the same site, they can be accessed simultaneously. Usually a site corresponds to one physical peer. We name the set of all sites relevant for the query \( S \), \( |S| \leq m + 1 \).

Our algorithms combine the concept of cost based query optimization with the basic execution paradigm of TPUT [CW04]. This means that the execution is done in the following four phases, where the phases 1, 3 and 4 are the same for all algorithms.

1. **Exploration Step**: \( P_{\text{init}} \) communicates directly with the \( m \) cohort peers and retrieves the top-\( k \) documents along with statistics and score models that will be used in the selectivity estimation.

2. **Query Optimization**: \( P_{\text{init}} \) constructs the execution plan according to the selected algorithm.

3. **Candidate Retrieval**: \( P_{\text{init}} \) sends the execution plan along with the estimated \( \frac{\min_k}{k} \) thresholds to the involved peers. After having received this information, thus being able to establish connections to children and parent peers the leaf peers start sending \((\text{docId}, \text{score})\)-pairs to their parents. Subsequently, the former parents, now being treated as children, combine the received information and forward the \((\text{docId}, \text{worstscore})\)-pairs to their parents, and so on, until the computations has reached the root node. The pruning in the intermediate top-\( k \) steps is executed w.r.t. the documents’ bestscores. Note that the bestscores are updated during the intermediate merge.

4. **Missing Scores Lookup Phase**: \( P_{\text{init}} \) constructs the final list of top-\( k \) candidates, determines the documents with missing scores, and sends the docIds to the corresponding peers. The peers lookup the missing scores and send them to \( P_{\text{init}} \), which produces the top-\( k \) result.

This phase structure is identical to the TPUT phases (except the optimization phase), and in fact TPUT can be formulated as a special case of this algorithm, as we will see below.

4.2 Algebraic Optimization

Query optimization requires the evaluation of different equivalent execution alternatives. Therefore, the query optimizer has to decide if two alternatives are really equivalent. This is usually done by formulating the query in a formal algebra and using algebraic equivalences. What is different here (in the top-\( k \) case) is that these equivalences are somewhat relaxed. The query optimizer also accepts alternatives that are not equivalent in a strict sense, as long as the top-\( k \) entries are the same.

This basic premise is shown in Figure 2 and is the idea behind most top-\( k \) execution strategies: The problem of finding the top \( k \) tuples is reduced to finding all tuples with a sufficient score. Due to the subset relation, more than \( k \) tuples can satisfy the condition. This relaxation allows for more efficient evaluation strategies, as reasoning about score is much simpler than reasoning about ranks.
\( F_k(S) = \) the first \( k \) tuples of \( S \)
\( \text{min}_k(S) = \min\{s.\text{score}|s \in F_k(S)\} \)
\( \sigma_{\geq t}(S) = \{s \in S|s.\text{score} \geq t\} \)
\( \text{top-}k(S) \subseteq \sigma_{\text{min}_k(S)}(S) \)

\( C_k(S_1 \ldots S_n) = \bigcup_{1 \leq i \leq n} F_k(S_i) \)
\( \text{min}_k(S_1 \ldots S_n) = \min(C_k(S_1 \ldots S_n)) \)
\( \text{top-}k(S_1 \ldots S_n) = \text{top-}k(\bigcup_{1 \leq i \leq n} S_i) \)
\( \text{top-}k(S_1 \ldots S_n) \subseteq \bigcup_{1 \leq i \leq n} \sigma_{\geq \text{min}_k(S_1 \ldots S_n)}/n(S_i) \)

In practice, the exact (ideal) threshold \( \text{min}_k(S) \) is not known a-priori and is approximated (conservatively). TPUT does this by sampling the first \( k \) entries of each input. The TPUT approach is formalized in Figure 3; we concentrate on aggregation by score summation (\( \bigcup \)), other aggregations schemes can be used analogously. The basic idea is to estimate \( \text{min}_k(S) \) by fetching some tuples (\( k \) from each list) and using the \( k \)-th score in the aggregation as a lower bound for \( \text{min}_k(S) \). This is a safe choice, as at least \( k \) tuples above the thresholds have already been seen. Now all tuples in the lists have to have a score of at least \( \text{min}_k(S)/n \), that can be shown easily.

While this is a nice formalism to get a suitable threshold for aggregation, it offers no optimization opportunities. The threshold propagation is fixed. The TPUT paper [CW04] mentions the possibility of hierarchical TPUT, but offers no formalism. We now provide a formalism and suitable equivalence rules to allow for an algebraic optimization of the TPUT structure. Figure 4 shows the formalism of our approach: First, we do not push the threshold down in a fixed scheme but use a more general transformation as in Figure 2. This is just a minor variation, but essential to give the query optimizer freedom to optimize. More importantly, we present an equivalence that splits a top-\( k \) problem into smaller top-\( k \) problems that can be solved recursively. This equivalence is used by the query optimizer to find the optimal execution strategy. Note that TPUT itself is a special case of this equivalence, it partitions the input directly into single lists.

\[ \text{top-}k(S_1 \ldots S_n) \subseteq \sigma_{\geq \text{min}_k(S_1 \ldots S_n)}(\bigcup_{1 \leq i \leq n} S_i), \]
\[ \sigma_{\geq t}(S_1 \ldots S_n) \equiv \sigma_{\geq t}(\bigcup_{1 \leq i \leq l} \sigma_{\geq t/1}(P_i)) \]
where \( P_1 \ldots P_l \) is a partitioning of \( S_1 \ldots S_n \)

4.3 Building Blocks

The previous section presented the algebraic optimization. However, the formulation was very high level. A more detailed look is required to understand the distributed execution, furthermore some logic is required to handle the random lookups in the last phase of the algorithm. We concentrate on the logical operators here, as the corresponding physical operators used during plan generation are mostly obvious. Thus we assume that only one
physical operator exists for each logical operator. However, this is not a limitation of the algorithms, they can easily handle alternative physical operators. This could be useful for different evaluation strategies, e.g. for using random lookups in the last phase vs. scanning the lists sequentially.

For each logical operator we specify the optimization rule that is used to create an appropriate plan. As we optimize in a distributed system, the optimizer must be able to ship data to other sites. This is done by the transfer operator, the rule checks if the output is at the correct site and ships otherwise:

$$\text{transfer}(\text{target}, p)$$

if $p$.site = target
    return $p$
else
    return a plan that ships $p$ to target

To make the decision about data transfers easier, the algorithms always consider all possible sites in each step. The output of a step is therefore not a single plan, but a set of plans, with one plan for each possible result site. Since it might be cheaper to perform a calculation at one site and ship only the result, the optimization rule ship examines all pairs of sites $(s_1, s_2)$ and considers using a transfer instead of performing the computation locally:

$$\text{ship}(\text{plans})$$

for $\forall s_1 \in S, s_2 \in S, s_1 \neq s_2$
    $p = \text{transfer}(s_2, \text{plans}[s_1])$
    if $p$.costs < $\text{plans}[s_2]$.costs
        $\text{plans}[s_1] = p$
return plans

The top-k operation itself consists of three operations: base-top-k scans a list and produces all entries above a $\min_k$ threshold (this corresponds to $\sigma_{\geq \min_k}(S)$), intermediate-top-k combines intermediate results ($\sigma_{\geq \min_k}(S_1 \ldots S_n)$) and top-k produces the final top-k list (the rank based top-k operator). The base rule simply inserts a base-top-k operator, it only has to make sure that the score distribution is adjusted according to the $\min_k$:

$$\text{base-top-k}(\min_k, t)$$

$p = \text{a new plan to scan } t$
$p$.card = documents above $\min_k$
$p$.scoreDistribution = distribution $\geq \min_k$
plans = empty plan set
$\text{plans}[\text{site of } t] = p$
return ship(plans)

The intermediate steps combines multiple partial top-k results (either from base-top-k or another intermediate step) into a new partial result. The document scores are aggregates, pruned according to the given $\min_k$ threshold and the resulting tuples are sorted by descending score. As the intermediate step can combine an arbitrary number of intermediate results the input of this building block is a set of plansets, one planset per input operator. The rule iterates over all sites and combines the plans for each site.

331
intermediate-top-k($min_k, input$)

```plaintext
plans = empty plan set
for $\forall s \in S$
  $i = \{p[s]|p \in input\}$
  $h = \text{convolution of } \{p.scoreDistribution|p \in i\}$
  $plans[s] = \text{a new plan to combine } i$
  $plans[s].card = h.documents\text{ above } min_k$
  $plans[s].scoreDistribution = \text{part of } h \geq min_k$
return plans
```

The final top-$k$ operator takes an intermediate result and constructs the final top-$k$ list by retrieving the missing scores. Note that we only have to consider the unary case, otherwise an intermediate top-$k$ can be used for aggregation.

top-k($min_k, plans$)
```
plans = empty plan set
for $\forall s \in S$
  plans[s] = \text{a new plan to finalize } plans[s]
  plans[s].card = k
return plans
```

4.4 Algorithms

Using these building blocks, the optimization algorithms can be formulated easily. To illustrate the plan construction, we first formulate TPUT for a set of terms $T$ using these constructs:

tput($T, min_k$)
```
b = \{\text{base-top-k}($min_k/|T|, t$)|t \in T\}
i = \text{intermediate-top-k}($min_k, b$)
p = \text{top-k}($min_k, i$)
return \text{p}[P_{init}]$
```

TPUT reads all lists up to $min_k/|T|$, combines the results and finally calculates the top-$k$ by looking up the missing scores. The last two steps are always executed at $P_{init}$, therefore the plan for $P_{init}$ is returned.

A simple improvement of the TPUT algorithm is what we call the simple optimization algorithm: It behaves like TPUT, but considers performing the aggregation at a different peer than $P_{init}$. As the final step only produces $k$ tuples, pushing the aggregation down can greatly reduce the costs. However, this decision must be made cost based: Pushing the aggregation down induces additional latency, which can be higher than the gain from the push down. The push-down is done by using the ship rule described above.

simple($T, min_k$)
```
b = \{\text{base-top-k}($min_k/|T|, t$)|t \in T\}
i = \text{intermediate-top-k}($min_k, b$)
p = \text{top-k}($min_k, i$)
```
A much larger class of execution plans is possible when allowing additional intermediate aggregations. As already mentioned in [CW04], a tree structure can allow for much larger min-$k$ thresholds. However the optimization is also more difficult, especially since an intermediate step can aggregate an arbitrary number of input streams. The optimization is split in two parts: One finds the optimal way to construct intermediate results and the other constructs the optimal top-$k$ after the intermediate results are known.

The intermediate results can be optimized recursively: Given a set of terms, the optimizer recursively solves all partitionings of the terms and combines them to an intermediate result. Dynamic programming (DP) is used to reduce the search space, as term combinations are used multiple times. The DP table maps from \((\text{terms}, \text{min} \, k) \rightarrow \text{(planset)}\), i.e., for each term/minimum score combination the optimal plan for each site is stored. Here we use the top-down formulation of dynamic programming (memoization), which naturally follows the recursive optimization scheme:

\[
\begin{align*}
solveIntermediate(T, \text{min} \, k) \\
\text{if } (T, \text{min} \, k) \text{ has already been solved} & \quad \text{return the known solution} \\
\text{if } |T| = 1 & \quad p = \text{base-top-k}(\text{min} \, k, t \in T) \\
\text{else} & \quad p = \text{empty plan set} \\
& \quad \text{for } \forall P = \{T_i \subset T\}, P \text{ partitioning of } T \\
& \quad \quad i = \{\text{solveIntermediate}(T_i, \text{min} \, k/|P|)|T_i \in P\} \\
& \quad \quad p' = \text{intermediate-top-k}(\text{min} \, k, i) \\
& \quad \quad \text{for } \forall s \in S \\
& \quad \quad \quad \text{if } p'[s].\text{costs} < p[s].\text{costs} \\
& \quad \quad \quad \quad p[s] = p'[s] \\
& \quad \quad p = \text{ship}(p) \\
& \quad \quad \text{store } p \text{ as solution for } (T, \text{min} \, k - k) \\
\text{return } p
\end{align*}
\]

Note that the algorithm is simplified, see Section 4.5 for performance improvements. The optimal intermediate results can now be used to construct the complete plan quite easily, as the largest intermediate result (all terms) can be used as input for the final top-$k$ operator:

\[
\begin{align*}
optimal(T, \text{min} \, k) \\
& \quad i = \text{solveIntermediate}(\text{min} \, k, T) \\
& \quad p = \text{top-k}(\text{min} \, k, i) \\
& \quad p = \text{ship}(p) \\
\text{return } p[P_{init}]
\end{align*}
\]

The \textit{optimal} algorithm considers the whole search space of possible execution trees. In particular, it never produces a worse plan than \textit{input} or \textit{simple}, as these algorithm produce plans that are inside this search space.

Figure 5 shows an example of an optimized query plan, an output of the above stated optimization technique.
4.5 Large Queries

As stated above, the optimal algorithm and especially the solveIntermediate function considers the whole search space, which, even when memoization is used, means at least $2^{|T|}$ different plans (more because of different $\min_k$ values). While this is negligible for common queries (e.g. GOV queries with at most 4 terms), the search space becomes a problem for large queries, e.g. XGOV queries with up to 18 terms.

The search space can be reduced by using cost bound pruning: For a global cost bound the optimizer can use a cheap heuristic (e.g. simple) to get a plan quickly, which can be used to prune plans early. During the optimization the optimizer can use already constructed solutions to prune against later ones. In particular, it is possible to find lower cost bounds for a given term set by looking at the original score distributions: If a tuple is already above $\min_k$ without convolution it will also be after the convolution. These heuristics allow estimating the costs early, which greatly reduces the search space.

Still the search space grows exponentially, and depending on the implementation and the available hardware the optimal solution will be too expensive to compute for certain queries. Then an idea from the query optimization for joins can be used: Iterative dynamic programming [KS00] performs the full DP computation only for subproblems with a given maximum, chooses one such solution, removes it from the search space and repeats the DP step until the problem is solved. For the top-$k$ processing this means that optimal intermediate results for up to a given size are computed, then one solution is chosen (we used the cheapest one) and the terms occurring in this solution are considered solved and replaced by a new pseudo-term. This is repeated until the problem is solved. While this does not guarantee the optimal solution, it results in quite good plans in practice (and of course the optimal solution is still possible).
5 Distributed Statistics and Cost Model

Using query optimization techniques for distributed top-$k$ retrieval assumes that some execution alternatives are better than others. This can be formalized using a cost model, which describes how many resources are used by each alternative. This requires accurate cardinality estimations to find the optimal execution plan. While this is always the case for query optimization in general, the cardinalities during top-$k$ processing depend on the scores of the computed data. Therefore standard cardinality estimators are not enough, the query optimizer needs to estimate score distributions. Prior work on this issue either used fairly crude models like assuming Normal distributions [ISA+04], which is not a good fit for real-data scores, or required extensive computations like sampling of histogram maintenance [TWS04] that may incur high costs in a distributed setting with a high-latency network.

When optimizing distributed queries, there exist mainly two interesting optimization goals: Either minimize the query response time or minimize the total resource consumption, especially the network bandwidth. Minimizing the query response time is obviously relevant, as someone (either a user or a program) is waiting for the answer. Minimizing the network bandwidth is also interesting, as it allows for more parallel queries. The algorithms presented in the next section can handle both goals and we show experimental results for both in Section 6, but we concentrate on minimizing response time.

In the following we will shortly discuss the statistics that we use as a basis for our cost model. The gathering of these distributed statistics is then part of the first phase of our algorithm: The query initiator retrieves from each peer the top $k$ documents along with the statistics for the particular attribute (i.e. term).

The main difficulty here is, however, to precisely describe single (per-attribute) score distributions in a way that allows for a highly accurate prediction of the number of documents with score above a certain threshold. Moreover, as we are interested in employing a hierarchical top-$k$ algorithm, and thus during optimization logically split the top-$k$ query into several sub-queries we additionally need score distribution models for aggregated data, as we will see below.

For text-based IR with keyword queries, the query-to-document similarity function is typically based on statistics about frequencies of term occurrences, e.g., the family of $tf\cdot idf$ scoring functions [Cha02] or more advanced statistical language models [CL03]. Here, terms are canonical representations of words (e.g., in stemmed form) or other text features.

In this work we consider Poisson Mixture Models and Equi-Depth Histograms to model score distributions for single-term index lists. Score distributions of multi-term index lists (i.e. combined single-term index lists using score aggregations) are computed as convolutions [All90].

Given an index list for a particular term we can model the frequency of occurrences using a Poisson Distribution:

$$\pi(k, \theta) = \frac{e^{-\theta} \cdot \theta^k}{k!}$$

$\pi(k, \theta)$ is the probability that a particular term occurs exactly $k$ times in a particular document. The parameter $\theta$ is the mean of the distribution and is used to adapt the Poisson model to a given distribution. A nice property of Poisson distributions is that the convolution of a Poisson distribution with parameter $\theta_1$ and a Poisson distribution with parameter
\( \theta_2 \) is a Poisson distribution with parameter \( \theta_1 + \theta_2 \). No other probabilistic distribution has this property that the convolution reproduces the same distribution function just with different parameters.

Unfortunately, simple Poisson distributions are not a particularly good fit for capturing the scores of real data. However, mixtures of Poisson distributions are a fairly accurate, realistic model [CG]. In our work we use Two Poisson Mixes to describe the score distribution for each index list in an ultra compact way as each Two Poisson Model requires only 3 floating point numbers so that the additional network resource consumption is negligible. The Two Poisson Model is a simple example of a Poisson mixture:

\[
P_{2p}(x, \theta) = \alpha \pi(x, \theta_1) + (1 - \alpha) \pi(x, \theta_2)
\]

[Har75] showed how to use the method of moments to fit the three parameters of the Two Poisson Model \( \theta_1, \theta_2, \) and \( \alpha \), from the first three moments.

However, in general, Poisson Mixture Models suffer from the inability to capture extreme variations in the score distributions. Histograms are a better tunable approach to represent score distributions but the convolution of equi-depth histograms is computationally expensive and they cause higher network traffic than Two Poisson Mixes, albeit providing a more accurate estimation so that we are able to trade off accuracy vs efficiency.

For Poisson Mixes, and Histograms the quality of the convolution w.r.t. to the score distribution of the true data depends to a large extent on the cardinality of the intersection of the involved data. Without detailed knowledge of the cardinality of the resulting data, the score distribution model will be way off, thus the query optimizer cannot work properly. For instance, recall that the convolution of two Poisson distributions is a Poisson distribution with the mean being the sum of the two means of the original distributions. In an extreme case the underlying data of the two distributions is nearly disjoint, thus the mean of the convolution is not at all equal to the sum of means. As adequate score distribution models are an essential part of an effective query optimizer, we have focused on integrating information about the input data’s mutual correlation into the convolution of the data’s score distribution. In particular, we are interested in the cardinality of the intersection (union) of two data sets (i.e. index lists). Estimating overlap of sets has been receiving increasing attention for modern emerging applications, such as data streams, internet content delivery, etc. In prior work [MBTW06] we have conducted a comprehensive evaluation of Bloom Filters [Blo70], Hash Sketches [FM85], and min-wise independent permutations (MIPs) [BCFM00] that show that MIPs are best suitable for our purpose.

Now the query optimizer uses these statistics to estimate the number of tuples above a certain threshold. When only considering base relations (i.e. unaggregated data), this estimation can be done directly by using the available score distribution. The only point to keep in mind is that the threshold affects the score distribution, as all tuples below the threshold are missing afterwards. But this can be implemented easily. More interesting is the score distribution of aggregated data: Assuming independence, the score distribution of the sum of two scores can be estimated by a simple convolution of the score distributions of the summands:

\[
P_{S_1+S_2}(x) = \sum_{i+j=x} P_{S_1}(i) * P_{S_2}(j)
\]

However this estimation is only correct if the scores are indeed summed up. Here, this means that the estimation assumes that every tuple in \( S_1 \) finds a matching tuple in \( S_2 \). If this not the case (e.g. if \( S_1 \) and \( S_2 \) are disjoint), the score is overestimated. We takes this effect into account by using the intersect and union estimations provided by MIPs: Only the tuples in the intersection are convoluted, for the others the score distribution is
unchanged. This lead us to the following estimation:

\[
P_{S_1+S_2}(x) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \left[ \sum_{i+j=x} P_{S_1}(i) \times P_{S_2}(j) \right] + \\
(1 - \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}) \left[ \frac{|S_1| P_{S_1}(x) + |S_2| P_{S_2}(x)}{|S_1| + |S_2|} \right] .
\]

In our experiments this estimation predicts the actual score distribution very well. Taking the size of the intersection into account is important, as it encourages intersecting relatively disjoint sets early, which can greatly reduce the output cardinality.

6 Experimentation

6.1 Setup and Test Data

Our implementation of the testbed and the related algorithms was written in C++. All peer related data were stored locally at the peer’s disk. Experiments were performed on 3GHz Pentium machines. For simplicity, all processes ran on the same server. Two real-world data collections were used in our experiments: GOV and IMDB. The GOV collection consists of the data of the TREC-12 Web Track and contains roughly 1.25 million (mostly HTML and PDF) documents obtained from a crawl of the .gov Internet domain (with total index list size of 8 GB). We used 50 queries from the Web Track’s distillation task where each query consists up to 5 terms. In our experiments, the index lists associated with the terms contained the original document scores computed as \( tf \times log idf \). \( tf \) and \( idf \) were normalized by the maximum \( tf \) value of each document and the maximum \( idf \) value in the corpus, respectively. In addition, we employed an extended GOV (XGOV) setup, which we utilized to test the algorithms’ performance on a larger number of query terms and associated index lists. The original 50 queries were expanded by adding new terms from synonyms and glosses taken from the WordNet thesaurus (http://www.cogsci.princeton.edu/~wn/). The expansion resulted in queries with, on average, twice as many terms, with the longest query containing 18 terms. For instance, the GOV query juvenile delinquency has been expanded into juvenile delinquency youth minor crime law jurisdiction offense prevention. The IMDB collection consists of data from the Internet Movie Database (http://www.imdb.com). In total, our test collection contains about 375,000 movies and over 1,200,000 persons (with a total index list size of 140 MB), structured into the object-relational table schema Movies (Title, Genre, Actors, Description). Title and Description are text attributes and Genre and Actors are set-valued attributes. Genre contains 2 or 3 genres. Actors included only those actors that appeared in at least 5 movies. The IMDB queries contain text and structured attributes.

6.2 Competitors

**TPUT**: This is the 3-phase algorithm as described in [CW04]. TPUT comes in two flavors: the original and a version with compression for long docIDs. This optimized version, instead of sending (docID, score) pairs, hashes the docID into a hash array where it stores its score and sends the hash array of scores. Even in the experiments conducted in [CW04]
the compressed optimized version did not always perform better, so we report only the results for the original TPUT version.

**Simple**: This algorithm is essentially the same as TPUT but allows a moving coordinator.

**Optimal**: As one of our key contributions is to show the suitability and significant benefits of optimizing hierarchical top-$k$ algorithms, we implemented a hierarchical version of TPUT. This algorithm essentially consists of multiple TPUT instances organized in a tree like structure.

### 6.3 Metrics

**Network Bandwidth Consumption**: This represents the total number of bytes transferred during query execution. It includes the bytes for the actual data items and the size of the required statistics as described in Section 5.

**Query Response Time**: This represents the elapsed, "wall-clock"time for running the benchmarks. It includes network time, local i/o time, and time for query optimization and local processing. Our optimization is primarily focused on minimizing the query response time but as experiments indicate the overall network traffic will decrease too, since the network traffic is an essential part of the overall response time.

### 6.4 Experiments

We report on experiments performed for each of the benchmarks, GOV, XGOV, IMDB. In all experiments queries are for the top-10 results. Running the experiments over multiple nodes in a network would be inherently vulnerable to interference from other processes running concurrently and competing for cpu cycles, disk arms, and network bandwidth. To avoid this and produce reproducible and comparable results for algorithms ran at different times, we opted for simulating disk IO latency and network latency which are dominant factors. Specifically, each random disk IO was modeled to incur a disk seek and rotational latency of 9 ms, plus a transfer delay dictated by a transfer rate of 8MB/s. For network latency we utilized typical round trip times (RTTs) of packets and transfer rates achieved for larger data transfers between widely distributed entities [SL00]. We assumed a packet size of 1KB with a RTT of 150 ms and used it to measure the latency of communication phases for data transfer sizes in each connection up to 1KB. When cohorts sent more data, the additional latency was dictated by a "large"data transfer rate of 800 Kb/s. This figure is the average throughput value measured (using one stream – one cpu machines) in experiments conducted for measuring wide area network throughput (sending 20MB files between SLAC nodes (Stanford’s Linear Accelerator Centre) and nodes in Lyon France [SL00] using NLANR’s iPerf tool [Tir03]. Hence, the overall response times were the sum of cpu times for an algorithm’s local processing, IO times, and network communication times. Since the execution is running in parallel, each operator in the execution plan has to wait for the slowest of its inputs.

Note that the execution time and network consumption includes the transfer of the required statistical data. While the TPUT algorithm does not require statistics, the other algorithms ship the histograms (or poisson mixes) and MIPs during Phase 1. This overhead is included in the results.

338
6.5 Performance Results

Figure 6(left) shows the average query response time for the GOV benchmark where queries with the same number of query terms were grouped together. For the two-term queries there is almost nothing to gain as there are not many alternatives for the execution plan. However, for the three- and four-term queries we clearly outperform TPUT. Figure 6(right) shows the total network traffic for the GOV benchmark. We clearly beat TPUT for all numbers of query terms; TPUT causes two times more network traffic than the optimized query execution, and even the simple approach shows a pretty strong improvement. Recall at this point that we optimize the query response time and not the network resource consumption, thus we can observe a bigger gain in query response time than in overall network traffic, as can be seen in the Figures.

Figure 6: Query Response Times (left) and Total Network Traffic (right) for the GOV Benchmark

Figure 7(left) shows the average response time for the XGOV benchmark. We clearly outperform TPUT for all number of query terms. Note that the XGOV benchmark does not contain queries with 16 or 17 terms. Figure 7(right) shows the total network traffic for the XGOV benchmark. Although we optimize the queries with respect to the overall response time the total network traffic is also smaller.

Figure 7: Query Response Times (left) and Total Network Traffic (right) for the XGOV Benchmark

Figure 8(left) shows the query response time for the IMDB benchmark. Similar to the GOV benchmark, all algorithms show nearly the same performance for the two keyword
queries. However, for queries with three or more keywords the optimization has remarkable performance gains. For one particular query we gain a factor of around 32 for the optimal plan. The median factor of the performance-improvement is around 5. We observe a similar behavior for the overall network traffic as shown in Figure 8(right).

Figure 8: Query Response Time (left) and Total Network Traffic (right) for the IMDB Benchmark

To compare the usefulness of the Histogram based query optimization with the Two Poisson Model based optimization, Figure 9 show the relative performance gains in query response time of the optimized query plans for these two score distribution models for the XGOV benchmark. Obviously, as expected, the histogram based optimization shows a better performance than the Poisson based approach. As discussed above, Histograms offer a more accurate way to represent an index list’s score distribution, and although Poisson Models are more compact, the overhead for the histograms is nearly negligible too. As mentioned in Section 5, (Two) Poisson Models cannot capture extreme variations in the score distribution that were present in our data. In particular, the IMDB index lists have an extremely skewed score distribution with ties. The scores from the GOV collection were based on \( tf \cdot idf \) that are skewed too. A more smoothed scoring function like BM25 [RW94] would have been better suitable for the Two Poisson Model. However, Poisson Mixes give reasonably good approximations and are interesting as they have extremely small overhead.

Figure 9: Comparison of Histogram and Two Poisson Model based Optimization for the XGOV Benchmark.
7 Conclusion

This paper has addressed efficient algebraic optimization of top-k queries in a distributed environment. We have shown how to deal with score-threshold based operators during query optimization. The proposed algorithm uses compact data synopsis, shipped at query execution time to find the optimal query execution plan for a particular information need. The experimental evaluation shows that the additional resource consumption caused by the query optimization is nearly negligible and that the optimized queries are superior to the standard top-k queries both in terms of network resource consumption and query response time. Future work includes the integration of other distributed top-k algorithms than [CW04] and multi-query optimization across different distributed top-k queries.

References


[YSMQ] Clement Yu, Prasoon Sharma, Weiyi Meng und Yan Qin. Database selection for processing k nearest neighbors queries in distributed environments. In JCDL ’01.

Efficient Reverse k-Nearest Neighbor Estimation

Elke Achtert, Christian Böhm, Peer Kröger, Peter Kunath, Alexey Pryakhin, Matthias Renz

Institute for Computer Science
Ludwig-Maximilians Universität München
Oettingenstr. 67, 80538 Munich, Germany
{achtert,boehm,kroegerp,kunath,pryakhin,renz}@dbs.ifi.lmu.de

Abstract: The reverse $k$-nearest neighbor ($R_k$NN) problem, i.e., finding all objects in a data set the $k$-nearest neighbors of which include a specified query object, has received increasing attention recently. Many industrial and scientific applications call for solutions of the $R_k$NN problem in arbitrary metric spaces where the data objects are not Euclidean and only a metric distance function is given for specifying object similarity. Usually, these applications need a solution for the generalized problem where the value of $k$ is not known in advance and may change from query to query. In addition, many applications require a fast approximate answer of $R_k$NN-queries. For these scenarios, it is important to generate a fast answer with high recall. In this paper, we propose the first approach for efficient approximative $R_k$NN search in arbitrary metric spaces where the value of $k$ is specified at query time. Our approach uses the advantages of existing metric index structures but proposes to use an approximation of the nearest-neighbor-distances in order to prune the search space. We show that our method scales significantly better than existing non-approximative approaches while producing an approximation of the true query result with a high recall.

1 Introduction

A reverse $k$-nearest neighbor ($R_k$NN) query returns the data objects that have the query object in the set of their $k$-nearest neighbors. It is the complementary problem to that of finding the $k$-nearest neighbors ($k$NN) of a query object. The goal of a reverse $k$-nearest neighbor query is to identify the "influence" of a query object on the whole data set. Although the reverse $k$-nearest neighbor problem is the complement of the $k$-nearest neighbor problem, the relationship between $k$NN and $R_k$NN is not symmetric and the number of the reverse $k$-nearest neighbors of a query object is not known in advance. A naive solution of the $R_k$NN problem requires $O(n^2)$ time, as the $k$-nearest neighbors of all of the $n$ objects in the data set have to be found. Obviously, more efficient algorithms are required, and, thus, the $R_k$NN problem has been studied extensively in the past few years (cf. Section 2).

As we will discuss in Section 2 these existing methods for $R_k$NN search can be categorized into two classes, the hypersphere-approaches and the Voronoi-approaches. Usually, it is very difficult to extend Voronoi-approaches in order to apply them to general metric objects. Hypersphere-approaches extend a multidimensional index structure to store each ob-
ject along with its nearest neighbor distance. Thus, although most hypersphere-approaches are only designed for Euclidean vectors, these methods can usually be extended for general metric objects. In principle, the possible performance gain of the search operation is much higher in the hypersphere-approaches while only Voronoi-approaches can be extended to the reverse $k$-nearest neighbor problem with an arbitrary $k > 1$ in a straightforward way. The only existing hypersphere-approach that is flexible w.r.t. the parameter $k$ to some extend is limited by a parameter $k_{max}$ which is an upper bound for the possible values of $k$. All these recent methods provide an exact solution for the $R_k$NN problem. However, in many applications, an approximate answer for $R_k$NN queries is sufficient especially if the approximate answer is generated faster than the exact one. Those applications usually need a solution for general metric objects rather than a solution limited to Euclidean vector data and, additionally, for handling $R_k$NN queries for any value of $k$ which is only known at query time.

One such sample application is a pizza company that wants to evaluate a suitable location for a new restaurant. For this evaluation, a $R_k$NN query on a database of residents in the target district could select the set of residents that would have the new restaurant as its nearest pizza restaurant, i.e. are potential customers of the new restaurant. In addition, to keep down costs when carrying out an advertising campaign, it would be profitable for a restaurant owner to send menu cards only to those customers which have his restaurant as one of the $k$-nearest pizza restaurant. In both cases, an approximate answer to the $R_k$NN query is sufficient. Usually, the database objects in such an application are nodes in a traffic network (cf. Figure 1). Instead of the Euclidean distance, the network distance computed by graph algorithms like Dijkstra is used.

Another important application area of $R_k$NN search in general metric databases is molecular biology. Researchers all over the world rapidly detect new biological sequences that need to be tested on originality and interestingness. When a new sequence is detected, $R_k$NN queries are applied to large sequence databases storing sequences of biological molecules with known function. To decide about the originality of a newly detected sequence, the $R_k$NNs of this sequence are computed and examined. Again, an approximate answer of the launched $R_k$NN queries is sufficient. In addition, it is much more important to get quick results in order to enable interactive analysis of possible interesting sequences. Usually, in this context, the similarity of biological sequences is defined in terms of a metric distance function such as the Edit distance or the Levenstein distance. More details on this application of $R_k$NN search in metric databases can be found in [DP03].

In general, the $R_k$NN problem appears in many practical situations such as geographic information systems (GIS), traffic networks, adventure games, or molecular biology where the database objects are general metric objects rather than Euclidean vectors. In these application areas, $R_k$NN queries are frequently launched where the parameter $k$ can change from query to query and is not known beforehand. In addition, in many applications, the efficiency of the query execution is much more important than effectiveness, i.e. users want a fast response to their query and will even accept approximate results (as far as the number of false drops and false hits is not too high).

In this paper, we propose an efficient approximate solution based on the hypersphere-approach for the $R_k$NN problem. Our solution is designed for general metric objects and
allows R$k$NN queries for arbitrary $k$. In contrast to the only existing approach, the parameter $k$ is not limited by a given upper bounding parameter $k_{\text{max}}$. The idea is to use a suitable approximation of the $k$NN distances for each $k$ of every object in order to evaluate database objects as true hits or true drops without requiring a separate $k$NN search. This way, we approximate the $k$NN distances of a single object stored in the database as well as the $k$NN distances of the set of all objects stored in a given subtree of our metric index structure. To ensure a high recall of our result set we need an approximation of the $k$NN distances with minimal approximation error (in a least square sense). We will demonstrate in Section 3 that the $k$-nearest neighbor distances follow a power law which can be exploited to efficiently determine such approximations. Our solution requires a negligible storage overhead of only two additional floating point values per approximated object. The resulting index structure called AMR$k$NN (Approximate Metric R$k$NN)-Tree can be based on any hierarchically organized, tree-like index structure for metric spaces. In addition, it can also be used for Euclidean data by using a hierarchically organized, tree-like index structure for Euclidean data.

The remainder of this paper is organized as follows: Section 2 introduces preliminary definitions, discusses related work, and points out our contributions. In Section 3 we introduce our novel AMR$k$NN-Tree in detail. Section 4 contains a comparative experimental evaluation. Section 5 concludes the paper.

2 Survey

2.1 Problem Definition

Since we focus on the traditional reverse $k$-nearest neighbor problem, we do not consider recent approaches for related or specialized reverse nearest neighbor tasks such as the
bichromatic case, mobile objects, etc.

In the following, we assume that $\mathcal{D}$ is a database of $n$ metric objects, $k \leq n$, and $\text{dist}$ is a metric distance function on the objects in $\mathcal{D}$. The set of $k$-nearest neighbors of an object $q$ is the smallest set $\mathcal{N}_k(q) \subseteq \mathcal{D}$ that contains at least $k$ objects from $\mathcal{D}$ such that
\[ \forall o \in \mathcal{N}_k(q), \forall \hat{o} \in \mathcal{D} - \mathcal{N}_k(q) : \text{dist}(q, o) < \text{dist}(q, \hat{o}). \]

The object $p \in \mathcal{N}_k(q)$ with the highest distance to $q$ is called the $k$-nearest neighbor ($k$NN) of $q$. The distance $\text{dist}(q, p)$ is called $k$-nearest neighbor distance ($k$NN distance) of $q$, denoted by $\text{nndist}_k(q)$.

The naive solution to compute the reverse $k$-nearest neighbor of a query object $q$ is rather expensive. For each object $p \in \mathcal{D}$, the $k$-nearest neighbors of $p$ are computed. If the $k$-nearest neighbor list of $p$ contains the query object $q$, i.e., $q \in \mathcal{N}_k(p)$, object $p$ is a reverse $k$-nearest neighbor of $q$. The runtime complexity of one query is $O(n^2)$. It can be reduced to an average of $O(n \log n)$ if an index such as the M-Tree [CPZ97] (or, if the objects are feature vectors, the R-Tree [Gut84] or the R*-Tree [BKSS90]) is used to speed-up the nearest neighbor queries.

### 2.2 Related Work

An approximative approach for reverse $k$-nearest neighbor search in higher dimensional space is presented in [SFT03]. A two-way filter approach is used to generate the results. Recently, in [XLOH05] two methods for estimating the $k$NN-distance from one known $\kappa$NN-distance are presented. However, both methods are only applicable to Euclidean vector data, i.e., $D \subseteq \mathbb{R}^d$.

All other approaches for the R$k$NN search are exact methods that usually produce considerably higher runtimes. Recent approaches can be classified as Voronoi-approaches or hypersphere-approaches.

**Voronoi-approaches** usually use the concept of Voronoi cells to prune the search space. The above-mentioned, approximate solution proposed in [SFT03] can be classified as Voronoi-based approach. In [SAA00], a Voronoi-based approach for reverse 1-nearest neighbor search in a 2D data set is presented. It is based on a partition of the data space into six equi-sized units where the gages of the units cut at the query object $q$. The nearest neighbors of $q$ in each unit are determined and merged together to generate a candidate set. This considerably reduces the cost for the nearest-neighbor queries. The candidates are then refined by computing for each candidate $c$ the nearest neighbor. Since the number of units in which the candidates are generated increases exponentially with $d$, this approach is only applicable for 2D data sets. Recently, in [TPL04] the first approach for R$k$NN search was proposed, that can handle arbitrary values of $k$. The method uses any hierarchical tree-based index structure such as R-Trees to compute a nearest neighbor ranking.
of the query object $q$. The key idea is to iteratively construct a Voronoi cell around $q$ from the ranking. Objects that are beyond $k$ Voronoi planes w.r.t. $q$ can be pruned and need not to be considered for Voronoi construction. The remaining objects must be refined, i.e. for each of these candidates, a $k$NN query must be launched. In general, Voronoi-based approaches can only be applied to Euclidean vector data because the concept of Voronoi cells does not exist in general metric spaces.

**Hypersphere-approaches** use the observation that if the distance of an object $p$ to the query $q$ is smaller than the 1-nearest neighbor distance of $p$, $p$ can be added to the result set. In [KM00] an index structure called RNN-Tree is proposed for reverse 1-nearest neighbor search based on this observation. The RNN-Tree precomputes for each object $p$ the distance to its 1-nearest neighbor, i.e. $nndist_1(p)$. The objects are not stored in the index itself. Rather, for each object $p$, the RNN-Tree manages a sphere with radius $nndist_1(p)$, i.e. the data nodes of the tree contain spheres around objects. The RdNN-Tree [YL01] extends the RNN-Tree by storing the objects of the database itself rather than circles around them. For each object $p$, the distance to $p$’s 1-nearest neighbor, i.e. $nndist_1(p)$ is aggregated. In general, the RdNN-Tree is a R-Tree-like structure containing data objects in the data nodes and MBRs in the directory nodes. In addition, for each data node $N$, the maximum of the 1-nearest neighbor distance of the objects in $N$ is aggregated. An inner node of the RdNN-Tree aggregates the maximum 1-nearest neighbor distance of all its child nodes. In general, a reverse 1-nearest neighbor query is processed top down by pruning those nodes $N$ where the maximum 1-nearest neighbor distance of $N$ is greater than the distance between query object $q$ and $N$, because in this case, $N$ cannot contain true hits anymore. Due to the materialization of the 1-nearest neighbor distance of all data objects, the RdNN-Tree needs not to compute 1-nearest neighbor queries for each object. Both, the RNN-Tree and the RdNN-Tree, can be extended to metric spaces (e.g. by applying an M-Tree [CPZ97] instead of an R-Tree). However, since the $k$NN distance needs to be materialized, it is limited to a fixed $k$ and cannot be generalized to answer $Rk$NN-queries with arbitrary $k$. To overcome this problem, the MR$k$NNCoP-Tree [ABK+06b] has been proposed recently. The index is conceptually similar to the RdNN-Tree but stores a conservative and progressive approximation for all $k$NN distances of any data object rather than the exact $k$NN distance for one fixed $k$. The only limitation is that $k$ is upper-bounded by a parameter $k_{\text{max}}$. For $Rk$NN queries with $k > k_{\text{max}}$, the MR$k$NNCoP-Tree cannot be applied [ABK+06a]. The conservative and progressive approximations of any index node are propagated to the parent nodes. Using these approximations, the MR$k$NNCoP-Tree can identify a candidate set, true hits, and true drops. For each object in the candidate set, a $k$NN query need to be launched for refinement.

### 2.3 Contributions

Our solution is conceptually similar to that in [ABK+06b] but extends this work and all other existing approaches in several important aspects. In particular, our method provides the following new features:
1. Our solution is applicable for R$k$NN search using any value of $k$ because our approximation can be interpolated for any $k \in \mathbb{N}$. In contrast, most previous methods are limited to R$k$NN queries with one predefined, fixed $k$ or $k \leq k_{max}$.

2. Our distance approximation is much smaller than the approximations proposed in recent approaches and, thus, produces considerably less storage overhead. As a consequence, our method leads to a smaller index directory resulting in significantly lower query execution times.

3. In contrast to several existing approaches, our method does not need to perform $k$NN queries in an additional refinement step. This also dramatically reduces query execution times.

4. Our distance approximations can be generated from a small sample of $k$NN distances (the $k$NN distances of any $k \in \mathbb{N}$ can be interpolated from these approximations). Thus, the time for index creation is dramatically reduced.

In summary, our solution is the first approach that can answer R$k$NN queries for any $k \in \mathbb{N}$ in general metric databases. Since our solution provides superior performance but approximate results, it is applicable whenever efficiency is more important than complete results. However, we will see in the experimental evaluation that the loss of accuracy is negligible.

3 Approximate Metric R$k$NN Search

As discussed above, the only existing approach to R$k$NN search that can handle arbitrary values of $k$ at query time and can be used for any metric objects (not only for Euclidean feature vectors) is the MR$k$NNCoP-Tree [ABK+06b] that extends the RdNN-tree by using conservative and progressive approximations for the $k$NN distances. This approach, however, is optimized for exact R$k$NN search and is limited to a its flexibility regarding the parameter $k$ is limited by an additional parameter $k_{max}$. This additional parameter must be specified in advance, and is an upper bound for the value of $k$ at query time. If a query is launched specifying a $k > k_{max}$, the MR$k$NNCoP-Tree cannot guarantee complete results. In our scenario of answering approximate R$k$NN queries, this is no problem. However, since the MR$k$NNCoP-Tree constraints itself to compute exact results for any query with $k \leq k_{max}$ it generates unnecessary overhead by managing conservative and progressive approximations. In general, an index for approximate R$k$NN search does not need to manage conservative and progressive approximations of the $k$NN distances of each object but only needs one approximation.

Thus, for each object, instead of two approximations (a conservative and a progressive) of the $k$NN distances which is bound by a parameter $k_{max}$, we store one approximation of the $k$NN distances for any $k \in \mathbb{N}$. This approximation is represented by a function, i.e. the approximated $k$NN distance for any value $k \in \mathbb{N}$ can be calculated by applying this function. Similar to existing approaches, we can use an extended M-Tree, that aggregates for
each node the one approximation of the approximations of all child nodes or data objects contained in that node. These approximations are again represented as functions. At runtime, we can estimate the \( k \)NN distance for each node using this approximation in order to prune nodes analogously to the way we can prune objects. Since the approximation does not ensure completeness, the results may contain false positives and may miss some true drops. As discussed above, this is no problem since we are interested in an approximate \( R \)NN search scenario.

In the following, we introduce how to compute an approximation of the \( k \)NN distances for arbitrary \( k \in \mathbb{N} \). After that, we describe how this approximation can be integrated into an M-Tree. At the end of this section, we outline our approximate \( Rk \)NN search algorithm.

### 3.1 Approximating the \( k \)NN Distances

A suitable model function for the approximation of our \( k \)NN distances for every \( k \in \mathbb{N} \) should obviously be as compact as possible in order to avoid a high storage overhead and, thus, a high index directory.

In our case, we can assume that the distances of the neighbors of an object \( o \) are given as a (finite) sequence

\[
NNdist(o) = (\text{ndist}_1(o), \text{ndist}_2(o), \ldots, \text{ndist}_{k_{\text{max}}}(o))
\]

for any \( k_{\text{max}} \in \mathbb{N} \) and this sequence is ordered by increasing \( k \). Due to monotonicity, we also know that \( i < j \Rightarrow \text{ndist}_i(o) \leq \text{ndist}_j(o) \). Our task here is to describe the discrete sequence of values by some function \( f_o : \mathbb{N} \rightarrow \mathbb{R} \) with \( f_o(k) \approx \text{ndist}_k(o) \).

As discussed above, such a function should allow us to calculate an approximation of the \( k \)NN distance for any \( k \), even for \( k > k_{\text{max}} \) by estimating the corresponding values.

From the theory of self-similarity [Sch91] it is well-known that in most data sets the relationship between the number of objects enclosed in an arbitrary hypersphere and the scaling factor (radius) of the hypersphere (the same is valid for other solids such as hypercubes) approximately follows a power law:

\[
encl(\varepsilon) \propto \varepsilon^{d_f},
\]

where \( \varepsilon \) is the scaling factor, \( encl(\varepsilon) \) is the number of enclosed objects and \( d_f \) is the fractal dimension. The fractal dimension is often (but not here) assumed to be a constant which characterizes a given data set. Our \( k \)NN sphere around any object \( o \in D \) can be understood to be such a scaled hypersphere where the distance of the \( k \)NN is the scaling factor and \( k \) is the number of enclosed objects. Thus, it can be assumed that the \( k \)NN distances also follow the power law, i.e.

\[
k \propto \text{ndist}_k(o)^{d_f}.
\]

Transferred in log-log space (for an arbitrary logarithmic basis, e.g. for basis \( e \)), we have
a linear relationship [Sch91]:

$$\log(nndist_k(o)) \propto \frac{1}{df} \cdot \log(k).$$

This linear relationship between $k$ and the $k$NN distance in log-log space is illustrated for different sample data distributions and a sample 2D real-world data set\(^1\) in Figure 2. Obviously this linear relationship is not perfect. However, as it can be anticipated from Figure 2, the relationship between $\log(k)$ and $\log(nndist_k(o))$ for any object $o$ in a database of arbitrary distribution, exhibit a clear linear tendency.

From this observation, it follows that it is generally sensible to use a model function which is linear in log-log space — corresponding to a parabola in non-logarithmic space — for the approximation. Obviously, computing and storing a linear function needs considerably less overhead than a higher order function. Since we focus in this section on the approximation of the values of the $k$NN distance over varying $k$ in a log-log sense, we consider

---

\(^1\)The real-world data represents the spatial coordinates of landmarks in Sacramento, CA. The data originates from: http://www.census.gov
the pairs \((\log(k), \log(nndist_k(o)))\) as points of a two-dimensional vector space \((x_k, y_k)\). These points are not to be confused with the objects stored in the database (e.g. the object \(o\) the nearest neighbors of which are considered here) which are general metric objects. Whenever we speak of points \((x, y)\) or lines \(((x_1, y_1), (x_2, y_2))\) we mean points in the two-dimensional log-log space where \(\log(k)\) is plotted along the x-axis and \(\log(nndist_k(o))\) for a given general metric object \(o \in D\) is plotted along the y-axis.

Like in most other applications of the theory of self-similarity, we need to determine a classical regression line that approximates the true values of \(nndist_k(o)\) with least square error. A conventional regression line \(f_o(x) = m_o \cdot x + t_o\) would find the parameters \((m_o, t_o)\) minimizing least square error:

\[
\sum_{k=1}^{k_{max}} (y_k - (m_o \cdot \log k + t_o))^2 \rightarrow \min
\]

where \(y_k = \log nndist_k(o)\), which evaluates the well known formula of a regression line in 2D space. As indicated above, since this line is the best approximation of a point set, it is exactly the approximation of the \(k\)NN distances we want to aggregate. In other words, for each object \(o \in D\), we want to calculate the function \(f_o(x) = m_o \cdot x + t_o\) that describes the regression line of the point set \(\{(\log k, \log nndist_k(o)) | 1 \leq k \leq k_{max}\}\).

From the theory of linear regression, the parameters \(m_o\) and \(t_o\) can be determined as

\[
m_o = \frac{\left(\sum_{k=1}^{k_{max}} y_k \cdot \log k\right) - k_{max} \cdot \bar{y} \cdot \frac{1}{k_{max}} \sum_{k=1}^{k_{max}} \log k}{\left(\sum_{k=1}^{k_{max}} (\log k)^2\right) - k_{max} \cdot \left(\frac{1}{k_{max}} \sum_{k=1}^{k_{max}} \log k\right)^2}
\]

where \(\bar{y} = \frac{1}{k_{max}} \sum_{k=1}^{k_{max}} \log nndist_k(o)\), and

\[
t_o = \bar{y} - m_o \cdot \frac{1}{k_{max}} \sum_{k=1}^{k_{max}} \log k.
\]

### 3.2 Aggregating the Approximations

So far, we have shown how to generate an accurate approximation for each object of the database. When using a hierarchically organized index structure, the approximation can also be used for the nodes of the index to prune irrelevant sub-trees. Usually, each node \(N\) of the index is associated with a page region representing a set of objects in the subtree which has \(N\) as root. In order to prune the subtree of node \(N\), we need to approximate the \(k\)NN distances of all objects in this subtree, i.e. page region. If the distance between the query object \(q\) and the page region of \(N\), called MINDIST, is larger than this approximation, we can prune \(N\) and thus, all objects in the subtree of \(N\). The MINDIST is a lower
bound for the distance of \( q \) to any of the objects in \( N \). The aggregated approximation should again estimate the \( k \)NN distances of all objects in the subtree representing \( N \) with least squared error. This is a little more complex than a simple regression problem.

Obviously, given a data node \( N \) with \(|N|\) data objects \( o_i \in N \), the parameters of the optimal regression line \( F_N(x) = m_N \cdot x + t_N \) that approximates the \( k \)NN distances of all objects in \( N \) can be determined as follows:

\[
    m_N = \frac{\sum_{o_i \in N} (\sum_{k=1}^{k_{\max}} y_k^{o_i} \cdot \log k) - \frac{1}{k_{\max}} \cdot \sum_{o_i \in N} \bar{y}^{o_i} \cdot \frac{|N|}{k_{\max}} \sum_{k=1}^{k_{\max}} \log k}{|N| \cdot (\sum_{k=1}^{k_{\max}} (\log k)^2) - k_{\max} \cdot (\frac{1}{k_{\max}} \sum_{k=1}^{k_{\max}} \log k)^2}
\]

and

\[
    t_N = \frac{1}{|N|} \sum_{o_i \in N} \bar{y}^{o_i} - m_o \cdot \frac{1}{k_{\max}} \sum_{k=1}^{k_{\max}} \log k,
\]

where \( y_k^{o_i} = \log nndist_k(o_i) \) and

\[
    \bar{y}^{o_i} = \frac{1}{k_{\max}} \sum_{k=1}^{k_{\max}} \log nndist_k(o_i).
\]
The first equation can be reformulated as

\[ m_N = \sum_{o_i \in N} (\sum_{k=1}^{k_{\text{max}}} y_{o_i}^k \cdot \log k) - \sum_{o_i \in \bar{N}} y_{o_i}^k \cdot \sum_{k=1}^{k_{\text{max}}} \log k \]

\[ |N| \cdot (\sum_{k=1}^{k_{\text{max}}} (\log k)^2) - \frac{1}{k_{\text{max}}} (\sum_{k=1}^{k_{\text{max}}} \log k)^2 \]

Thus, in order to generate an optimal approximation \( f_N \) for any directory node \( N \) with child nodes \( C_i \), we need to aggregate \( \sum_{o_i \in C_i} \sum_{k=1}^{k_{\text{max}}} y_{o_i}^k \) and \( \sum_{o_i \in \bar{C}_i} \) for each \( C_i \). Thus, we store for each child nodes \( C_i \) two additional values

\[ v_1 = \sum_{o_i \in C_i} \sum_{k=1}^{k_{\text{max}}} y_{o_i}^k \]

and

\[ v_2 = \sum_{o_i \in \bar{C}_i} \]

in order to compute the distance approximation of the parent node \( N \). Obviously, the required storage overhead is negligible. On the other hand, we can now generate for each node \( N \) in the tree the optimal regression line for the \( k \)NN distances of all objects located in the subtree of \( N \).

The idea of aggregating the \( k \)NN distance approximations for directory nodes is visualized in Figure 3. The approximation \( f_N \) of a node \( N \) representing objects \( p_1, p_2, p_3 \) is depicted. The regression line \( f_N \) approximates the \( k \)NN distances of \( p_1, p_2, p_3 \) with least square error.

We call the resulting index structure AMR\( k \)NN-Tree (Approximate Metric Reverse \( k \)NN-Tree). The original concepts of the AMR\( k \)NN-Tree presented here can be incorporated within any hierarchically organized index for metric objects. Obviously, our concepts can also be used for \( R_kNN \) search in Euclidean data by integrating the approximation into Euclidean index structures such as the R-tree [Gut84], the R*-tree [BKSS90], or the X-tree [BKK96].

### 3.3 \( R_kNN \) Search Algorithm

The algorithm for approximate \( R_kNN \) queries on our novel AMR\( k \)NN-Tree is similar to the exact \( R_kNN \) query algorithms of the RdNN-Tree and the MR\( k \)NNCoP-Tree. However, our index structure can answer \( R_kNN \) queries for any \( k \) specified at query time. Let us point out that the value of \( k \) is not bound by a predefined \( k_{\text{max}} \) parameter, although the approximation of the \( k \)NN distances are computed by using only the first \( k_{\text{max}} \) values, i.e. the \( k \)NN distances with \( 1 \leq k \leq k_{\text{max}} \). The \( k \)NN distance for any \( k > k_{\text{max}} \) can be extrapolated by our approximations in the same way as for any \( k \leq k_{\text{max}} \). In addition, due
Approximate_RkNN_query(D, q, k)

// D is assumed to be organized as AMRkNN-Tree
queue := new Queue;
insert root of AMRkNN-Tree into queue;
while not queue.isEmpty()
    N := queue.getFirst();
    if N is node then
        if MINDIST(N, q) \leq mN \cdot \log k + tN then
            insert all elements of N into queue;
        end if
    else // N is a point
        if log(dist(N, q)) \leq mN \cdot \log k + tN then
            add N to result set;
        end if
    end if
end while

Figure 4: Algorithm for approximate RkNN query.

to the use of a metric index structure, our AMRkNN-Tree is applicable to general metric objects.

Similar to the M-Tree concept, a node N of our AMRkNN-Tree is represented by its routing object \( N_o \) and the covering radius \( N_r \). All objects represented by node \( N \) have a distance less than \( N_r \) to \( N_o \). The logarithm of the aggregated kNN distance of a node \( N \), denoted by \( kNN_{agg}(N) \), can be determined from the approximation \( f_N(x) = m_N \cdot x + t_N \) of \( N \) by

\[
kNN_{agg}(N) = m_N \cdot \log k + t_N.
\]

Note that the true (i.e. non-logarithmic) approximation of the aggregated kNN distance of \( N \) is \( e^{kNN_{agg}(N)} \). To avoid unnecessary complex computations, we adapt the definition of the MINDIST between a node and a point to the logarithmic scale of \( kNN_{agg}(N) \). Thus, the MINDIST of a node \( N \) and a query point \( q \), denoted by MINDIST\((N, q)\), is defined as

\[
MINDIST(N, q) = \log(\max\{dist(q, N_o) - N_r, 0\}).
\]

The pseudo code of the approximate RkNN query algorithm is depicted in Figure 4. A query \( q \) is processed by traversing the index from the root of the index to the leaf level. A node \( N \) needs to be refined if the MINDIST between \( q \) and \( N \) is smaller than the aggregated kNN distance approximation of \( N \), i.e. MINDIST\((q, N) \leq kNN_{agg}(N) \). Those nodes, where the MINDIST to \( q \) is larger than their aggregated kNN distance approximation are pruned, i.e. if MINDIST\((N, q) > kNN_{agg}(N) \).

The traversal ends up at a data node. Then, all points \( p \) inside this node are tested using their approximation \( f_p(x) = m_p \cdot x + t_p \). A point \( p \) is a hit if

\[
\log(\text{dist}(N, q)) \leq m_N \cdot \log k + t_N.
\]
Otherwise, if \( \log(\text{dist}(N, q)) > m_N \cdot \log k + t_N \), point \( p \) is a miss and should be discarded.

In contrast to other approaches that are designed for \( R_k \)NN search for any \( k \), our algorithm directly determines the results. In particular, we do not need to apply an expensive refinement step to a set of candidates. This further avoids a significant amount of execution time.

## 4 Evaluation

All experiments have been performed on Windows workstations with a 32-bit 4 GHz CPU and 2 GB main memory. We used a disk with a transfer rate of 50 MB/s, a seek time of 6 ms and a latency delay of 2 ms. In each experiment we applied 100 randomly selected \( R_k \)NN queries to the particular dataset and reported the average results. The runtime is presented in terms of the elapsed query time including I/O and CPU-time. All evaluated methods have been implemented in Java.

We compared our AMR\( k \)NN-Tree with the index proposed in [ABK+06b] that is designed for exact \( R_k \)NN search in general metric spaces for any \( k \leq k_{\max} \) and the sequential scan. The approach in [ABK+06b] claims to outperform all other approaches on general metric data as well as on Euclidean data. We will show, that our AMR\( k \)NN-Tree is much more efficient than this state-of-the-art approach on both general metric data and Euclidean data.

### 4.1 Datasets

**Metric \( R_k \)NN search.** Our experiments were performed using two real-world datasets. The first one is a road network dataset derived from the city of San Juan, CA, which contains 18,236 nodes and 23,874 edges. The average degree of the nodes in this network is 2.61. The dataset is online available\(^2\). The nodes of the network graph were taken as database objects from which subsets of different size were selected to form the test data set. For the distance computation we used the shortest-path distance computed by means of the Dijkstra algorithm. The second dataset consists of 10,000 protein sequences taken from SWISSPROT database\(^3\), the Levenstein distance was used as similarity distance. For

\(^2\)http://www.fh-oov.de/institute/iapg/personen/brinkhoff/generator/
\(^3\)http://www.expasy.org/sprot/
both datasets we used an M-Tree with a node size of 4 KByte.

**Euclidean R\textsuperscript{\kappa}NN search.** We also integrated our concepts into an X-Tree [BKK96] in order to support R\textsuperscript{\kappa}NN search in Euclidean data. We used three real-world datasets for our experiments including a set of 5-dimensional vectors generated from the well-known SEQUOIA 2000 benchmark dataset and two ”Corel Image Features” benchmark datasets from the UCI KDD Archive\textsuperscript{4}. The first Corel Image dataset contains 9 values for each image (”ColorMoments”), the second Corel Image dataset contains 16-dimensional texture values (”CoocTexture”). The underlying X-Tree had a node size of 4 KByte.

The characteristics of the real-world datasets used for our evaluation are summarized in Table 1.

### 4.2 Comparison to competing approaches in Euclidean space

In Euclidean space, there exist two competitors PDE and kDE [XLOH05] as discussed in Section 2.2. In an initial setup, we compare the performance of our approach to both competing approaches by measuring the average \(k\)NN-distance error. For all experiments, we set \(k_{\text{max}} = 100\). The \(\kappa\) parameter for the competing techniques was set to 50. Figure 5(a-c) depicts the error for varying parameter \(k\). Because PDE and kDE store the exact distance for \(k = \kappa\), the error for both techniques decreases when \(k\) converges to \(\kappa\). For \(k \neq \kappa\), the distance approximations of PDE and kDE are significantly worse than those of our approach. For the 16-dimensional Corel Image dataset, our AMR\(k\)NN approach outperforms the competing techniques by a factor between 4 and 6, for \(k \leq 30\) resp. \(k \geq 70\). In a next experiment, we evaluated the error for varying database size, as depicted in Figure 5(d). The results show that the quality of the distance approximations for all three techniques is almost independent from the database size, i.e. is not affected by the density of the dataset.

Because the quality of the distance approximations of the AMR\(k\)NN-Tree clearly outperforms the distance approximations of PDE and kDE for varying parameter \(k\) and varying database size, we do not take PDE and the kDE into account in the remaining experiments.

### 4.3 Runtime w.r.t. database size

We altered the number of database objects in order to evaluate the scalability of the competing methods w.r.t. the database size. Throughout all experiments, we set \(k = 50\) and \(k_{\text{max}} = 100\).

**Metric R\textsuperscript{\kappa}NN search.** A comparison of our novel index structure with the state-of-the-art approach applied to our real-world metric datasets is shown in Figure 6. It can be seen that our AMR\(k\)NN-Tree clearly outperforms the competing MR\(k\)NNCoP-Tree on the road

\textsuperscript{4}http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html
network dataset (cf. Figure 6(a)). The performance gain of our approach over the existing method also grows with increasing database size. Both approaches show a linear scalability w.r.t. the number of data objects, but the increase of runtime of our AMR$k$NN-Tree is smaller than the increase of runtime of the MR$k$NNCoP-Tree. The runtime of the sequential scan also grows linear with increasing number of database objects. It is not shown in Figure 6(a) for clearness reasons. In fact, we observed that the performance gain of our AMR$k$NN-Tree over the sequential scan grows with increasing database size from a factor of 150 to about 850.

A similar observation can be made on the dataset containing biological sequences. The results are illustrated in Figure 6(b). Again, the sequential scan is not shown due to clarity reasons.

Euclidean R$k$NN search. In Figure 7 a comparison of our novel index structure with the state-of-the-art approach applied to our real-world Euclidean datasets is presented. As it can be observed, our AMR$k$NN-Tree clearly outperforms the competing MR$k$NNCoP-Tree on all three datasets. In addition, the performance gain of our approach over the existing method also grows with increasing database size on all datasets. Both competing
approaches show a linear scalability w.r.t. the number of data objects, but the increase of runtime of our AMR$k$NN-Tree is significantly smaller than the increase of runtime of the MR$k$NNCoP-Tree. The superiority of our AMR$k$NN-Tree is even more obvious on Euclidean data. The runtime of the sequential scan is also not shown in the charts presented in Figure 7 for clearness reasons. In fact, the sequential scan is outperformed by both methods by a factor of clearly over 100.

4.4 Runtime w.r.t. parameter $k$

We executed R$k$NN queries on a database with varying $k$ and compared the scalability of both competing methods with the sequential scan. The parameter $k_{\text{max}}$ was set to 100 for both approaches in all experiments.

**Metric R$k$NN search.** The results of these experiments on the metric datasets are depicted in Figure 8. Applied to the road network dataset with 10,000 nodes, our novel AMR$k$NN-Tree clearly outperforms the current state-of-the-art approach (cf. Figure 8(a)). With increasing $k$, the performance gain of our method over the competitor further grows. The runtime of the sequential scan is independent of the choice of $k$ and was observed at 140 seconds per query for any $k$. It is not shown in Figure 8(a) for clearness reasons.

A similar observation can be made when applying the competing methods to the dataset of 10,000 biological sequences. The results are illustrated in Figure 8(b). For clarity reasons, the runtime of the sequential scan (approx. 100 seconds) is again not shown. It can be observed that with increasing $k$, the performance gain of our method over the competitor is even stronger rising.

**Euclidean R$k$NN search.** The results of these experiments on the Euclidean datasets are depicted in Figure 9. All three datasets contained 50,000 objects. Applied to the SEQUOIA data, it can be seen that our approach scales linear with a very low slope. On the
other hand, the MR$k$NNCoP-Tree exhibits a stronger rise of runtime. Similar observations can be made on the Corel Image datasets (cf. Figure 9(b) and Figure 9(c)). In summary, in almost all parameter settings, our novel AMR$k$NN-Tree is at least 4 times faster than the MR$k$NNCoP-Tree. The sequential scan scales constant for any value of $k$. The reported runtimes on the three Euclidean datasets of this naive solution are between 450 and 500 seconds. Those runtimes are not shown in Figure 9(a), Figure 9(b), and Figure 9(c) for clearness reasons.

4.5 Effectivness

The two probably most widespread concepts for measuring the effectiveness are the recall and the precision. The recall measures the relative number of true hits reported as result, whereas precision measures the relative number of reported objects that are true hits. Usually, a user does not care so much about false positives, i.e. objects reported as hits that
are true drops, as far as no true hits are missing. Thus, for measuring the quality of our approximate results, we focused on the recall. This measurement is the most important measurement to judge the quality of approximate results.

**Metric R\(k\)NN search.** We evaluated the effectiveness of our approximate R\(k\)NN search on our metric datasets. In this experiment, we set \(k_{max} = 100\) and executed several R\(k\)NN queries for \(10 \leq k \leq 200\). The results are depicted in Figure 10(a). As it can be seen, in almost all experiments, the recall is clearly above 90%. On the sequence dataset, the recall falls below 80% for low \(k\) values but rises significantly over 90% at about \(k = 60\). This very accurate effectiveness is complemented by a rather high precision of the reported queries (between 80 - 97 %). It is worth mentioning, that the recall does not decrease significantly when answering R\(k\)NN queries with \(k > k_{max}\). This observation confirms the claim that our AMR\(k\)NN-Tree is applicable to any \(k \in \mathbb{N}\).

**Euclidean R\(k\)NN search.** A similar observation can be made when evaluating the recall of our method on the Euclidean datasets. Again we set \(k_{max} = 100\) and executed several R\(k\)NN queries for \(10 \leq k \leq 200\). The results are depicted in Figure 10(b). As it can be seen, for most parameter settings, the recall is clearly above 90%. Again we observed a rather high precision (between 80 - 98 %). We also want to point out that the recall does not decrease significantly when answering R\(k\)NN queries with \(k > k_{max}\). Once again, this observation confirms the claim that our AMR\(k\)NN-Tree is applicable to any \(k \in \mathbb{N}\).

## 5 Conclusions

In this paper, we proposed the first solution for approximate R\(k\)NN search in general metric spaces for any \(k \in \mathbb{N}\). Our approach is based on the observation known from the theory of self-similarity that the relationship between \(k\) and the \(k\)NN distance of any object is linear in log-log space. We proposed to calculate an approximation of the \(k\)NN
distances of any database object by means of a regression line in the log-log space from a set of sample $k$NN distances. The $k$NN distance of any $k$ can then be interpolated from this approximation. We showed how these approximations can be integrated into any hierarchically organized index structure (e.g. the M-Tree for metric objects or the R-Tree for Euclidean vectors) by propagating the approximations of child nodes into parent nodes. Our resulting index called AMR$k$NN-Tree has achieved significant performance boosts compared to existing approaches. In addition, our experiments showed that our performance gain caused only a negligible loss in accuracy.

For future work, we will examine parallel and distributed solutions to the R$k$NN problem.

References

Figure 10: Recall of our method on real-world datasets.


On Deriving Net Change Information From Change Logs
– The DELTALAYER-Algorithm –

Stefanie Rinderle¹, Martin Jurisch¹, Manfred Reichert²
¹Institute DBIS, Ulm University, Germany, {stefanie.rinderle, martin.jurisch}@uni-ulm.de
²IS Group, University of Twente, The Netherlands, m.u.reichert@utwente.nl

Abstract: The management of change logs is crucial in different areas of information systems like data replication, data warehousing, and process management. One barrier that hampers the (intelligent) use of respective change logs is the possibly large amount of unnecessary and redundant data provided by them. In particular, change logs often contain information about changes which actually have had no effect on the original data source (e.g., due to subsequently applied, overriding change operations). Typically, such inflated logs lead to difficulties with respect to system performance, data quality or change comparability. In order to deal with this we introduce the DeltaLayer algorithm. It takes arbitrary change log information as input and produces a cleaned output which only contains the net change effects; i.e., the produced log only contains information about those changes which actually have had an effect on the original source. We formally prove the minimality of our algorithm, and we show how it can be applied in different domains; e.g., the post-processing of differential snapshots in data warehouses or the analysis of conflicting changes in process management systems. Altogether the ability to purge change logs from unnecessary information provides the basis for a more intelligent handling of these logs.

1 Introduction

1.1 Problem Description

The management of log information is crucial in different areas of information systems. In addition to, for example, transaction logs in database management systems the optimized processing of change (log) information (e.g., differential snapshots [LGM96]) is gaining more and more importance. In this context, one prominent example is the update of a data warehouse based on change information. More precisely, for such an update heterogeneous data sources are monitored. According to the particular monitoring strategy, updates within the data sources are then fed into the staging area of the data warehouse (cf. Figure 1a). How these data updates can be determined depends on the kind of data sources. For example, database systems offer mechanisms such as replication or triggers in this context which ease the extraction and processing of update information. However, very often data updates have to be extracted from sources (e.g., legacy systems) which do not offer any support for getting this update information. In such cases, the only way is to produce snapshots of the data sources before and after the update and to calculate the difference (i.e., the differential snapshot) between them [LGM96, JNS+97] (cf. Figure 1a). A differ-
ential snapshot can be expressed by a set of change operations (i.e., INSERT, UPDATE, DELETE operations) leading from the source data to the target data.

In addition to data warehouses, change information also plays an important role in the domain of adaptive process management technology [RRD04b]. Such systems enable process changes at different levels, which occur frequently in practice [Wes01]: Single process instances may have to be adapted, for example, to deal with exceptional situations (e.g., by adding, deleting, or shifting process steps, cf. Figure 1b). Furthermore process templates\(^1\) may have to be changed, e.g., to react on new regulations or to implement process optimizations. The information about these changes has to be stored for several reasons. In [RRJK06] we discuss traceability (e.g., for the medical domain) and correctness checks in the context of concurrent process changes as important use cases.

Both application domains, data warehouses and process management systems, show that change logging is very important. However, one problem hampers the management and the use of change logs: Very often, change logs consists of a possibly large amount of unnecessary information. In data warehouses, for example, this information comprises wasteful DELETE/INSERT or INSERT/DELETE pairs within the differential snapshots; these pairs are often produced when calculating the differential snapshots. A wasteful DELETE/INSERT pair expresses the deletion of a database entry X followed by the insertion of X (i.e., the same entry) in the sequel. The same problem arises in the context of

\(^1\)Process templates describe the general structure of a process (e.g., control and data flow), cf. Figure 1b.
process management systems where change logs may contain change operations overriding the effects of previous changes. One example is illustrated for process instance I2 in Figure 1b: The first user updates the position of process activity A to (B, D) followed by an update of the second user to position (D, E) (i.e., the overall effect for both UPDATE operations is \text{UPDATE}(A, (D, E)) \). The presence of such unnecessary information within change logs may cause a number of problems and difficulties:

- **Performance problems due to inflated logs**: In case of differential snapshots this may lead to performance problems in case of tight update windows \cite{DDJ98}.

- **Data quality problem**: In data warehouses, redundant information from differential snapshots has to be cleaned within the staging area. Redundant change information might be desirable for traceability reasons on the one hand. On the other hand, \text{DELETE/INSERT} and \text{INSERT/DELETE} pairs do not have any real background and therefore should to be cleaned before analyzing the data.

- **Comparability problem**: In process management systems many correctness checks (e.g., dealing with concurrently applied changes) are based on comparing change logs. In this context, unnecessary information leads to non-comparable change logs, which significantly hampers correctness checks in the sequel \cite{RRD04a, RRJK06}.

This paper aims at tackling these problems which might be relevant for other use cases as well (e.g., when updating materialized views in data warehouses).

### 1.2 Contribution

In \cite{RRJK06} we have already discussed requirements for capturing (process) change logs in adaptive process management systems. We have also shown how to purge unnecessary information from process change logs at the logical level. In this paper, we present the \textsc{DeltaLayer} algorithm which takes arbitrary change log information as input and produces a cleaned output which only contains the net change effects (i.e., information about changes which actually have had an effect on the database, the data source, or the process repository). The resulting net changes are considered as being correct if they are minimal with respect to the sets of inserted, deleted, and updates tuples. This constitutes an extension of the minimality notion as used in, for example, approaches for updating materialized views \cite{GL95}. When compared to approaches on transaction equivalence \cite{AV88} which mainly operate at the logical level, our approach tackles the challenge to efficiently realize minimized change logs at the physical level. Contrary to \cite{AV88}, for example, the \textsc{DeltaLayer} algorithm produces output (i.e., minimized change logs) which can be directly used for data extraction in data warehouses or correctness checks within a process management system. This also implies that our approach is independent of any transactional concepts (i.e., transactions and associated recovery mechanisms can be specified on top of the change logs information as considered in this paper).
Since the output of the DELTALAYER algorithm is structured it can be accessed and queried more easily, for example, to post-process the net changes: When updating materialized views in data warehouses, it might be useful to determine which attributes are affected by changes and which are not. Regarding adaptive process management systems the access to specific entities (such as process activities) support the efficient processing of the necessary correctness checks (e.g., whether two changes are conflicting) [RRD04a].

After presenting the DELTALAYER algorithm we will show how it can be implemented within commercial systems such as Oracle database or IBM DataPropagator as well as within differential snapshot algorithms (e.g., the Window algorithm as, for example, presented in [LGM96]) in order to purge the output from wasteful information. For applications such as data warehouses, it is sufficient that the DELTALAYER algorithm is able to process the basic operations DELETE, UPDATE, and INSERT [LGM96]. However, it offers the possibility to process more complex changes on top of these basic operations as well (e.g., transactions, set-oriented updates, or "high-level" process graph changes). We will show this by means of an example for a "high-level" operation in the area of adaptive process management systems. In this context we can also see that the minimality property of the DELTALAYER algorithm is crucial. Minimality guarantees that two arbitrary change logs become comparable after running the DELTALAYER algorithm on them. This, in turn, is essential for any check which requires comparison of change information.

The paper is organized as follows: In Section 2 fundamental notions are introduced. We present the DELTALAYER algorithm and prove its minimality property in Section 3. Section 4 discusses several applications of the DELTALAYER algorithm. In Section 5 we discuss related work, and we finish with a summary and outlook in Section 6.

2 Fundamentals

In this section we introduce fundamental notions which we use further on in the paper and which are necessary for a basic understanding.

First of all, we restrict our considerations to the following set of basic change operations: \( CO := \{ \text{INSERT}(T), \text{UPDATE}(T), \text{DELETE}(T) \} \). We assume that tuple T has a key \( T.Key \) and a list of attributes \( T.A = [a_1, ..., a_n] \). Using this notion, for example, differential snapshots can be easily expressed: \( \text{INSERT}(T) \) refers to the insertion of a data field with key \( T.Key \) and attribute list \( T.A \) (cf. [LGM96]). Regarding change logs in adaptive process management systems, \( \text{INSERT}(T) \) expresses the insertion of a new activity node \( T.Key \) into the underlying process template (e.g., activity node X is inserted into process template S, cf. Figure 1b). The additional parameters stored within \( T.A \) might, for example, specify the position where \( T.Key \) is inserted. Furthermore, \( \text{UPDATE}(T) \) expresses an update of a data field having key \( T.Key \) with attributes \( T.A \) or an update of the position of a process activity node \( T.Key \) within a process template (e.g., updating the current position of activity node A to new position (D, E) for instance I2, cf. Figure 1b). Finally, for a delete operation it is sufficient to only specify the key of

\[ \text{DELETE}(T) \]

\[ \text{UPDATE}(T) \]

\[ \text{INSERT}(T) \]

\[ \text{DELETE}(T) \]
the tuple to be deleted.

In the following we denote attribute values which remain non-specified as NULL. This might be used, for example, when applying UPDATE operations for which only some of the attributes are updated whereas for the others the old values are kept (cf. Table 1).

One or several change operations as described above can be used to express modifications on data sources (e.g., update of source tables contained in differential snapshots). More precisely, we denote an ordered sequence of change operations as change log. Formally: $cL := < op_1, ..., op_n >$ where $op_i \in CO$ ($i = 1, ..., n$) and $op_i$ affects tuple $T$ having key $T.Key$ and attribute list $T.A$. Note that an operation $op_i$ is performed before another operation $op_j$ ($op_i, op_j \in cL$) for $i < j$. As running example we use the following simple change log for the remainder of this paper:

**Change Log 1 (Example Change Log)**

$$op_1 = \text{INSERT}((2, "HELLO")),$$
$$op_2 = \text{INSERT}((3, "BYE")),$$
$$op_3 = \text{DELETE}((2, NULL)),$$
$$op_4 = \text{INSERT}((4, "BYEBYE")),$$
$$op_5 = \text{UPDATE}((4, "HELLO")),$$
$$op_6 = \text{UPDATE}((4, "BYE")),$$
$$op_7 = \text{DELETE}((5, NULL)),$$
$$op_8 = \text{INSERT}((5, "CIAO")),$$
$$op_9 = \text{INSERT}((6, "HI")),$$
$$op_{10} = \text{INSERT}((7, "HIHI")) >$$

When analyzing this sample change log we can see that its actual effects on the original data source can be captured by a fraction of the original change operations, i.e.,

$$< \text{INSERT}((3, "BYE")), \text{INSERT}((4, "BYE")), \text{UPDATE}((5, "CIAO"))^{3}, \text{INSERT}((6, "HI")), \text{INSERT}((7, "HIHI")) >$$

From literature studies and experiences with our process management system we learned that unnecessary change pairs within change logs occur rather often. Examples are differential snapshots [LGM96] and change logs in process management systems [RRJK06]. Regarding the first case, wasteful change pairs are produced by the algorithms to compute the differential snapshots; in the latter case the unnecessary information might be caused by users trying out the best solutions for change operations (see, for example, instance I2 in Figure 1b). In any case such wasteful change pairs might cause performance problems [DDJ+98] and / or hamper correctness checks based on the change logs [RRJK06]. In Table 1 we summarize all different kinds of wasteful change pairs which might occur within change logs. Table 1 also analyzes their actual effects on the data source.

---

3In this case we have to check if attribute value CIAO means an update on the original table, otherwise the \text{INSERT}((5, "CIAO")) operation would be unnecessary too.
Table 1: Wasteful Change Pairs within Change Logs

<table>
<thead>
<tr>
<th>Change Pair</th>
<th>Effect</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT(T)/DELETE(T)</td>
<td>neutral</td>
<td>INSERT(12, &quot;HELLO&quot;), DELETE(12, NULL)</td>
</tr>
<tr>
<td>UPDATE(T)/DELETE(T)</td>
<td>delete</td>
<td>UPDATE(12, &quot;HELLO&quot;), DELETE(12, NULL) ⇒ DELETE(12, NULL)</td>
</tr>
<tr>
<td>DELETE(T)/INSERT(T)</td>
<td>a) same attribute values as tuple (2,...) had in source before ⇒ neutral</td>
<td>DELETE(12, NULL), INSERT(12, &quot;HELLO&quot;) ⇒ INSERT(12, &quot;BYE&quot;)</td>
</tr>
<tr>
<td></td>
<td>b) different attribute values as tuple (2,...) had in source before ⇒ update</td>
<td>DELETE(12, NULL), INSERT(12, &quot;BYE&quot;) ⇒ UPDATE(12, &quot;BYE&quot;)</td>
</tr>
<tr>
<td>UPDATE(T)/UPDATE(T)</td>
<td>a) second update has same attribute values as T in source ⇒ neutral</td>
<td>original attribute values of T in source (2, &quot;HELLO&quot;), UPDATE(12, &quot;BYE&quot;)</td>
</tr>
<tr>
<td></td>
<td>b) updates attribute values compared to attribute values of T in source ⇒ update</td>
<td>original attribute values of T in source (2, &quot;HELLO&quot;), UPDATE(12, &quot;BYE&quot;)</td>
</tr>
<tr>
<td></td>
<td>c) value update for more than one attribute ⇒ merge into one update</td>
<td>original attribute values of T in source (2, &quot;HELLO&quot;, 100), UPDATE(12, &quot;BYE&quot;, NULL)</td>
</tr>
<tr>
<td>INSERT(T)/UPDATE(T)</td>
<td>a) for one attribute ⇒ insert</td>
<td>INSERT(12, &quot;HELLO&quot;), UPDATE(12, &quot;BYE&quot;) ⇒ INSERT(12, &quot;BYE&quot;)</td>
</tr>
<tr>
<td></td>
<td>b) for more than one attribute ⇒ merge into insert operation</td>
<td>INSERT(12, &quot;BYE&quot;, NULL), UPDATE(12, NULL, 111) ⇒ INSERT(12, &quot;BYE&quot;, 111)</td>
</tr>
</tbody>
</table>

3 On Calculating the Minimal Effect of Change Logs – the DELTALAYER Algorithm

So far we have discussed different kinds of wasteful change combinations within change logs (cf. Section 2). In this section we present the DELTALAYER algorithm which purges a change log from change combinations as summarized in Table 1 and produces a "net information" output (i.e., an output only containing change operations which reflect the delta between target and source table).

3.1 The DELTALAYER Algorithm

Generally, the DELTALAYER algorithm (cf. Algorithm 1) receives a change log as input (such as Example 1) and produces an output within the DeltaLayer format. [RRJK06]
which only contains the net information of the input change log. In (1) the DeltaLayer structure is initialized; either a new DeltaLayer structure is created or an already existing one is used. Then the algorithm steps through the change log (i.e., an ordered sequence of change operations).

Algorithm 1 (DELTA: Layer: ChangeLog \( cL := \langle \text{op}_1, \ldots, \text{op}_n \rangle \mapsto \text{DeltaLayer} \ D_{Net} \))

\[
\text{DeltaLayer} \ D_{Net} := \text{newDeltaLayer} \\
\text{forall} (i = 1, \ldots, n) \text{ do} \\
\quad \text{if} \ \text{op}_i = \text{INSERT}(T) \ \text{then} \\
\quad \quad D_{Net} = \text{insertTuple}(D_{Net}, T) \\
\quad \text{else if} \ \text{op}_i = \text{DELETE}(T) \ \text{then} \\
\quad \quad D_{Net} = \text{deleteTuple}(D_{Net}, T) \\
\quad \text{else if} \ \text{op}_i = \text{UPDATE}(T) \ \text{then} \\
\quad \quad D_{Net} = \text{updateTuple}(D_{Net}, T) \\
\quad \text{fi} \\
\text{od}
\]

For each change operation \( \text{op}_i \), its type is determined (i.e., \text{INSERT}, \text{UPDATE}, \text{DELETE}). Based on the current change type, Algorithm 1 calls associated functions \text{insertTuple}(\ldots), \text{deleteTuple}(\ldots), \text{or} \text{updateTuple}(\ldots). Basically, these functions check if and what effect current operation \( \text{op}_i \) has on the target data table. For this decision the effects of already applied change operations have to be taken into account (cf. Table 1).

Let us assume that Algorithm 1 calls function \text{insertTuple}(\ldots) with the current DeltaLayer and the current change operation \( \text{op}_i = \text{INSERT}(T) \). First of all, function \text{insertTuple}(\ldots) checks whether tuple \( T \) has already been deleted before (i.e., by a change operation \( \text{op}_k \) with \( k < i \)). This corresponds to case 3 in Table 1. As we can see from this table, function \text{insertTuple}(\ldots) now has to distinguish whether the attribute values used when applying \( \text{op}_i \) are the same or different from the attribute values having been used for prior \( \text{op}_k = \text{DELETE}(T) \) operation (i.e., the original values in the source table in case of a DELETE operation). This is done in lines \((\star)\). If the attribute values used by \( \text{op}_i \) and \( \text{op}_k \) are different, obviously, \( \text{op}_i \) has to be stored within the DeltaLayer as \text{UPDATE}(T) operation (i.e., if \( \text{update} = \text{true} (\diamond) \)). In any case, \( T \) is deleted from the set of deleted tuples and therefore the previous \( \text{DELETE}(T) \) operation \( \text{op}_k \) is correctly purged from DeltaLayer. Except the \( \text{DELETE}(T)/\text{INSERT}(T) \) combination, according to Table 1, there is no other possibly wasteful change combination where an INSERT operation is applied in the second place. Therefore, if \( T \notin D_{old}.\text{deletedTuples} \), \( T \) can be added to the set of inserted tuples and \text{INSERT}(T) is currently stored within DeltaLayer.

Function 1 (insertTuple (DeltaLayer \( D_{old} \), newTupel \( T_{new} \)) \( \mapsto \text{DeltaLayer} \ D_{Net} \))

\[
D_{Net} := D_{old} \\
\text{boolean contains} := \text{false}
\]

A table including a column where the type of the particular change operation is indicated (e.g., \( I \) for an \text{INSERT} operation). Another possibility would be to keep an own table for \text{INSERT}, \text{UPDATE}, and \text{DELETE} operations. The choice may depend on factors such as performance or query optimization.
forall (T ∈ Dold.deletedTupels) do //†
if (T.Key = Tnew.Key) then //‡
boolean update := false
Tupdate = new T<Tnew.Key,{}>
n = |T.A|
forall (k = 1, ..., n) do
if (T.a_k ≠ Tnew.a_k) then //⋆
Tupdate.a_k = Tnew.a_k //⋆
update = true
fi
od
if (update = true) then //⋄
DNet.updatedTupels := DNet.updatedTupels ∪ Tupdate
fi
fi
DNet.deletedTupels := DNet.deletedTupels \ T
contains := true
break
fi
od
if (contains = false)
DNet.insertedTupels := DNet.insertedTupels ∪ Tnew
fi
return DNet

We omit the code for functions deleteTuple(...) and updateTuple(...) due to space restrictions, but explain their essence in the following: For op_i = DELETE(T), if there has been a previous operation op_k with op_k = INSERT(T) (k < i), op_k and op_i are purged from DeltaLayer by function deleteTuple(...). Reason is that combination INSERT(T)/DELETE(T) has no actual effect according to case 1 in Table 1. By contrast, case 2 in Table 1 is more interesting: Although there has been a previous UPDATE(T) operation, we have to insert the original attribute values into the DeltaLayer which have been contained within the source table before any change operation has been applied. Otherwise the DeltaLayer does not reflect correctly applicable changes. To get correct values it is not sufficient to check the source table due to intermediate (and already purged) UPDATE operations. Therefore, one alternative is to look up the original values from the target table which might decrease the performance of the algorithm (see Section 3.2). However doing so does not cause a blocking of the target table and therefore does not lead to an increase of the update window. Alternatively, the original attributes can be stored in an auxiliary data structure as soon as an UPDATE operation is applied. This raises storage needs but increases performance. The mechanisms used in function updateTuple(...) can be seen as a combination of those ones used for functions insertTuples(...) and deleteTuples(...).

Figure 2 shows how the DELTALAYER algorithm works on the input changes presented in Example 1. The crossed-out entries reflect the purged entries within the DeltaLayer.
Note that the algorithm can also be used if the Delta Layer is updated each time a change operation occurs, i.e., the input of algorithm 1 becomes a one-element change log. This approach is, for example, applied in our adaptive process management system ADEPT2, i.e., each time a change operation is applied either to a process instance or to a process template, the according Delta table is updated using the DELTALAYER algorithm. In Section 4.2 we will show how the same principle can be directly integrated within the Window algorithm producing differential snapshots.

3.2 Performance Considerations for the DELTALAYER Algorithm

In this section, we provide some performance considerations for the DELTALAYER algorithm. Obviously, the complexity of Algorithm 1 without considering functions insertTuple(...), deleteTuple(...), and updateTuple(...) is $O(n)$ with $n$ equals the number of change operations contained in the input change log. More interesting are the complexity considerations for functions insertTuple(...), deleteTuple(...), and updateTuple(...).

For function insertTuple(...) we obtain a complexity of $O(n * c) = O(n)$ since this function consists of a scan of set deletedTuples with $n$ elements $\Rightarrow O(n)$ (cf line †), a simple comparison in line (‡), and a check of all $c$ attributes of T $\Rightarrow O(c)$ with $c$ is constant. Furthermore, adding T to the set of updated or inserted tuples as well as deleting T from the set of deletedTuples (if necessary) is all of complexity $O(1)$. 

Figure 2: Applying DELTALAYER to Example Change Log
Function `deleteTuple(...)` also has a complexity of \( O(n) \): The check of the set of inserted tuples has a complexity of \( O(n) \) (\( n \) equals the number of inserted tuples). The scan of the set of updated tuples is of complexity \( O(m) \) with \( m \) equals the number of updated tuples. Therefore we obtain a complexity of \( O(n + m) = O(n) \) for `deleteTuple(...)`. (delete and insert into the sets of inserted, deleted, and updates tuples are again of a complexity of \( O(n) \)).

For function `updateTuple(...)` we obtain a worst case complexity of \( O(n^2) \): First of all there is a scan of the set of inserted tuples having complexity of \( O(n) \) with \( n \) equals the number of inserted tuples. The following check of the attribute set of a tuple \( T \) has complexity of \( O(c) \) with \( c \) equals the number of attributes of \( T \). The conditional scan of the set of updated tuples is of complexity \( O(m) \) (\( m \) equals the number of updated tuples). Then the number of attributes of tuple \( T \) is checked again with complexity of \( O(k) \) with \( k \) equals the number of attributes of tuple \( T \). Then the target table has to be accessed with complexity of \( O(t) \) with \( t \) equals the number of tuples in the target table. Note that here no locks on the target table become necessary. Furthermore we can decrease the complexity by storing the original attribute values of the affected tuples. Finally, a scan of the attributes of tuple \( T \) in the target table becomes necessary having complexity of \( O(l) \) with \( l \) equals the number of attributes of tuple \( T \). Altogether, the complexity of `updateTuple(...)` turns out as \( O(n + c) + O(m * k * t * l) \) where \( c, k, \) and \( l \) are constant. Consequently, the complexity is \( O(n) + O(m * t) = O(n) + O(n^2) = O(n^2) \).

Generally speaking, the number of the tuple in the DeltaLayer is quite small when compared to the number of tuples in source and target tables since only the minimal set of change operations is stored. Therefore scanning the sets of inserted, updated, and deleted tuples can be accomplished rather quickly. Generally, processing DELTALAYER algorithm is not the part potentially causing performance problems. The critical part is feeding the data into the data warehouse afterwards since the target tables have to be locked during this time (update window). The DELTALAYER algorithm minimizes the volume of the data to be fed into the data warehouse significantly and therefore can increase the update window quite dramatically. Furthermore, as we will show in Section 4.2, the DELTALAYER algorithm has not to be applied once for all (and possibly large) change logs. It is possible to directly integrate the DELTALAYER algorithm into, for example, differential snapshot algorithms. Therefore the different data mechanisms can work with the DELTALAYER algorithm asynchronously regaring the time updates on the data tables occur.

### 3.3 Correctness of the DELTALAYER Algorithm

In this section we show that the output produced by Algorithm 1 is correct, i.e., the output captured within the DeltaLayer is minimal. Minimality has been considered as important requirement in the context of updating materialized views as well. For example, in [GL95] minimality of the bag algebra expressions for updating a derived view has been proven for `INSERT` and `DELETE` operations. `UPDATE` operations applied to the base view have not been directly taken into account since they have been always transformed into associated `DELETE/INSERT` pairs. This is contrary to minimality itself.
In the following, we will adopt the minimality requirements for \texttt{INSERT} and \texttt{DELETE} operations for our algorithmic approach but will extend it by considering \texttt{UPDATES}.

**Definition 1 (Minimality of the DeltaLayer)** Let $S$ be the tuple set before the change and let $S'$ be the tuple set after applying change operations $op_i (i = 1, \ldots, n)$ captured within change log $cL = \langle op_1, \ldots, op_n \rangle$. Let further the sets of actually inserted, deleted, and updated tuples be defined as follows:

1. Actually deleted tuples: $\bigtriangledown cL := \{ t \mid t \in S \setminus S' : \nexists t' \in S' \text{ with } t'.Key = t.Key \}$
2. Actually inserted tuples: $\bigtriangleup cL := \{ t \mid t \in S' \setminus S : \nexists t' \in S \text{ with } t'.Key = t.Key \}$
3. Actually updated tuples: $\bigtriangledown cL := \{ t \mid t \in S \setminus S' \lor t \in S' \setminus S : \exists t' \in S' \text{ with } t'.Key = t.Key \}$

Then we call a change log or related data structure (e.g., a delta layer) $D$ minimal if and only if the following conditions hold:

1. $D.deletedTuples = \bigtriangledown cL$
2. $D.insertedTuples = \bigtriangleup cL$
3. $D.updatedTuples = \bigtriangledown cL$

The definitions of $\bigtriangledown cL$, $\bigtriangleup cL$, and $\bigtriangledown cL$ are illustrated in Figure 3a from a set-based point of view. As it can be seen from Figure 3b the distinction between updated and inserted or updated and deleted tuples is done by comparing keys.

![Figure 3: Sets of Inserted, Deleted, and Updated Tuples](image)

**Theorem 1 (Minimality of DeltaLayer)** Let $S$ be a tuple set and let $cL$ be a change log transforming tuple set $S$ into tuple set $S'$. Let further DeltaLayer $D$ be the output resulting from the application of Algorithm 1 to $cL$. Then $D$ is minimal according to Definition 1.

Due to lack of space we only sketch the proof of Theorem 1 for the set of inserted tuples $D.insertedTuples$. The proofs for deleted and updated tuples, however, can be accomplished in a similar way by contradiction.

**Proof Sketch:**
Proof by contradiction $\implies$ we have to show
1. \( D.\text{deletedTuples} \neq \forall \text{cL} \lor \)

2. \( D.\text{insertedTuples} \neq \triangle \text{cL} \lor (\ast) \)

3. \( D.\text{updatedTuples} \neq \triangledown \text{cL} \)

\[ \implies \neg (D \text{ produced by Algorithm 1}) \]

We proof \((\ast) \implies \neg (D \text{ produced by Algorithm 1}).\)

Auxiliary assumptions: a) \( \forall \ t \text{ used in the following } t \in S \lor t \in S' \text{ holds}, \)
b) \( \text{INSERT}(t) \) is the last entry in \( \text{cL} \) for \( t \), c) it is not possible to apply an \( \text{INSERT}(t) \) operation after an \( \text{UPDATE}(t) \) operation (would be rejected by the database system),

\[ D.\text{insertedTuples} \neq \triangle \text{cL} \equiv \]

\[ D. \text{insertedTuples} \neq \{t \mid t \in S' \setminus S, \exists t' \in S \text{ with } t'.\text{Key} = t.\text{Key} \} \implies \exists t1 \in D.\text{insertedTuples} \]

1. \( t1 \in S \setminus S' \lor \)

2. \( t1 \in S \cap S' \lor \)

3. \( (t1 \in S' \setminus S \land \exists t2 \in S \text{ with } t1.\text{Key} = t2.\text{Key}) \)

\[ \implies \]

1) \( t1 \in S \setminus S' \implies t1 \text{ has been deleted or updated } \implies \text{contradiction to (34) of Algorithm 1 and to b) and c) of auxiliary assumptions} \)

2) \( t1 \in S \cap S' \implies (t1 \text{ has not been affected by any change}) \lor \text{INSERT}(t1) / \text{DELETE}(t1) \) pair in \( \text{cL} \) (with same attributes) \lor \text{DELETE}(t1) / \text{INSERT}(t1) pair in \( \text{cL} \) \lor \text{UPDATE}(t1) / \text{UPDATE}(t1) pair in \( \text{cL} \) (with original values of \( t1 \) in \( S \) for last update)\)

\[ \implies \text{contradiction to 1) } \lor \text{contradiction to (14,28,33) } \lor \text{contradiction to (40,42,56,57) } \lor \]

\[ \text{contradiction to (76,77,97) of Algorithm 1} \]

3) \( t1 \in S' \setminus S \land \exists t2 \in S \text{ with } t1.\text{Key} = t2.\text{Key} \implies \text{UPDATE}(t1) \text{ in } \text{cL} \implies \text{contradiction to (63) of Algorithm 1 and to b) and c) of auxiliary assumptions} \)

4 On Applying the DELTALAYER Algorithm

In this section we provide different application scenarios for the DELTALAYER algorithm. They range from commercial systems such as Oracle or IBM DataPropagator to an extension of the used change operations within process management systems.

\[ \text{For all change combinations see Table 1.} \]
4.1 Application within Commercial Systems

In the following, we show how the DEELAYER algorithm can be applied in the context of two main commercial systems which offer support for data warehouses:

**Oracle Database:** Within the data warehousing guide of Oracle [Dat03] different possibilities for data change capture are summarized. Closely related to the problems described in this paper is the `merge` statement which is an extension of SQL. Basically, the merge statement is used to transform data from source tables into the format of the target table.

```
(1) MERGE INTO products t USING products_delta s
(2) ON (t.prod_id=s.prod_id)
(3) WHEN MATCHED THEN UPDATE SET
(4) t.prod_list_price=s.prod_list_price, t.prod_min_price=s.prod_min_price
(5) WHEN NOT MATCHED THEN INSERT (prod_id, prod_name, prod_desc,
prod_subcategory, prod_subcategory_desc, prod_category, prod_category_desc, prod_status,
prod_list_price, prod_min_price)
(6) VALUES (s.prod_id, s.prod_name, s.prod_desc, s.prod_subcategory,
(7) s.prod_subcategory_desc, s.prod_category, s.prod_category_desc, s.prod_status,
(8) s.prod_list_price, s.prod_min_price);
```

Figure 4: `MERGE` Statement provided by Oracle Data Warehousing Guide [Dat03]

In the above statement `products t` refers to the target table and `products_delta` refers either to the source or to the delta table. In (2) the merge criteria are stated. If the merge criteria are fulfilled then the associated tuples within the source / delta tables are updated (3-4). Otherwise new tuples are inserted (5-10) (where column name of source and target table do not necessarily have to match). The `merge`-statement offers the possibility of expressing an "if-then-else"-like semantics (cf. Figure 4.1). Therefore it could be used in a slightly modified way to implemented the "if-then-else"-constructs of the DEELAYER algorithm. Using a table representing the changes of an unpurged log as input, the modified `merge`-statements (one for each change operation) produce a correct net delta in the output table.

Another, maybe even simpler possibility is to implement the DEELAYER algorithm within JAVA and make it accessible using the mechanism of user-defined functions of Oracle database.

**IBM DataPropagator:** Another commercial system dealing with delta information in the context of replicating distributed data sources is IBM DataPropagator [IBM95]. Using IBM DataPropagator, basically, it is possible to produce so called condensed change data tables which are supposed to contain net update information. However, condensed change data tables are neither minimal nor correct in most cases. Let us apply the triggers presented in [IBM95] on the change combinations summarized in Table 1. If, for example, a change of type `DELETE(T)` is applied, the before trigger, first of all, deletes any entry for which key equals `T.Key` holds (regardless whether it is an update or insert operation) and the after trigger stores the `DELETE(T)` operation afterwards. Doing so, in case of unnec-
essary INSERT(T)/DELETE(T) pairs the DELETE(T) operation remains within the condensed change data table (therefore the table is not minimal). Furthermore, this result is even not correct if key T.Key is not key of the data warehouse schema afterwards (otherwise the database system would simply reject the application of the DELETE(T) operation). In case of unnecessary INSERT(T)/UPDATE(T) pairs the triggers always produce an incorrect output since tuple T having key T.Key is no longer present when the UPDATE(T) operation is applied. Consequently, the triggers provided for IBM DataPropagator so far could be improved by implementing them according to the DELTALAYER algorithm. Due to lack of space we abstain from details here.

4.2 Post Processing of Differential Snapshots

Another application for the DELTALAYER algorithm is post processing the output of differential snapshot algorithms. As already mentioned in [LGM96] this would constitute a valuable extension in order to minimize update windows within the data warehouse [DDJ+98]. One alternative in this context is to take the whole differential snapshot as input for the DELTALAYER algorithm (real post processing). However, we can also think of directly integrating the DELTALAYER algorithm into the differential snapshot algorithms. For the Window algorithm [LGM96], for example, this can be accomplished rather easily: instead of writing entries from AgingBuffer1 into the DELETE queue and entries from AgingBuffer2 into the INSERT queue all entries from the aging buffers can be directly written into the DeltaLayer. There an immediate purge of wasteful INSERT/DELETE pairs takes place.

4.3 An Extension Towards Arbitrary Change Operations

INSERT/DELETE/UPDATE operations are rather simple changes. In applications such as data warehouses [GM95] or process management systems more advanced change operations become necessary in practical applications [RRJK06]. In our ADEPT process management system, for example, we offer the change framework depicted in Figure 5a. Here, first of all, we distinguish between primitive and high-level change operations in order to offer better user support. Thereby the high-level change operations are constructed by combining an arbitrary amount of change primitives. Furthermore, high-level change operations are equipped with a set of (formal) pre- and post-conditions which guarantee the correctness of the resulting process when applying a high-level change operation.

The main differences between high-level operations and change primitives are their properties and the intention they are used for. High-level operations act as an interface provided to the end-user. Furthermore – contrary to the set of change primitives – the set of high-level change operations does not necessarily have to be closed (i.e. the process management system can be extended by new high-level change operations any point in time). Change primitives are mainly used to create the data structures (in particular the
DeltaLayer) based on which (high-level) changes are represented at system-level. More precisely, there is a mapping for each high-level change operation to associated change primitives (including those high-level operations which newly defined within the process management system). Based on the concept of mapping high-level change operations at system level to a change primitive representation, the associated change primitives can be used for manipulating the internal representation, i.e. the DeltaLayer. Therefore, for high-level change operations the DELTALAYER algorithm can again be used in order to keep the DeltaLayer correct and minimal. Minimality of the change information, in turn, is crucial for correctness checks in the context of concurrently applied changes which are mainly based on comparing the associated change logs.

In Figure 5b, for example, a process template change is illustrated which is accomplished by applying two high-level change operations \( \text{sInsert}(...) \) and \( \text{sMove}(...) \) at the user level. At the system level, the high-level operation have been automatically transformed into a change primitive representation. Note that the DELTALAYER algorithm presented in this paper can be easily adapted to the change primitives used in adaptive process management systems (e.g., \( \text{addNode}(...) \), \( \text{addEdge}(...) \), or \( \text{addDataElement}(...) \)) as it has been shown in [RRJK06]. Based on this primitive representation, the DELTALAYER algorithm produces the purged DeltaLayer which automatically keeps a minimal change information for correctness checks at any time.

![Figure 5: a) ADEPT Change Framework and Example Process Template Change](image-url)

**Figure 5:** a) ADEPT Change Framework and Example Process Template Change

Change Logs

<table>
<thead>
<tr>
<th>Change Log</th>
<th>High-Level Change Operations</th>
<th>Change Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>sInsert(...)</td>
<td>( \text{sInsert}(...) ), ( \text{sDelAct}(...) ), ...</td>
<td>( \text{addNode}(...) ), ( \text{deleteNode}(...) ), ...</td>
</tr>
<tr>
<td>sMove(...)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MAPPING**

\( cLT^1(S) = (\text{sInsert}(S, \text{Lab test}, \text{Examine Patient}, \text{Deliver report}), \text{sMove}(S, \text{Inform Patient}, \text{Prepare Patient}, \text{Examine Patient})) \)

\( cL^{rev}_{\text{S}} = (\text{addNode}(S, \text{Lab test}), \text{removeEdge}(S, \text{Examine patient, Deliver Report, Ctrl}), \text{addEdge}(S, \text{Examine Patient, Lab test, Ctrl}), \text{addEdge}(S, \text{Lab test, Deliver report, Ctrl}), \text{removeEdge}(S, \text{Enter order, Inform patient, Ctrl}), \text{removeEdge}(S, \text{Inform patient, Prepare Patient, Ctrl}), \text{removeEdge}(S, \text{Prepare patient, Examine patient, Ctrl}), \text{addEdge}(S, \text{Enter order, Prepare patient, Ctrl}), \text{addEdge}(S, \text{Prepare Patient, Inform patient, Ctrl}), \text{addEdge}(S, \text{Inform patient, Examine Patient, Ctrl})) \)

Purged Delta Layer

378
5 Related Work

There are several suggestions how to extract and feed updates from data sources into a data warehouse. One complete architecture has been proposed within the WHIPS project [HGMW+95] ranging from data extraction to the update of materialized views.

**Snapshot-Based Approaches:** If data sources do not offer any support for extracting data updates (typical for legacy systems) the only way to do so is to build snapshots of the files before and after the change. Then the difference between those snapshots is computed, e.g., by the algorithm proposed in [LGM96]. Commercial systems such as Oracle database offer the possibility to create different snapshots [BG04]. The resulting change log is loaded into the data warehouse afterwards. However, such algorithms produce wasteful change combinations as well, i.e., unnecessary \( \text{INSERT}(T)/\text{DELETE}(T) \) or \( \text{DELETE}(T)/\text{INSERT}(T) \) pairs as depicted in Table 1. This may cause overhead when loading the files afterwards [LGM96] as well as unnecessary checks during the transformation phase within the staging area later on. Therefore a post processing as proposed in this paper would be beneficiary for working with differential snapshots.

**Replication-Based Approaches:** Replication approaches range from copying database tables to storing the updates tuples within so called delta tables. In particular the latter is related to our approach.

**Materialized Views:** The principles of net changes and delta tables may be also interesting for the maintenance of views in data warehouses [LYG99, LYC+00, ZGHW95, GL95], i.e., the problem of propagating changes of a base table \( B \) to a view \( V \) derived from \( B \). Here the incremental update of views is of particular interest, i.e., updated view \( V' \) is calculated as original view \( V \) minus the set of deleted tuples \( \triangledown V \) plus the set of inserted tuples \( \triangle V \). The challenge is to determine sets \( \triangledown V \) and \( \triangle V \). This comprises two aspects: first of all, it has to be determined which tuples have been changed in \( B \) and secondly, how to calculate the sets \( \triangledown V \) and \( \triangle V \) from that (e.g., the necessary aggregations). Our approach can be taken to optimize the result of step one, the determination of changes applied to \( B \): Since \text{UPDATES} on \( B \) are expressed as \text{INSERT} and \text{DELETE} pairs [LYG99] the minimality requirement for the view updates only refer to \text{INSERT} and \text{DELETE} operations [GL95]. However, our algorithm additionally considers minimality regarding \text{UPDATE} operations.

**Object-Oriented Approaches:** Delta objects have been proposed for object-oriented databases [SBDU97]. Here the changes are associated with the objects, not stored within a global delta log. Furthermore, the purpose of managing the delta information is different. In this context, delta information is, for example, used for debugging and run-time testing.

**Adaptive Process Management:** In general, adaptivity in process management systems has been a hot topic in literature for many years. However, there are only few approaches dealing with an efficient implementation of advanced process management functionality [Wes98, KAS+03]. So far, they have neglected issues related to change log management. Our ADEPT system is one of the very few available research prototypes for adaptive, high-performance process management [RRD04b, RRKD05, RRJK06].
6 Summary and Outlook

In this paper we have introduced the DELTALAYER algorithm which receives a change log as input and produces a structured net change within the DeltaLayer format as output. We have shown that the algorithm produces a correct (i.e., minimal) net change with respect to the set of inserted, deleted, and updated tuples. Furthermore in the paper different application areas for the DELTALAYER algorithm have been provided such as replication mechanisms in commercial systems (e.g., Oracle or IBM DataPropagator), post processing of differential snapshots to feed data updates into data warehouses, or correctness checks for adaptive process management systems.

At the moment we are integrating the DELTALAYER algorithm within the implementation of our process management engine ADEPT2 (for more information see www.aristaflow.de). This is also particularly interesting since maintaining a DeltaLayer provides transparency to any change operations which is added to the change framework afterwards. Based on the implementation we will conduct performance studies within different scenarios.

In future work we want to study more application areas for the DELTALAYER algorithm. One example is the cleaning of data within the staging area of a data warehouse (cf. Figure 6). The idea is to purge data (if desired!) after necessary transformation using metadata within a DeltaLayer. Doing so, for example, redundancies can be purged.

![Figure 6: Cleansing Data from Different Sources withing Staging Area](image-url)

References


Untersuchung des Einflusses verschiedener Bild-Features und Distanzmaße im inhaltsbasierten P2P Information Retrieval

Soufyane El Allali Daniel Blank Martin Eisenhardt Andreas Henrich Wolfgang Müller

vorname.nachname@wiai.uni-bamberg.de


1 Einführung

Peer-to-Peer-(P2P)-Netzwerke entstehen durch den Zusammenschluss mehrerer autonomer, kooperierender Rechnerknoten, die ohne den Einsatz eines zentralen Servers interagieren. Solche Netze eignen sich besonders für die dezentrale Verwaltung großer Datenmengen bzw. für die gemeinsame Nutzung von Ressourcen. Im Vergleich mit einer klassischen Client-Server-Architektur erhöhen sie die Ausfallsicherheit des Gesamtsystems, da sie nicht von einem single point of failure abhängig sind.

Eine Vielzahl eingesetzter P2P-Systeme nutzt bei der Suche lediglich inhaltsbeschreibende Annotationen (sog. Tags) bzw. Teile des Dateinamens, um Medienobjekte zu finden\textsuperscript{1}. Auch Flickr\textsuperscript{2} unterstützt ausschließlich eine tag-basierte Bildsuche. Diese Herangehensweise greift zu kurz, da einerseits Informationen in Tags bewusst verfälscht werden können und andererseits Homonyme, Synonyme oder Sprachvarianten die Suche erheblich erschweren. Ein Großteil der Nutzer macht von der Möglichkeit ihre Bilder mit Tags an-

\textsuperscript{1} Diese Arbeit wurde von der Deutschen Forschungsgemeinschaft im Rahmen des Projekts „Skalierbares, inhaltsbasiertes Retrieval von Text und Multimedia-Dokumenten in Peer-to-Peer Netzwerken“ gefördert.
\textsuperscript{2} z.B. KaZaa, http://www.kazzao.com, letzter Abruf: 04.10.2006
\textsuperscript{3} Flickr, http://www.flickr.com, letzter Abruf: 30.11.2006


Um die dabei erzielten Ergebnisse besser einschätzen zu können, vergleichen wir ferner die zunächst betrachteten P2P-Systeme, bei denen lediglich Zusammenfassungen im Netz verteilt werden (die Indexdaten zu den einzelnen Dokumenten verbleiben auf den Peers, auf denen die Dokumente selbst liegen), mit einer Variante, bei der selektiv auch Datensätze bzw. Indexdaten zwischen Peers transferiert werden, sowie einer auf CANs basierenden verteilten Indexstruktur [GYGM04].


2 Ein Überblick über verwandte Ansätze


Inhaltsbasierte Suchdienste in strukturierten P2P-Netzen wie bspw. Minerva [Be05] oder PRISM [Sa05] lassen sich auf Basis verteilter Hashtabellen (sog. DHTs) implementieren. Verschiedene Erweiterungen von CANs [TXM02, GYGM04] erlauben ebenso die


Obwohl das Spektrum der im zentralisierten CBIR verwendeten Farb-Features sehr breit ist (eine Übersicht geben [Ha06]), fehlen Vergleiche verschiedener Bild-Features im Falle von P2P-Systemen. Deselaers et al. [DKN04] verglichen verschiedene Features für den zentralen Fall. Sie stellen fest, dass die Wahl des jeweiligen Features sehr stark domänendependent ist. So sind Farbhistogramme (z.B. [SB91, ZLZ99, SC95]) unentbehrlich um bei Farbfotografien gute Ergebnisse zu erzielen. Da sich unsere Datenkollektion (s. Abschnitt 5) aus Photographien verschiedener Nutzer mit unterschiedlicher Herkunft, Hobbys, etc. zusammensetzt, verwenden wir in unseren Experimenten (s. Abschnitt 5) u.a. einige Varianten dieser Farbhistogramme.

Puzicha et al. [Pu99] verglichen Distanzmaße für verschiedene Anwendungsszenarien. Einige Maße, die hierbei vielversprechende Resultate erzielen, sind in Abschnitt 4.3 erläutert und werden anschließend verwendet. Zusätzlich untersuchen wir weitere Distanzmaße, die sich im Bereich des Bildretrievals als nützlich erwiesen haben [HR05, Qu04].

3 Die Peer-to-Peer-Umgebung

Unsere Untersuchungen basieren auf PlanetP [CAN02]. Im Folgenden verwenden wir clusterzentroid-basierte Zusammenfassungen, die durch einen Rumor Spreading-Prozess im P2P-Netz verteilt werden. Der zugrunde liegende Mechanismus ist sehr gut in [CAN02] bzw. [MEH05] beschrieben, so dass wir uns in den folgenden Abschnitten auf eine knappe Darstellung beschränken können. Für unser Verfahren ist es von Bedeutung, dass sich die
Peers in periodischen Zeitabständen gegenseitig kontaktieren, um Zusammenfassungen auszutauschen und so das P2P-Netz aktuell zu halten.

**Zusammenfassungen für eine effiziente Ressourcenauswahl**


Zur Gewinnung der Cluster könnte man etwa den \(k\)-Means-Algorithmus einsetzen. Frühere Arbeiten haben aber gezeigt, dass das Ranking der Peers nur unter gewissen Umständen von einem verteilten \(k\)-Means-Clustering der Dokumente profitiert [Ei06]. Daher werden im Rahmen dieser Arbeit zufällig 256 Dokumente aus der Dokumentenkollektion als Cluster-Zentroide ausgewählt, so dass diese die Verteilung der Datenpunkte widerspiegeln.


**Ranking der Peers**

Der Mechanismus, der die Peers bzgl. der Anfrage rankt und determiniert, in welcher Reihenfolge die Peers kontaktiert werden, nutzt die Zusammenfassungen der einzelnen Peers sowie die zufällig ausgewählten Cluster-Zentroide. Unter drei verschiedenen Rankingmechanismen hat sich *StableSortRanker* als der vielversprechendste erwiesen [Ei06].

Dieser Mechanismus trifft eine Entscheidung auf Grundlage von \(L_{cl}\), einer Liste, die die globalen Cluster-Zentroide, sortiert in aufsteigender Ordnung bezüglich ihrer Distanz zur Anfrage, enthält. Das erste Element dieser Liste entspricht immer dem Zentroid des Anfrageclusters, d.h. dem Zentroid des Clusters in dem die Anfrage selbst liegt. Peers mit vielen Dokumenten im Anfragecluster werden höher gerankt als Peers mit wenigen Dokumenten im Anfragecluster. Sofern Peer \(\alpha\) und Peer \(\beta\) die gleiche Anzahl an Dokumenten im momentan betrachteten Cluster haben, wählt *StableSortRanker* das nächste Element aus \(L_{cl}\) und vergleicht rekursiv Peer \(\alpha\) und Peer \(\beta\) bezüglich der Anzahl ihrer Dokumente in diesem Cluster, bis entweder eine Entscheidung getroffen werden kann oder das Ende von \(L_{cl}\) erreicht ist.

**4 Feature-Extraktion und Ähnlichkeitsberechnung**

Bei der inhaltsbasierten Suche werden Medienobjekte in der Regel durch hochdimensionale Feature-Vektoren \(\vec{d} = (d_1, ..., d_\delta)\) repräsentiert. Häufig verwendete Feature-Klassen sind hierbei Farbe, Textur und die Form von Objekten, die auf dem Bild zu sehen sind, sowie deren räumliche Lage. Anfragen werden häufig in Form einer *query by example* gestellt, bei der der Anfragende ein oder mehrere Anfragebilder auswählt, zu denen rele-
vante Bilder aus der Dokumentenkollektion gefunden werden sollen. Um nun die zu einer gegebenen Anfrage relevanten Dokumente finden zu können, muss auch das Anfragebild in Form eines Feature-Vektors $\vec{q} = (q_1, ..., q_δ)$ repräsentiert sein.

In Abschnitt 5 werden Messungen basierend auf verschiedenen Feature-Distanz-Kombinationen vorgestellt. Im Folgenden werden zunächst die verwendeten Farb-Features vorgestellt. Ferner untersuchen wir in unseren Experimenten die Retrieval-Leistung bei Reduzierung der Dimensionalität der Feature-Vektoren, weshalb in Abschnitt 4.2 kurz auf die Hauptkomponentenanalyse eingegangen wird. Abschnitt 4.3 beschreibt die von uns untersuchten Distanzmaße, die als Basis für die Ähnlichkeitsberechnung zwischen Dokumenten und Anfrage dienen.

4.1 Farb-Features

Die Analyse von Farbverteilungen bietet den Vorteil, dass sie größtenteils unabhängig vom Blickwinkel des Fotografen und der gewählten Auflösung ist. Als Farbmodelle werden oft-mals der HSV-, der RGB-, sowie der CIE-Farbraum verwendet. Swain und Ballard [SB91] verwenden Farbhistogramme, um Dokumente in einer Bilddatenbank zu indexieren, wobei die Länge $\delta$ der Histogramme $\vec{d} = (d_1, ..., d_δ)$ durch die Farben des Farbmodells determiniert wird. Die Werte $d_i$ entsprechen hierbei den relativen Vorkommenshäufigkeiten eines Farbwertes $i$ im Bild. Um die Repräsentationen kompakt zu halten, bietet sich die Möglichkeit der Quantisierung. In dieser Arbeit werden zwei Arten globaler Farbhistogramme basierend auf dem HSV-Farbraum verwendet, die jeweils ein Farbhistogramm für das gesamte Bild berechnen, ohne es in Regionen aufzuteilen.

**HSV36q:** Zhang et al. [ZLZ99] schlagen eine Quantisierung in 36 Farben (sog. bins) vor, wobei die Quantisierung nicht gleichmäßig erfolgt. Vielmehr wird die Hue-Komponente des HSV-Farbmodells in sieben Farben unterteilt, so dass diese den Farben, die in der chinesischen Sprache bekannt sind, entsprechen. Die Saturation/Value-Ebene des Farbraums wird in sechs Regionen unterteilt, wobei für $V \leq 0.2$ unabhängig von S- und H-Wert ein Bin vorgesehen ist. Daraus resultieren $7 \cdot 5 + 1 = 36$ Dimensionen. Acht der 36 Farben sind Grautöne, weshalb sich dieses Quantisierungsschema sowohl für Farb- als auch für Schwarzweißbilder eignet.

**HSV166q:** Ein anderes im CBIR häufig verwendetes Quantisierungsschema wurde von Smith und Chang [SC95] vorschlagen und quantisiert den HSV-Farbraum gleichmäßig in 166 Bins; 18 Intervalle in der Hue-Dimension, drei in der Saturation-Dimension und drei in der Value-Dimension. Vier weitere Bins repräsentieren Grauwerte.

**LocHistHSV36q:** Lokale Farbhistogramme erfassen die Farbverteilungen bestimmter Regionen eines Bildes. So ist es bspw. möglich das Bild in eine bestimmte Anzahl $n$ kleiner Bilder zu unterteilen und für diese $n$ Farbhistogramme zu berechnen. In dieser Arbeit wird ein Bild in 16 rechteckige Regionen unterteilt und für diese wird jeweils ein HSV-Farbhistogramm berechnet, das den Farbraum wie beschrieben in 36 Bins quantisiert. Daraus resultiert ein 576-dimensionaler Feature-Vektor.

**COLCOHER:** Farbkohärenzvektoren (CCVs) [PZM96] klassifizieren ein Pixel eines Bins
als kohärent, wenn es Teil einer großen Region mit ähnlichen Farben ist. Ist dies nicht der Fall, fällt es in die Klasse inkohärent. Hierzu werden zwei Histogramme mit je 64 Dimensionen berechnet. Insgesamt ergeben sich so Feature-Vektoren mit 128 Dimensionen. CCVs vermeiden einen Vergleich von kohärenten Pixeln eines Bildes mit inkohärenten eines anderen Bil des und umgekehrt. Zunächst wird ein Pixelwert geglättet und durch den Durchschnittswert der acht benachbarten Pixel repräsentiert. Im Anschluss werden die Pixel an hand des RGB-Farbraumes gleichförmig in 64 Bins quantisiert (drei Farbkanäle mit je vier Farben) und anschließend klas sifi ziert. Maßgebend für die Einordnung eines Pixels als kohärent ist die Menge der gleichfarbigen, benachbarten Pixel, die größer als ein festgelegter Schwellenwert (5% der Pixelanzahl eines Bil des) sein muss.


Neben Farb-Features haben wir ebenso verschiedene Textur-Features betrachtet. Da die Erkenntnisse hieraus sich größtenteils mit den bei der Analyse der Farb-Features gewonnenen decken, verweisen wir auf [AI06].

### 4.2 Hauptkomponentenanalyse


### 4.3 Distanzmäße

Typische $k$-NN-Anfragen suchen im Datenbestand nach den $k$ Feature-Vektoren, die den geringsten Abstand zum Anfragevektor $\vec{q}$ aufweisen. Der Abstand zweier Vektoren $\vec{q}$ und $\vec{d}$ wird hierbei mittels sog. Distanzmaße $\text{dist}(\vec{q}, \vec{d})$ ermittelt. Da wir in Abschnitt 5 die Leistungsfähigkeit verschiedener Feature-Distanz-Kombinationen untersuchen, werden nun die von uns eingesetzten Distanzmaße vorgestellt. Das Spektrum der im CBIR verwendeten Distanzmaße ist breit, einen kurzen Überblick geben u.a. [RTG00].

**Minkowski-Distanz:** $\text{dist}_{L_m}(\vec{q}, \vec{d}) = (\sum_{i=1}^{|q_i - d_i|^m})^{1/m}$

Im Bereich CBIR häufig verwendete Distanzmaße sind drei Ausprägungen der Minkowski-Distanz [SB91, SO95, RTG00]; es sind dies die Manhattan-Distanz $\text{dist}_{L_1}$, die Eukli-
Dische Distanz $dist_{L_2}$ sowie die $L_{max}$-Distanz $dist_{L_{max}}$. Letztere resultiert aus obiger Formel für $\lim_{m \to \infty} dist_{L_m}(\vec{q}, \vec{d})$ und entspricht dem Betrag der maximalen Differenz zwischen zwei Vektorkomponenten, die den gleichen Index besitzen.

**Fraktionale Distanz:**
Während klassische Minkowski-Distanzmaße für $m \geq 1$ definiert sind, erweitern [AHK01] diese Definition auch für Werte $0 < m < 1$ mit dem Ziel, ein gegenüber $L_1$ oder $L_2$ günstigeres Verhalten des Distanzmaßes zu erzielen. Howarth und Rüger [HR05] bestätigen in ihren Untersuchungen, dass ein Wert von $m = 1/2$ meist bessere Retrieval-Ergebnisse als etwa $dist_{L_1}$ oder $dist_{L_2}$ liefert. Daher verwenden wir $m = 1/2$ in unseren Experimenten.

**Symmetrische Kullback-Leibler Divergenz:** $dist_{SKL}(\vec{q}, \vec{d}) = \frac{1}{2} \sum_i (q_i - d_i) \log \frac{q_i}{d_i}$
Bei der Kullback-Leibler Divergenz handelt es sich um ein Maß, das seinen Ursprung in der Informationstheorie hat. Es misst die minimale durchschnittliche Anzahl von verschwendeten Bits, wenn man einen Prozess mit Verteilung $\vec{q}$ auf der Basis von $\vec{d}$ kodiert. Da im Bereich des CBIR für zwei Feature-Vektoren $\vec{d}'$ und $\vec{d}''$ gelten soll $dist(\vec{d}', \vec{d}'') = dist(\vec{d}'', \vec{d}')$, verwenden wir eine symmetrische Variante der Kullback-Leibler Distanz.

**Kosinusmaß:** $dist_{cos}(\vec{q}, \vec{d}) = \frac{\sum_i q_i \cdot d_i}{\sqrt{\sum_i q_i^2 \cdot \sum_i d_i^2}}$
Vielfach wird im Bereich des Information Retrievals bei $k$-NN-Anfragen, speziell auch im CBIR [Qu04], der Kosinus des Winkels zweier Vektoren als Maß für die Unähnlichkeit zweier Dokumente eingesetzt. Je geringer dieser Winkel desto größer ist die Ähnlichkeit der durch die Vektoren repräsentierten Dokumente.

Bin-By-Bin-Distanzmaße, wie sie zuvor vorgestellt wurden, vergleichen die Feature-Vektoren komponentenweise. Dem liegt die Annahme zugrunde, dass die Komponenten von ihrer semantischen Bedeutung her orthogonal sind. Dies ist jedoch gerade bei Farbhistogrammen nicht gegeben, beispielsweise ist die Farbe hellrosa der Farbe rosa~ahnlicher als der Farbe hellblau. Bei Bin-By-Bin-Distanzmaßen werden jedoch rosa und hellblau beide als gleich unähnlich zu hellrosa betrachtet. Mit Cross-Bin-Distanzmaßen können demgegenüber Zusammenhänge zwischen den Bins erfasst werden:

**Match-Distanz:** $dist_{match}(\vec{q}, \vec{d}) = \sum_i |Q_i - D_i|$ Sowohl die Match-Distanz als auch die im Folgenden vorgestellte Kolmogorov-Smirnov-Distanz arbeiten mit kumulierten Histogrammen. Das kumulierte Histogramm $\vec{D}$ eines Vektors $\vec{d}$ ist definiert als $(D_1, ..., D_k)$, wobei $D_i = \sum_{j \leq i} d_j$. Die Match-Distanz zweier eindimensionaler Vektoren ist demnach definiert als die Manhattan-Distanz ihrer kumulierten Histogramme.

**Kolmogorov-Smirnov-Distanz:** $dist_{KS}(\vec{q}, \vec{d}) = \max_i |Q_i - D_i|$ Bei der Kolmogorov-Smirnov-Distanz handelt es sich um eine Maßzahl aus der Statistik, die definiert ist als $L_{max}$-Distanz zweier kumulierter Verteilungen.

5 Experimente

Die Experimente in Abschnitt 5.1 basieren auf 50 Simulationsläufen mit jeweils 100 Anfragen, wobei jeweils ein zufällig ausgewähltes Dokument aus der Kollektion als Anfra-
Abbildung 1: Verteilung der Peer-Größe bei 250k-bzw. 50k-Kollektion


5.1 Vergleich des verteilten mit dem zentralisierten Retrieval-Ergebnis


Abbildung 2: Farb-Features bzgl. zu kontaktierender Peers (li.) bzw. gesehener Dokumente (re.)


Der Einfluss der Hauptkomponentenanalyse (Feature: HSV36q, Distanz: Manhattan) auf das Retrieval ist in Abb. 3 (re.) dargestellt. Erwartungsgemäß vergrößert sich der Aufwand zur Annäherung des zentralen Ergebnisses mit zunehmender Dimensionsanzahl. Je geringer die Anzahl der Dimensionen der Feature-Vektoren, desto weniger Peers müssen kontaktiert werden, um die Top-20-Dokumente aufzufinden. Die Ursache hierfür liegt wohl
Abbildung 3: Einfluss der Distanzmaße (li.) bzw. PCA (re.) auf die Anzahl zu kontaktierender Peers (Feature: HSV36q)

Abbildung 4: Anteil zu kontaktierender Peers, um alle Top-20-Bilder zu finden

im *Curse of Dimensionality*, der eine Ähnlichkeitssuche im hochdimensionalen Raum aufwändig macht [HAK00]. Gibt der Nutzer sich mit 60% der Top-20 zufrieden, so müssen jedoch bei allen Dimensionalitäten gleich viele Peers (etwa 3,5%) kontaktiert werden.


### 5.2 Retrieval-Experimente mit Content-Addressable Networks

Content-Addressable Networks [Ra01] waren die ersten mehrdimensionalen Indexstrukturen für P2P-Netze. Sie erlauben es Schlüssel/Wert-Paare abzulegen. Jeder Schlüssel ist hierbei ein $\delta$-dimensionaler Vektor $\vec{v}$ aus dem $\delta$-dimensionalen Einheitshyperwürfel $\vec{v} \in [0; 1]^\delta$.

In einem CAN ist jeder Peer für eine achsenparallele, quaderförmige Region des Ein-
heitswürfels zuständig, d.h. alle Schlüssel/Wert-Paare \((\vec{v}, x)\), die im CAN indexiert sind, werden im für \(\vec{v}\) zuständigen Knoten gespeichert. Jeder Peer hält Verbindung zu denjenigen Peers, die für angrenzende Regionen zuständig sind. In ihrer ursprünglichen Form ermöglichen CANs effiziente (exakte) Membership-Anfragen. In einem \(\delta\)-dimensionalen CAN mit \(N\) Knoten müssen \(O(\sqrt{N})\) Routingschritte durchgeführt werden, um eine Membership-Anfrage zu beantworten\(^3\).

Ähnlich wie [TXM02, GYGM04] nutzen wir hier Erweiterungen von CANs für Ähnlichkeitsanfragen. Hierzu sind Designentscheidungen zu fällen, die im Wesentlichen die sogenannte Splitstrategie und die Anfragebearbeitung an sich betreffen.

Neue Peers gliedern sich in Standard-CANs sofort in das Netzwerk ein. In unseren Experimenten gehen wir zur besseren Vergleichbarkeit davon aus, dass sich neue Peers zunächst in einer Warteschlange einreihen. Beim Einfügen eines Schlüssel/Wert-Paares wird jeweils der für den neuen Vektor \(\vec{v}_n\) zuständige Peer \(p_{v_n}\) bestimmt. Enthält er mehr als \(n_{\text{split}}\) Schlüssel/Wert-Paare, so wird der \(p_{v_n}\) zugeordnete Teilraum aufgeteilt. Dazu wird ein neuer Peer \(p'\) aus der Warteschlange entfernt und in das CAN eingegliedert. Die Performance des CANs hängt nun entscheidend von der Splitstrategie ab, d.h. der Auswahl der Dimension, entlang derer die Zuständigkeitsregion von \(p_{v_n}\) in zwei Zuständigkeitsregionen aufgeteilt wird.

Bei einer exakten Anfrage ist sicher, dass sich der gesuchte Schlüssel \(\vec{q}\) in exakt einem Peer befindet, nämlich dem Peer, der für \(\vec{q}\) zuständig ist. Bei Ähnlichkeitsanfragen hingegen ist jedoch nicht einmal sicher, dass der zum Anfragevektor \(\vec{q}\) ähnlichste Vektor in dem Peer \(p_q\) zu finden ist, der für \(\vec{q}\) zuständig ist. Es sind also von \(p_q\) ausgehend Peers zu suchen, die die \(k\) nächsten Nachbarn enthalten. Wir haben sowohl die Splitstrategie als auch die Methode der Anfragebearbeitung so gewählt, dass CANs bezüglich der von uns gemessenen Eigenschaften möglichst gut abschneiden.

**Splitstrategie:** Anders als [GYGM04] verwenden wir eine datenabhängige Splitstrategie. In dem zu splittenden Knoten \(p_s\) werden entlang jeder Dimension Mittelwert und Varianz der in \(p_s\) enthaltenen Daten berechnet [HSW89]. Als Splitdimension \(i_s\) wird die Dimension mit der höchsten Varianz gewählt und die Kollektion entlang dieser Dimension so aufgeteilt, dass diejenigen Punkte \(\vec{v}\), deren \(i_s\)-te Komponente \(v_{i_s}\) kleiner dem Mittelwert ist, in \(p_s\) verbleiben. Die anderen Schlüssel/Wert-Paare werden in den neuen Peer migriert.

**Anfragebearbeitung:** Die Anfragebearbeitung besteht aus zwei Schritten. Zunächst muss der für den Anfragevektor zuständige Peer \(p_q\) gefunden werden. Dann müssen von \(p_q\) ausgehend eventuell Nachbarn (Kandidaten-Peers \(p_{c}\)) kontaktiert werden, die einen oder mehrere der \(k\)-NN von \(\vec{q}\) enthalten könnten. Hierzu verwaltet \(p_q\) eine Prioritätswarteschlange, die Punkte und Regionen, geordnet nach ihrer Entfernung zu \(\vec{q}\), enthält [HS99]. Wird ein Punkt aus der Warteschlange gezogen, so ist er ein \(k\)-NN von \(\vec{q}\). Wird eine Region gezogen, so könnte sie einen \(k\)-NN enthalten. Der zuständige Peer wird kontaktiert. Er sendet eine Liste von Punkten und Regionen an \(p_q\). Dieser sortiert sie anschließend in die Prioritätswarteschlange ein. Das Verfahren wird solange fortgesetzt, bis die \(k\)-NN gefunden sind, oder die Warteschlange leer ist.

---

\(^3\) Small-world-CAN-Varianten, die logarithmische Komplexität bieten, existieren, wie z.B. [GYGM04]. Für die von uns betrachteten hochdimensionalen Räume ist die hier zu erwähnende Ersparnis allerdings nicht relevant.
Abbildung 5 zeigt auf der linken Seite einen Vergleich zwischen dem von uns bisher betrachteten StableSortRanker für PlanetP-artige Netze und unserer CAN-Implementierung für HSV36q unter Verwendung der Euklidischen Distanz. Sollen mehr als 80% der Top-20-Dokumente gefunden werden, verhält sich die CAN-Implementierung besser als StableSort(0).


5.3 Experimente mit menschlicher Relevanzbeurteilung

Bisher haben wir Messungen durchgeführt, bei denen die im zentralen Fall erzielten Ergebnisse als Benchmark verwendet wurden. Um diese Betrachtungsweise zu ergänzen werden wir nun Experimente vorstellen, bei denen Relevanzurteile von Experten als Benchmark genutzt werden. Bei der Relevanzbeurteilung wird eine Pooling-Strategie (vgl. [JvR75]) eingesetzt. Der Pool setzt sich je Anfrage aus den Top-N-Retrieval-Ergebnissen, die mit \(n\) verschiedenen Retrieval-Systemen ermittelt wurden, zusammen. Alle Dokumente dieses Pools werden von Experten nach Relevanz bzgl. der Anfrage beurteilt. Dokumente, die nicht im Pool enthalten sind, werden als irrelevant angesehen. Als Anfragen verwen-
Abbildung 6: Kontaktierte Peers vs. relevante Bilder gefunden (Feature: HSV36q)


Die 17 Anfragen wurden jeweils 30-mal für verschieden gewählte Cluster-Zentroide zur Histogrammerstellung ausgeführt, da die zufällige Wahl der Cluster-Zentroide das Ergebnis beeinflusst. Aus der Menge der möglichen Feature-Distanz-Kombinationen wählen wir die signifikantesten Ergebnisse. Abbildung 6 zeigt den relativen Recall unseres P2P-Systems bei Verwendung von HSV36q. Dieses Feature zeigt bei der Evaluierung zusammen mit vielen Distanzen ein besseres Verhalten als andere Features. Außerdem erweist es sich in Abbildung 2 besser als etwa HSV166q. Es weist darüber hinaus in Abb. 3 (li.) in Kombination mit \( \text{dist}_{\text{cos}} \) und \( \text{dist}_{\text{match}} \) ein ähnlich gutes bzw. besseres Verhalten als die Kombination von Farbmomenten und Manhattan-Distanz in Abb. 2 (li.) auf. Diese Kombination aus \( \text{dist}_{L_1} \) und Farbmomenten, die bei der Messung in Abb. 2 (li.) sehr gut abschneidet, fällt jedoch bei der Betrachtung des relativen Recalls klar hinter die besten Ergebnisse, wie sie in Abbildung 6 dargestellt sind, zurück.

Insbesondere das Kosinusmaß sowie die Euklidische Distanz zeigen in Abb. 6 ein besseres Verhalten als die anderen Distanzen. Um etwa 80% aller relevanten Dokumente zu finden, werden bei Verwendung von \( \text{dist}_{\text{cos}} \) bzw. \( \text{dist}_{L_2} \) lediglich etwa 12% der Peers kontaktiert. Insgesamt scheint die Kombination HSV36q im Zusammenspiel mit dem Kosinusmaß geeignet zu sein, da sie sowohl bei den Experimenten auf Basis der globalen Top-20-Dokumente als auch im Rahmen der Evaluatorung mit Relevanzbeurteilungen mit die besten Ergebnisse liefert. Die Verwendung der Euklidischen Distanz scheint bei Betrachtung von Abbildung 3 (li.) und Abbildung 6 ebenfalls berechtigt.
6 Zusammenfassung und Ausblick


Literatur


Kosten und Nutzen von Datenbankreorganisationen: Grundlagen, Modelle, Leistungsuntersuchungen

Stefan Dorendorf
Berufsakademie Gera /
Friedrich-Schiller-Universität Jena – Institut für Informatik
Lehrstuhl für Datenbanken und Informationssysteme
Ernst-Abbe-Platz 1-4
07743 Jena
Stefan.Dorendorf@informatik.uni-jena.de


1 Einleitung


398

In der Praxis sind diese Probleme nicht neu und Datenbankadministratoren sind gezwungen (oft pragmatisch), Lösungen zu finden, bei denen die Beeinflussung des normalen Datenbankbetriebs während einer Reorganisation möglichst gering ist. Auf Grund

- immer größer werdender Datenbanken auf der einen Seite und hoher Verfügbarkeitsanforderungen auf der anderen Seite,
- größerer Freiheitsgrade beim physischen Datenbankentwurf durch neue Speicherungskonzepte (objektrelational, XML usw.) bzw. durch neue Kombinationsmöglichkeiten und
- der Tatsache, dass Wartungsarbeiten wegen des damit verbundenen teilweise hohen Aufwands nicht einfach „auf Verdacht“ (online oder offline) ausgeführt werden können [Sch04]


2 Stand in Wissenschaft und Technik

Das Thema Reorganisation von physischen Speicherungsstrukturen, egal ob im Zusammenhang mit Datenbanken oder auch Dateisystemen, wurde in der Forschung in den vergangenen Jahren immer wieder mit unterschiedlicher Intensität und Zielstellung behandelt. Die genannten Beiträge stellen dabei nur eine beispielhafte Auswahl dar.

Einige ältere Beiträge befassen sich zunächst mit dem Begriff Datenbankreorganisation und dem Herausarbeiten von Ebenen, auf denen Datenbankreorganisationen ansetzen
können [NF76, SG79], sowie mit der Bestimmung von Reorganisationszeitpunkten bzw. der Festlegung von Reorganisationsintervallen [BG82, Tue78].


Weiterhin werden Möglichkeiten behandelt, durch die Verwendung geeigneter Speicherungsstrukturen und -konzepte auf der Definitionsebene den Aufwand für die Abarbeitung einer erwarteten Workload möglichst gering zu halten [AON96, GBG04, Omi89, SWS+05]. Dabei werden oft auch Verfahren zur Konvertierung der Strukturen beschrieben.

Das Thema der Umstellung von Datenbankschemata (Restrukturierung, bzw. Schemaevolution) wird in der Literatur ebenfalls ausführlich behandelt, soll hier nicht weiter aufgegriffen werden, da sich der Schwerpunkt der dortigen Betrachtungen wesentlich von dem des vorliegenden Beitrags unterscheidet.


3 Reorganisationsgründe und Reorganisationsmethoden

3.1 Degenerierungen


Datenlöschungen oder tupelverkürzende Änderungsoperationen verursachen eingestreute Freiplatzfragmente in Datenbereichen und zu dünn besetzten Indexknoten. Der freigewordene Platz wird oft nicht oder nicht sofort wieder benutzt. Dies bewirkt eine Verschlechterung der Speicherauslastung und eine Erhöhung des I/O-Aufwands bei Suchoperationen, da die Freiplatzfragmente mit gelesen werden müssen.


3.2 Reorganisationsmethoden


Reorganisationsmethoden

(I) zyklisch stattfindende Reorganisation
(II) parallele Reorganisation

Aus- und Wiedereinlagern
Reorganisation in eine Kopie
Reorganisation am Ort (in place)
kontinuierliche Reorganisation
inkrementelle Reorganisation

Abbildung 1: Reorganisationsmethoden

Bei der Methode I.1 wird zunächst der Inhalt der zu reorganisierenden Datenbankobjekte aus der Datenbank ausgelagert (entladen bzw. exportiert). Danach werden die Datenbankobjekte zerstört und wieder neu angelegt. Anschließend werden die zuvor ausgelagerten Daten wieder eingelagert (importiert bzw. geladen). Nach dem Auslagern der Daten stehen diese allerdings bis nach Abschluss des Wiedereinlagerns nicht für Anwendungsprozesse zur Verfügung. Diese relativ starke Einschränkung steht oft im Widerspruch zu hohen Verfügbarkeitsanforderungen. Weiterhin unterliegen die Daten in dieser Zeit – also außerhalb der Datenbank befindlich – auch nicht den Integritätsicherungsmechanismen des DBMS.

Bei Methode I.2, die bspw. von Oracle oder DB2 angewendet wird, werden die Daten intern an einen anderen Speicherort umkopiert. Ist die Methode als Online-Methode implementiert, so werden nach dem Kopieren der Daten zwischenzeitlich an den „Originaldaten“ vorgenommene Veränderungen auch auf die Daten der Kopie angewendet. Lediglich zum Abschluss der Reorganisation werden Benutzerzugriffe auf die Reorganisationskandidaten kurzzeitig unterbunden und die Originale der Datenbankobjekte werden durch die reorganisierten Kopien ersetzt.

Das wesentliche Merkmal der Methode I.3 ist, dass die die Tupel einer Tabelle darstellenden Sätze bzw. Indexeinträge einzeln innerhalb der bereits von der Tabelle belegten Speicherbereiche (quasi am Ort bzw. in place) so umkopiert werden, dass Freispeicherlücken wieder gefüllt oder angestrebte interne Sortierreihenfolgen wieder erreicht werden.

Bei einer parallelen Reorganisation kann die Wartung der Speicherungsstrukturen durch einen kontinuierlichen arbeitenden Reorganisationsprozess parallel zum laufenden Datenbankbetrieb durchgeführt werden (II.1). Bei der inkrementellen Methode (II.2) erfolgt die Reorganisation nach und nach. Sie wird in kleinen Einheiten durchgeführt und der aktuell vom Reorganisationsprozess bearbeitete Teil des Datenbestands wird für Zugriffe von Benutzertransaktionen gesperrt, während der Rest voll verfügbar bleibt.
4 Auswahl von Reorganisationskandidaten

4.1 Grundlegende Betrachtungen


Zur Bereitstellung der Informationen über die Workload stehen prinziell zwei unterschiedliche Varianten zur Verfügung. Ein Weg ist die manuelle Erstellung einer Workload-Beschreibung. Hier besteht aber das Problem, dass bei Datenbanken, die von unterschiedlichsten Anwendungen genutzt werden, nur mit einem erheblichen Aufwand ein Gesamtüberblick über die Operationen (SQL-Statements, Häufigkeiten usw.) aller Anwendungen erreicht werden kann, der dann in die Workload-Beschreibung einfließt. Eine andere mögliche Vorgehensweise stellt die Protokollierung der gegen die Datenbank gerichteten Anweisungen durch den Query-Prozessor oder eine aufgesetzte Komponente in einem repräsentativen Zeitraum dar. Hier besteht zwar die Gefahr, dass
das Protokoll einen erheblichen Umfang erreicht, allerdings existieren auch Arbeiten, die sich mit diesem Problem auseinander setzen und Lösungsmöglichkeiten präsentieren (z.B. [CGN02]). Eine einfache, aber durchaus wirkungsvolle Methode besteht bereits darin, die ausgeführten SQL-Anweisungen nur einmal im Protokoll zu speichern und zusätzlich einen Zähler mitzuführen, der die Häufigkeit der Ausführungen angibt. Abhängig davon, welche Datenbankobjekte (z.B. einzelne Tabellen oder alle Tabellen eines Table Space usw.) im Rahmen der Analyse betrachtet werden sollen, schließt sich an die Protokollierung evtl. noch eine Aufbereitung des Protokolls an, bei der nicht relevante Anweisungen entfernt werden. Die entstandene Workload-Beschreibung kann dann fortan verwendet werden, solange keine signifikanten Änderungen im Operations-Mix auftreten.


4.2 Quantifizierung des Nutzens einer Reorganisation


Während der im Rahmen von [Dor06] angestellten Untersuchungen wurde (quasi mangels Alternativen) ein eigenes I/O-Kostenmodell verwendet, das auch die

Bei der Bestimmung des Nutzens werden zunächst je Planoperator \( P \) über eine zugehörige Kostenfunktion die Kosten, die vor einer Reorganisation anfallen \( K_{P\text{vor}} \) und der jeweils durch vorhandene Degenerierungen verursachte prozentuale Mehraufwand \( M_{P} \) ermittelt. In Abbildung 2 ist die Arbeitsweise eines Index Range Scan dargestellt.

Abbildung 2: Blockzugriffe bei einem Index Range Scan

Die Anzahl notwendiger Blockzugriffe \( (\text{Gleichung } 1) \) ergibt sich hier zunächst aus der Summe der Anzahl Indexebenen \( (I_{\text{LEV}} \text{ – ohne Blattebene}) \) und der Anzahl Blätter des Indexbaums, die entlang der Verkettung durchlaufen werden müssen \( (I_{\text{LEAF}} \text{ bzw. ein Teil davon}) \). Über die in den Blättern des Indexbaums gespeicherten Verweise kann dann auf die einzelnen Tupel zugegriffen werden. Dabei müssen eventuell vorhandene Zeiger auf migrierte Tupel verfolgt werden \( (\text{Zugriff Nr. 4 in Abbildung } 2) \), deren Anzahl über \( A \) berücksichtigt wird. Weiterhin werden Informationen über den Grad der Übereinstimmung der Ordnung eines Index und der Sätze, auf die die Index-Einträge verweisen \( (\text{Clustering Factor} – I_{\text{CLUS}}) \), benötigt. Hier wird in den Statistikdaten von DBMS oftmals direkt angegeben, wie oft bei einem Index Scan über die gesamte Tabelle der Datenblock wechselt, in der der nächste zu lesende Satz gespeichert ist. Der Wert steht also zwischen „fast gar nicht springen” und „extremen Springen“.

405
Selektivität) gibt den Anteil des Suchbereichs an der Gesamtdatenmenge an, über den der Index Scan ausgeführt wird. Die Schätzung solcher Selektivitäten kann unter Nutzung von im Datenbankkatalog hinterlegten Informationen zu Werteverteilungen von Schlüsseln erfolgen. Die Werte \( I_{BRi} \), \( I_{BRh} \) sowie \( P_{BR} \) stehen für die Wahrscheinlichkeiten, dass die angeforderten Blöcke im Puffer gefunden werden (Pufferungsgrade). Bei B- und B*-Baum-Indexen werden im eInformationsschema diese Pufferungsgrade einzeln für jede Indexebene geführt. In [Kli05] wurde eine Methode entwickelt und prototypisch für Oracle 10g implementiert, die die Ermittlung solcher feingranularen Pufferungsgrade ermöglicht\(^1\). Deren Berücksichtigung gestattet eine Bestimmung des Nutzens von Datenbankreorganisationen bezüglich der erreichbaren Reduzierung von physischen I/O-Operationen (Zugriffe auf Datenträger).

Der in den Kosten für die jeweiligen Planoperatoren enthaltene I/O-Mehraufwand in Prozent kann planoperatorspezifisch errechnet werden. Für den betrachteten Index Range Scan kann der enthaltene Mehraufwand wie in Gleichung 2 gezeigt ermittelt werden.

\[
K_{\text{Pvor}} = \sum_{\text{Ebene}} \left( 1 - I_{BR} \right) + \max \left( I_{\text{LEAF}} \cdot \left( 1 - I_{BR} \right), \left( 1 - I_{BR} \right) \right) + \left( I_{\text{CLUST}} + A \right) \cdot \left( 1 - P_{BR} \right) \cdot S \quad \text{Gleichung 1}
\]


\[
K_{\text{Pnach}} = \frac{K_{\text{Pvor}}}{1 + \frac{M_t}{100}} \quad \text{Gleichung 3}
\]

Der in den Kosten für die jeweiligen Planoperatoren enthaltene I/O-Mehraufwand in Prozent kann planoperatorspezifisch errechnet werden. Für den betrachteten Index Range Scan kann der enthaltene Mehraufwand wie in Gleichung 2 gezeigt ermittelt werden.

\[
M_t = \sum_{\text{Ebene}} \frac{A \cdot \left( 1 - P_{BR} \right)}{\left( 1 - I_{BR} \right) + I_{\text{LEAF}} \cdot \left( 1 - I_{BR} \right) + I_{\text{CLUST}} \cdot \left( 1 - P_{BR} \right)} \quad \text{Gleichung 2}
\]

Sind die vor der Reorganisation anfallenden Kosten und der darin enthaltene Mehraufwand bekannt, so können die zu erwartenden Kosten \( K_{\text{Pnach}} \) für den Planoperator nach der Reorganisation über

\[
K_{\text{Pnach}} = \frac{K_{\text{Pvor}}}{1 + \frac{M_t}{100}} \quad \text{Gleichung 3}
\]


Durch die Summierung der durch die einzelnen Planoperatoren vor bzw. nach einer Reorganisation verursachten Kosten können die Gesamtkosten (bezogen auf die Anzahl

\(^1\) Die Pufferungsgrade beziehen sich auf einzelne Tabellen sowie auf die einzelnen Ebenen von Index-Bäumen. Bei der Nutzung von Partitionierungskonzepten können diese Informationen auch für die einzelnen Partitionen gesammelt werden. Damit können zumindest Überprüfungen des Kostenmodells durchgeführt werden. Für einen Einsatz großen Produktivumgebungen muss allerdings die Effizienz der aufgesetzten realisierten Methode noch verbessert werden.
notwendiger Blockzugriffe) vor ($K_{vor}$) und nach der Reorganisation ($K_{nach}$) ermittelt werden. Der Nutzen der Reorganisation ($N$) kann anschließend mit

$$N = \left(1 - \frac{K_{nach}}{K_{vor}}\right) \times 100$$

berechnet werden. Er entspricht der prozentualen Reduzierung der I/O-Kosten zur Abarbeitung der Workload durch die Datenbankreorganisation (Reorganisationsnutzen).

### 4.3 Abschätzung der Reorganisationskosten


Abbildung 3: Verschieben von Datensätzen zur Beseitigung von eingestreutem Freiplatz


Für diese Implementierung lässt sich der anfallende Aufwand relativ einfach bestimmen, wenn die Zahl der zu verschiebenden Datensätze bekannt ist. Dazu muss zunächst die Zahl der nach der Reorganisation belegten Datenblöcke \( P_{\text{IDEAL}} \) näherungsweise bestimmt werden. Vorgehensweisen bei solchen Speicherplatzabschätzungen sind üblicherweise für DBMS-Produkte beschrieben. Die Differenz zur aktuell belegten Anzahl Datenblöcke \( P_{\text{USED}} \) entspricht der Anzahl frei werdender Blöcke. Damit kann über Gleichung 5 die Anzahl der zu verschiebenden Datensätze geschätzt werden.

\[
T_{\text{MOVE}} = \left( P_{\text{USED}} - P_{\text{IDEAL}} \right) \frac{P_{\text{ANZT}}}{P_{\text{USED}}} 
\]  

Gleichung 5

Durch die Multiplikation der Zahl zu verschiebender Sätze mit dem bei Lösch- und Einfügeoperationen anfallenden Aufwand können die Kosten für die Reorganisationsdurchführung berechnet werden (Gleichung 6).

\[
C_{\text{Reorg}} = (C_{\text{insert}} + C_{\text{delete}}) \cdot T_{\text{MOVE}} 
\]  

Gleichung 6

Genauere Ausführungen zur Abschätzung der Kosten für Einfüge- (\( C_{\text{insert}} \)) und Löschoperationen (\( C_{\text{delete}} \)) finden sich in [Dor06, Wie05].

4.4 Auswahl von Reorganisationskandidaten


Die Problemstellung entspricht dabei einem aus der mathematischen Optimierung bekannten Rucksackproblem. Die einzelnen Reorganisationskandidaten stellen die

Die zum Packen des Rucksacks verwendbaren Gegenstände sind *unteilbar* und stehen jeweils nur einmal zur Verfügung. Auf die Reorganisationsproblematik übertragen bedeutet die „Unteilbarkeit“, dass sich das Reorganisationsgranulat nach dem bei den Kosten- und Nutzenanalysen verwendeten Granulat richtet. Wurden bspw. bei einer solchen Analyse eine Tabelle und die ihr zugeordneten Indizes als Einheit betrachtet, so wird für die Optimierung angenommen, dass eine Reorganisation auch auf die Tabelle und die ihr zugeordneten Indizes angewendet wird. Sollen evtl. auch nur einzelne Indizes reorganisiert werden, so müssen die Kosten- und Nutzenanalysen von vorn herein einzeln für die Indizes und (evtl.) den Datenbereich der Tabelle durchgeführt werden.

Bei der Ermittlung der *Nutzenwerte* werden die einzelnen Reorganisationskandidaten als unabhängig angesehen. Die Reorganisation eines Kandidaten zur Beseitigung von Degenerierungen hat keinen Einfluss auf den durch die Reorganisation eines anderen Kandidaten erreichbaren Nutzen. Würde eine Reorganisation allerdings zur Veränderung der physischen Repräsentation von Datenbankobjekten durchgeführt, so könnten (bspw. bei hierarchisch strukturierten Objekten) hier durchaus Abhängigkeiten existieren. Dieser Fall soll hier aber nicht weiter betrachtet werden.


unter den gegebenen Umständen den Verfügbarkeitsanforderungen und Forderungen nach geringen Kosten bestmöglich entspricht. Der Kostenwert für die ausgewählte Methode fließt dann in das Optimierungsverfahren ein.

5 Überprüfung in einer Beispielumgebung

Zu Überprüfungszwecken wurde das in Abbildung 4 dargestellte Tabellenschema unter Oracle 10g implementiert. Hier wird ein vereinfachter Ausschnitt der Daten modelliert, die zur Abwicklung von mobiler Kommunikation im Global System for Mobile Communications (GSM) benötigt werden.

Im sog. Home Location Register (HLR), einer zentralen Datenbank des Mobilfunkanbieters, werden die Daten über jeden Mobilfunkteilnehmer gespeichert. Dazu gehören u.a. die eigentlichen Teilnehmerdaten, Daten über die Dienste, die vom Mobilfunkanbieter zur Verfügung gestellt und von den Teilnehmern genutzt werden können sowie die Daten über geführte Anrufe, um die entsprechenden Abrechnungen zu erstellen. Im Unterschied zu Festnetzanschlüssen sind die Teilnehmer im Mobilfunkbereich frei beweglich. Diesem Aspekt wird dadurch Rechnung getragen, dass temporär Routing-Daten gespeichert werden, über die mit einer sog. MSISDN\(^2\) die für die Funkschnittstelle zum Mobiltelefon notwendigen Daten (LAI, TMSI)\(^3\) zugeordnet werden können. Die Tabellen wurden im Beispiel mit per Zufallszahlengenerator erzeugten Daten gefüllt. Anschließend wurden verschiedene Änderungsoperationen auf die Beispieldatenbank angewendet, um Degenerierungen (wie eingestreuten Freiplatz, migrierte Tupel und nicht eingehaltene interne Sortierreihenfolgen) zu erzeugen.

\(^2\) Die MSISDN ist die „normale“ Telefonnummer, die gewählt werden muss, um einen Teilnehmer zu erreichen.

\(^3\) Die Location Area Identity (LAI) und die Temporary Mobile Station Identity (TMSI) dienen zur Identifizierung der Mobilfunkzelle, in der sich der Teilnehmer aufhält und zur Identifizierung des Mobiltelefons in der Zelle.
Aus den in Abbildung 5 dargestellten SQL-Anweisungen wurden vier unterschiedlich zusammengesetzte Beispiel-Workloads mit je 5000 Anweisungen generiert, die anschließend auf die Tabellen angewendet wurden.

Abbildung 5: Anweisungen zur Generierung von Beispiel-Workload

Dabei werden in den verschiedenen generierten Workloads nicht immer alle Anweisungen verwendet. Enthalten sind

- in der ersten Workload die Anweisungen 0 bis 6,
- in der zweiten Workload die Anweisungen 0 und 1,
- in Workload Nr. 3 die Anweisungen 0, 4, 5 und 6 und
- in der vierten Workload alle acht aufgeführten Anweisungen.


In der Beispielumgebung werden die Anweisungen aufgrund entsprechender Entscheidungen des kostenbasierten Oracle-Optimierers wie folgt realisiert:

- Bei Anweisung 0 wird ein Index Lookup übergeführt.
- Die Anweisungen 1 und 2 führen jeweils eine sequenzielle Suche über den entsprechenden Tabellen aus.
- Anweisung 3 wird über einen Index Lookup realisiert.
- Anweisung 4 wird mittels eines Nested Loop Joins umgesetzt. Dabei wird zunächst über einen Index Lookup auf die Tabelle TEILNEHMER zugegriffen. Danach wird über einen Index Range Scan die Verbindung zur Tabelle ANRUFEB zugegriffen.
- Für Anweisung 5 werden die Tabellen TEILNEHMER und_ROUTING zunächst sequenziell gelesen und danach über einen Hash Join verbunden.
- Bei Anweisung 6 wird zunächst über einen Index Lookup eine Selektion auf der Tabelle TEILNEHMER durchgeführt. Das Ergebnis wird über einen Nested Loop Join mit der Tabelle TNDIENST verbunden, auf die mittels sequenziellem Suchen zugegriffen wird. Über einen weiteren Nested Loop Join wird anschließend die
Verbindung zur Tabelle DIENSTE hergestellt, auf die mittels Index Lookup zugegriffen wird.

- Anweisung 7 wird über einen Index Range Scan über der Tabelle TEILNEHMER realisiert.


Abbildung 6: Beispielrechnung für Workload 2


Die Abweichungen der errechneten Werte für physische Blockzugriffe (jeweils dritte Säule) von den gemessenen Werten (jeweils vierte Säule) sind etwas größer als bei den

Abbildung 7: Vergleich errechneter und gemessener Nutzen von Datenbankreorganisationen bei gleichem Degenerierungsgrad und unterschiedlicher Workload-Zusammensetzung

Zur Überprüfung des Berechnungsschemas zur Schätzung der bei In-Place-Reorganisationen anfallenden Kosten wurden (ebenfalls unter Nutzung von Oracle 10g) Untersuchungen durchgeführt und Messreihen aufgenommen. Basis war die Tabelle TEILNEHMER, die zunächst mit Beispieldaten geladen wurde. Anschließend wurden durch die Ausführung von DELETE-Anweisungen über die gesamte Tabelle hinweg Freispeicherfragmente erzeugt. Über den Attributen MSISDN, Name und Ort wurde jeweils ein Index angelegt. Alle Indexe besaßen während der Messreihen eine Höhe von 3 Ebenen. Der durchschnittliche Pufferungsgrad von Indexblöcken lag während der Messreihen bei ca. 98%, der der Datenblöcke bei ca. 82%. Die von Oracle angewendete asynchrone Ausführung von Schreiboperationen führte dazu, dass in der Beispielumgebung ca. 6,7 Änderungen an Indexblöcken bzw. ca. 4,3 Änderungen an Datenblöcken jeweils zu einer physischen Schreiboperation führten. Tabelle 1 zeigt einen Vergleich des rechnerisch ermittelten und des gemessenen physischen I/O-Aufwands zur Durchführung einer In-Place-Reorganisation der Tabelle TEILNEHMER bei verschiedenen Anteilen von eingestreutem Freiplatz und damit verschiedenen Anzahlen zu verschiebender Sätze.

Die gemessenen Aufwandswerte liegen dabei bei der ersten Messreihe etwas über und bei der zweiten Messreihe etwas unter den errechneten Werten. Dies ist darauf zurückzuführen, dass die Durchschnittswerte beider Messreihen in den Berechnungen für die Pufferungsgrade und die Anteile tatsächlich erfolgter Schreiboperationen eingeflossen sind. Durch die größere Anzahl zu verschiebender Sätze im Rahmen der zweiten Messreihe ergibt sich da auch eine höhere Lokalität der Zugriffe als bei der
ersten Messreihe. Durch die höhere Lokalität können anteilig mehr Zugriffe im Puffer erfolgen. Bei der ersten Messreihe liegt die Lokalität unter der, die sich aus den Durchschnittswerten für Pufferungsgrade und Schreiboperationen ergibt. Detailliertere Ausführungen zu den durchgeführten Messreihen finden sich in [Dor06].

<table>
<thead>
<tr>
<th>ca. 28 400 zu verschiebende Sätze (Freiplatzanteil ca. 27%)</th>
<th>ca. 42 300 zu verschiebende Sätze (Freiplatzanteil ca. 41%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>errechnete Werte</td>
<td>gemessene Werte</td>
</tr>
<tr>
<td>gelesene Datenblöcke</td>
<td>5112</td>
</tr>
<tr>
<td>geschriebene Datenblöcke</td>
<td>6532</td>
</tr>
<tr>
<td>gelesene Indexblöcke</td>
<td>1704</td>
</tr>
<tr>
<td>geschriebene Indexblöcke</td>
<td>12695</td>
</tr>
<tr>
<td>Summen</td>
<td>26043</td>
</tr>
</tbody>
</table>

Tabelle 1: Gegenüberstellung von errechnetem und gemessenem I/O-Aufwand

6 Zusammenfassung und Ausblick


Neben dem Nutzen müssen bei der Entscheidung über die Durchführung von Datenbankreorganisationen auch die durch sie verursachten Kosten berücksichtigt werden. Wie die Ermittlung solcher Kosten erfolgen kann, wurde am Beispiel der für das DBMS-Produkt Oracle implementierten In-Place-Reorganisationsfunktionalität dargestellt.

Anhand von durchgeführten Messreihen wurde die Funktionstüchtigkeit der vorgeschlagenen Methoden zur Kosten- und Nutzenermittlung gezeigt.

Basierend auf der Möglichkeit der näherungsweisen Vorhersage von Kosten und Nutzen von Datenbankreorganisationen wurde ein Ansatz zur Unterstützung von Datenbankadministratoren bei der Auswahl von Reorganisationskandidaten vorgestellt, der auf die Maximierung des von einer Datenbankreorganisation zu erwartenden Nutzens unter Berücksichtigung einer Kostenobergrenze für die Reorganisation abzielt.

**Literaturverzeichnis**

- [IBM02] *IBM DB2 Universal Database Command Reference Version 8*. International Business Machines Corporation, 2002
Armada: a Reference Model for an Evolving Database System

Fabian Groffen  Martin Kersten  Stefan Manegold
Centrum voor Wiskunde en Informatica
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
{Fabian.Groffen,Martin.Kersten,Stefan.Manegold}@cwi.nl

Abstract: The data on the web, in digital libraries, in scientific repositories, etc. continues to grow at an increasing rate. Distribution is a key solution to overcome this data explosion. However, existing solutions are mostly based on architectures with a single point of failure.

In this paper, we present Armada, a model for a database architecture to handle large data volumes. Armada assumes autonomy of sites, allowing for a decentralised setup, where systems can largely work independently. Furthermore, a novel administration schema in Armada, based on lineage trails, allows for flexible adaptation to the (query) workload in highly dynamic environments. The lineage trails capture the metadata and its history. They form the basis to direct updates to the proper sites, to break queries into multi-stage plans, and to provide a reference point for site consistency. The lineage trails are managed in a purely distributed fashion, each Armada site is responsible for their persistency and long term availability. They provide a minimal, but sufficient basis to handle all distributed query processing tasks.

The analysis of the Armada reference architecture depicts a path for innovative research at many levels of a DBMS. Challenging many conventional database assumptions and theories, it will eventually allow large databases to continue to grow and stay flexible.

1 Introduction

Soon we face a common repository size scaling into petabytes, filled with data that needs to be archived and processed. The rapidly improving technology cannot keep up with the data growth rate, hence data processing becomes more and more an expensive and time-consuming task. This problem is of major concern, since data processing is a core process for many businesses and applications. Yet a real solution to the data growth problem has to be found.

Scaling into multiple machines to process the data is currently successfully applied in grids and distributed databases. However, the centralised scaling technique using a large number of machines, is fragile from an availability point of view. All systems depend on the availability of one. Moreover, this single point-of-failure can easily get overloaded, thereby forming the bottleneck in serving a high workload.

The problem does not limit itself to large scale machines such as datagrid environments...
in scientific settings [LF04]. Also mobile and ambient settings deal with a data explosion problem. The ambient environment consists of a potentially large number of database-empowered sensory systems, which learn and exchange information to reach a common goal, e.g., increased experience without computers in sight [A+03]. Mobile environments are characterised by a large number of clients sharing information through multiple data brokers. The data ‘follows’ the device, which may be offline for lengthy periods [D+97].

Although in each environment the complexity can be controlled within the context of a single application and strict separation of roles, current distributed database offerings stem from an era where a limited number of always-on servers were prevalent. They lack functionality in a number of areas to provide a general solution. A novel reference architecture for a distributed database is urgently needed.

It should take site autonomy and volatility as both driving force and core feature of a system architecture. A sole central broker to guide all interactions is a dead end for the scalable solutions required. Instead, several sites may take such a role for a limited period and only for part of the data space. To co-ordinate their efforts, ‘contractual’ arrangements and economic models are needed. They should facilitate a ‘data market’ to ensure the desired system behaviour and be flexible to cope with temporal and evolutionary changes.

Of course, the setting is not completely new. It is the natural step forward in the area of highly distributed database technology. The underlying techniques are still based on data fragmentation and data replication to break the database into manageable portions, and query shipping versus data shipping for efficiency [OV99]. However, the volatile setting calls for better solutions to keep track of the data whereabouts, their status, and their lineage in the grand scheme. A portion of the database may be broken into pieces and migrated to autonomous sites with little control other than powerful re-conciliation algorithms when the pieces are fused in the future. The long liveness of such networks makes legacy of information, e.g., out of date schemas and queries, a ground rule rather than an exception. It calls for data management schemes optimised for incomplete and only partially consistent information.

Within the Armada project we study the building blocks for an organic database. The main contribution of this paper is a visionary description of the Armada model and its architecture, designed to facilitate evolutionary growth in a distributed environment. It uses data fragmentation, data replication and data fusion as the minimal basis for the lineage of data blocks, that allows maximal autonomy of the nodes co-operating in a distributed application.

Sites can easily join an Armada alliance by donating resources and taking responsibility for a portion of the data space prescribed in the database schema. They can also leave the alliance with minimal detrimental effect on its environment. The real size of the distributed system and data locality is largely hidden from individual nodes. The Armada administration allows for localisation of data without need for a central entity that becomes a bottleneck, single point-of-failure and hot-spot in busy systems.

The clients (applications) are put back into the loop to steer query processing and distributed transaction management. Common client policies can be captured in scenarios managed by a middleware software layer, but the autonomy of the Armada sites ultimately
relies on co-operative clients as well.

The remainder of the paper is structured as follows. Section 2 introduces the Armada model, its notation and operations. The realisation of the model in terms of an architectural overview is presented in Section 3. The effect of lineage trails on query processing is described in Section 4. A sample embedding of the Armada model in the trends in distributed database systems is indicated in Section 5. We conclude and give a short outlook in Section 6.

2 The Armada Model

The focus of this work is to create a reference model for a flexible, self-maintaining, efficient distributed database architecture. To achieve this goal, we try to avoid the classical bottlenecks that limit the efficiency of most existing and proposed architectures. These bottlenecks can be seen as the two extreme alternatives of storing and maintaining the metadata that is necessary to ensure correct and efficient handling of the actual data. Classical designs on the one end of the spectrum require a centralised server that holds all metadata, and hence forms a hotspot. The central server is accessed to lookup or update metadata for both operations that query/update the actual data and operations that change the structure of the system at large. The latter involves addition or removal of nodes and/or reorganisation of the data for load balancing purposes. It creates a bottleneck that limits the overall performance and scalability of such systems.

Designs on the opposite end of the spectrum avoid this hotspot by fully replicating all metadata. Such designs have to rely on the consistency of the replicated metadata, and hence, each structural change requires the (synchronised) update on all nodes in the system. Because all metadata is available locally, data operations are cheaper, but it significantly increases the price of structural operations, prohibiting efficient dynamic changes of the data distribution.

With the Armada model, we aim at finding a balance between these two extremes. On the one hand, Armada does not come with a centralised server, and thus avoids the bottleneck of metadata lookups. On the other hand, Armada does not require to replicate all metadata on all nodes. Instead, Armada finds a compromise by replicating metadata partially only, and being able to cope with incomplete or stale metadata. Obviously, each node holds its own local metadata, e.g., schema information about the portion of the database stored, and keeps it up-to-date. In addition, it holds some remote metadata, i.e., information from nodes in its vicinity. To limit maintenance overhead, the idea is to limit remote updates of metadata to those nodes that exchange data due to structural updates. Thus, remote metadata is not necessarily kept up-to-date at all times. Rather, an Armada-node assumes that its remote metadata is an approximation or a past snapshot of the situation of a remote node.

The inspiration for our novel reference model comes from the Armada analogy. An Armada is a fleet of ships, that forms a unity although each ship has a captain who is sovereign. The Armada model reflects this property in a minimal set of relations between
the captains of the ships. Each ship has cargo (data) stored in barrels (boxes) that are addressed by cargo documents (trails) kept by the captain. A captain can repack the cargo on his ship, and/or hand over (parts of) his cargo to one or more other ships in the Armada (cloning, chunking). Repackaging may also occur if barrels are empty or only partially used, such that multiple barrels are put in one (combining). The cargo documents describe the content of each barrel as well as the lineage of the respective cargo. A captain keeps one cargo document for each barrel he has aboard his ship. When handing over cargo to other ships, the respective cargo documents are duplicated; the original copy stays with the captain on the old ship and the other one is attached to the barrel on the new ship. Thus, each captain does not only know what cargo his ship currently carries, but also where he sent the cargo that he once had aboard, and where any cargo he ever transported came from. In fact, the cargo documents kept on each ship provide sufficient information to allow the captain to locate any cargo item in the whole Armada.

In the remainder of this section, we will briefly formalise the key components of the Armada model. We start by introducing the basic notation, terms, and definitions that make up the ‘static’ part of the Armada model, i.e., the part that is used to describe how the metadata is represented. After that, we proceed with the ‘dynamic’ part that models operations to perform structural changes on the Armada.

The goal of this work is to establish the Armada model as a generic framework for distributed database architectures. Hence, discussion of actual instantiations of the model, like strategies as to when, why and how to perform structural changes, is beyond the scope of this paper.

2.1 Notation, Terms and Definitions

We informally introduce the term (data) box to refer to the portion of the data that is hosted at a site. We assume that the content of a box can be described by an arbitrary function \( g \). The actual specification of such function is left to the instantiation of a specific Armada system. In the course of this section, we will provide some constraints for such functions. Section 3.2 will discuss these functions in more detail and give some simple examples.

Further, we use the term structural operations to refer to operations that create and modify the data distribution across sites, i.e., operations that replicate, (re-)fragment or merge portions of the data. Data boxes form the entities that these structural operations operate on.

**DEF. 1** Be \( B'_i, B'_{i+1}, \ldots, B'_{i+n} \) existing boxes in an Armada system with functions \( g'_i, g'_{i+1}, \ldots, g'_{i+n} \) describing the content of each box. A structural operation \( o \) operates on one or more boxes \( B'_i, B'_{i+1}, \ldots, B'_{i+n} \) and produces one or more new boxes \( B_j, B_{j+1}, \ldots, B_{j+m} \) with functions \( g_j, g_{j+1}, \ldots, g_{j+m} \) describing the content of these new boxes. A structural operation cannot generate new data, but must not “lose” any data, either. Hence, we

---

1The trail administration for each box is only valid at the time it is created. Afterwards, its references to successors may be outdated. For the site hosting the box this does mean, however, that it can reach the rest of the Armada through the sites it knows as stored in the trails, even though that might not be the most up-to-date state.
require that
\[ g_j \cup g_{j+1} \cup \cdots \cup g_{j+m} = g'_i \cup g'_{i+1} \cup \cdots \cup g'_{i+n} \].

Inspired by the cargo documents of the Armada analogy, we introduce \textit{lineage steps} and \textit{lineage trails} to store and administer metadata. A \textit{lineage step} captures the logistic information of applying a structural operation to a box:

- \( g \), the function that is applied (and hence describes the content of the new box),
- \( S \), the site that the new box is shipped to, and
- \( B \), the identifier of the new box (for the convenience of later reference).

\textbf{DEF. 2} A lineage step \( s = [g, S] : B \) is a composition that identifies the application of a structural operation, resulting in a new box \( B \) on site \( S \) with function \( g \) describing the content of the new box. The box \( B' \) that \( s \) is applied to is identified by the lineage trail \( T' \) that \( s \) is appended to (see below).

Each box in the Armada is uniquely identified by a \textit{lineage trail} that captures the whole history of its data.

\textbf{DEF. 3} A lineage trail, or trail for short, \( T = s_1.s_2.\cdots.s_l \) is a sequence of \( l \in \mathbb{N} \) lineage steps. With \( s_l = [g, S] : B \), \( T \) identifies box \( B \) on site \( S \).

\textbf{DEF. 4} Be \( B'', B' \), and \( B \) boxes on sites \( S'', S', S \) with their content described by functions \( g'', g', g \), respectively. Further be \( B'' \), \( B' \), and \( B \) identified by the trails \( T'' \), \( T' = T''.[g', S']: B' \) and \( T = T'.[g, S] : B \), respectively. We call

\[ T'' \]

a predecessor trail of box \( B' \),

\[ s' = [g', S']: B' \]

the local step of box \( B' \),

\[ T' = T''.s' \]

a local trail of box \( B' \),

\[ s = [g, S] : B \]

a successor step of box \( B' \),

and analogously for boxes \( B'' \) and \( B \).

The metadata maintained and stored for each box consists of \textit{a set} of predecessor trails, \textit{exactly one} local step, and \textit{a (possibly empty) set} of successor steps. The predecessor trails represent the box’ heritage. The local step describes the box itself, and the successor steps point to the box’ offspring. The predecessor trails and local step are set upon creation of a box, while the successor steps are only set once a box participates in a structural operation.

We assume that a structural operation (logically) removes all the data from its input boxes (transferring it to the newly created boxes), and destroys the input boxes. Only the respective metadata (lineage) is kept. This assumption relieves us from the need to consider different versions of each box, and thus helps to simplify the model. The assumption does not limit the generality of the model. In a practical implementation, this does not necessarily require a (physical) copy of all data with each structural operation. Instead, simply renaming the box can be sufficient.
To simplify the presentation, we will omit the set notation whenever a set of trails is empty or contains only one trail. In the first case, we simply omit the empty trails set; in the latter case, we depict the only element as singleton. Thus, the metadata for boxes $B''$, $B'$ and $B$ of Definition 4 is depicted as follows:

\[
\begin{array}{ccc}
\text{pre} & \text{loc} & \text{suc} \\
T'' = T''' & \{g'', S'' \} & B'' \\
T' = T''' & \{g', S' \} & B' \\
T = T' & \{g, S \} & B
\end{array}
\]

The set of successor steps is empty for all boxes to which no structural operation has been applied yet, i.e., all boxes that physically exist and store data. The set of predecessor trails is empty for one box in an Armada, the origin.

DEF. 5 An Armada instance is born as a single initial box $B_o$. We call $B_o$ the origin of the Armada instance. Obviously, the origin has no predecessor trails. Further, since no structural operation is applied to create the origin, there is no function that describes (restricts) $B_o$’s content. We indicate this by $\%$ in $B_o$’s local step:

\[
T_o = \{\%, S_o \} : B_o
\]

### 2.2 Structural Operations

To let an Armada evolve from the origin, we consider the following three structural operations.

**Replication: the clone operation**

DEF. 6 The **clone operation** operates on one box $B'$ with function $g'$ and generates one or more new boxes $B_1, \ldots, B_{j+m}$ that all contain a copy of $B'$’s data. Hence, their functions $r_j, \ldots, r_{j+m}$ are all identical to $g'$.

Replicating a data box is the action of copying its content to a new location. We call it the **clone operation**, denoted by function $r$. Consider the following example of cloning the origin box $B_o$:

\[
T_o = \{\%, S_1 \} : B_o ; \\
T_1 = \{r, S_1 \} : B_1 ; \\
T_2 = \{r, S_2 \} : B_2
\]

In this example, the origin has two successors, $B_1$ and $B_2$, which themselves have no successors.
Following Definition 6 the number of new boxes produced can also be a single one. Strictly, this is no cloning operation any more: since the original box is (logically) destroyed after the cloning, its data is not replicated, but rather moved to a single new location. However, there is no reason to prohibit this in the model.

Although we use different site identifiers for the two new boxes in the above example, it is perfectly sound with the model to produce two (or more) clones of a box on the same site. The question, whether this is reasonable in practice, is not relevant in the context of a reference model.

**Fragmentation: the chunk operation**

DEF. 7 The chunk operation operates on one box $B'$ with function $g'$ and generates one or more new boxes $B_j, \ldots, B_{j+m}$ that all contain a fraction of $B'$’s data. We require that all fractions are disjunct, but no data is lost, i.e., the following must hold for new boxes’ functions:

$$f_j \cup \cdots \cup f_{j+m} = g' \quad \text{and} \quad \forall k, l \in \{j, \ldots, j+m\}, k \neq l : f_k \cap f_l = \emptyset.$$

Fragmenting data means it gets spread out over multiple boxes. We call this the chunk operation, denoted by functions $f, f', f'', \ldots$. Consider the following example of chunking the origin box $B_o$:

$$T_o = [\% , S_1]: B_o ; \begin{cases} [f, S_1]: B_1 \\ [f', S_2]: B_2 \end{cases}$$
$$T_1 = [\% , S_1]: B_o ; [f, S_1]: B_1 ;$$
$$T_2 = [\% , S_1]: B_o ; [f', S_2]: B_2 ;$$

The origin has been chunked in two, using chunk functions $f$ and $f'$. Like with cloning, in case there is only one result box, a move operation is effectively being executed.

**Merging: the combine operation**

DEF. 8 The combine operation operates on one or more boxes $B'_i, B'_{i+1}, \ldots, B'_{i+n}$ with functions $g'_i, g'_{i+1}, \ldots, g'_{i+n}$ and produces a single new box $B$ that combines all the data of the input boxes. The produced box’ function $m$ spans the domain of $g'_i \cup g'_{i+1} \cup \cdots \cup g'_{i+n}$.

While cloning and chunking are growing operators, the combine operation is a shrink operation. Applying it to a number of boxes merges them into one. However, this operation is not restricted to acting as an inverse-operation to the clone and chunk operations, i.e., reconstructing a previously cloned or chunked box. Our model allows to apply the combine operation to an arbitrary set of boxes. This is depicted in the following example, where a clone ($B_4$) and a chunk ($B_6$) are combined into the a new box ($B_9$), creating a duplicate free combination of the inputs’ data.

---

2We felt free to ‘invent’ this verb.
Again, if there is just one box merged, the result is a semantical move of data.

### 2.3 An Armada Database

In practice, databases based on the Armada model evolve over time quickly. For many reasons, e.g., resource limits, boxes are the target of `chunk` and `clone` operations. An illustrative example of a database with 5 boxes is shown below.

```
T_4 = T_3 \cdot [r, S_1]:B_4 ; [m, S_1]:B_9
T_6 = T_2 \cdot [f'', S_2]:B_6 ; [m, S_1]:B_9
T_9 = T_4 \cdot [m, S_1]:B_9 ; [r, S_1]:B_9
```

In this example, we only use fragmentation functions to spread the data in the Armada over 5 boxes. Each box is hosted on a separate site for ease of presentation. The origin overflows when inserting 1.

![Figure 1: Sample Armada with 5 boxes.](image)

(a) the origin overflows when inserting 1

(b) box $B_1$ overflows when inserting 11

(c) the final state of the Armada

In this example, we only use fragmentation functions to spread the data in the Armada over 5 boxes. Each box is hosted on a separate site for ease of presentation. The origin...
box $B_o$ was first chunked into boxes $B_1$ and $B_2$. The first of these two children, $B_1$ is chunked again, resulting in boxes $B_3$ and $B_4$. The evolutionary steps are graphically shown in Figure 1 using symbols which indicate the coverage of the functions applied in the operations on the boxes. The symbol ‘□’ is used to represent the data at the origin of the Armada, in box $B_o$. The other symbols: ‘∩’, ‘⊤’, ‘↓’ and ‘⊔’ represent pieces of the origin box. Note that the symbols equally divide the original square symbol. This is of course only a drawing issue, which is not necessarily true for the fragmentation functions being used.

For this example, we describe how the tree from Figure 1 is built over time by inserting data into the Armada. In the initial situation, depicted in Figure 1a, only $B_o$ exists on site $S_1$. For the sake of the example, the boxes store simple integer values. Each box has a fixed capacity of 5 of such integers. Normally this capacity is determined by the site that hosts the boxes and the size of the data items, but for the sake of clarity we use these fixed sizes. The data to be inserted in the Armada, in order, is for the example:

$$D = \{2, 5, 7, 12, 23, 1, 72, 24, 11, 16\}$$

Since there only fit five integers in each box, the origin $B_o$ consists of $D(B_o) = \{2, 5, 7, 12, 23\}$ when the next integer, 1, is attempted to be inserted. Since it does not fit, a chunk operation is performed. In our example, we split equally, which results in $D(B_1) = \{2, 5, 1\}$ and $D(B_2) = \{7, 12, 23\}$. The fragmentation function $f_1$ used here selects the range $[0 . . . 5]$. The function $f_1'$ selects the complement of $f_1$: $(5 . . . \infty)$. Beware, this decision is taken at site $S_1$ in ‘full autonomy’, it is not inherent to the algorithm.

In Figure 1b, the state of the Armada after the first chunk operation is depicted. As can be seen, the data from the origin box $B_o$ has been moved to boxes $B_1$ and $B_2$. Note that the order of the items in the example is maintained, but this is not a restriction of the Armada model. The only restriction on the boxes is that each box only holds data that matches its respective local trail description.

Continuing the insertion of values, now the right box has to be searched. Inserting the values 72 and 24 ends up in box $B_1$. The origin box $B_o$ is not active any more, and will redirect if being consulted. Since it knows the functions of its successors, it can easily tell that both values fit in the $(5 . . . \infty)$ range of $B_1$ \(^3\). Also the next integer, 11, fits in $B_1$’s range, but since the box is full, a chunk operation has to be performed again. The result of this chunk operation is depicted in Figure 1c. Again the data values have been equally split over the two new boxes $B_3$ and $B_4$. The last integer to insert, 16, ends up in box $B_4$ guided by the ranges associated with the active boxes $B_2, B_3$ and $B_4$.

### 2.4 Localisation

Successful and efficient localisation of the box(es) that (potentially) hold the requested data is a vital prerequisite to allow query execution on an Armada system. Using the previous example, we will briefly sketch that the lineage trails provide sufficient information.

\(^3\)A more detailed description of how this redirection is decided upon is given in Section 2.4.
to find the responsible box(es) for the requested data.

Note that when clients contact the Armada, they are contacting one (or more) of its sites that host boxes, not the boxes themselves. The example from Figure 1c describes 5 boxes that are in fact hosted on 3 sites, \( S_1, S_2 \) and \( S_3 \).

Suppose a client \( c \) has a query which is answered by \( \ldots \), say 42. \( c \) can now contact any of the sites from the Armada. Any site that cannot handle the request by \( c \), will redirect it to the site that it knows has more specific information. The simplest case is when \( c \) connects directly to \( S_3 \). On \( S_3 \), only trail \( T_3 \) is available. This trail defines the box responsible for the data fragment \((12\ldots\infty)\). There are no successors for \( S_3 \) available, meaning \( S_3 \) is active.

Trail \( T_3 \) tells that the query for \( \ldots \) can be answered. In our example this means that \( S_3 \) can tell \( c \) that there is no 42 in the Armada.

In case \( c \) connects to \( S_1 \), \( S_1 \) has three trails at its disposal: \( T_0, T_1 \) and \( T_3 \), where \( T_3 \) is the most “specific” trail. Evaluating from that trail, \( c \)’s query cannot be answered, hence a redirect to the predecessor box has to be made. (There are no successors to consider for \( T_3 \).) Since the predecessor box \( T_1 \) is on the same site, the redirection can be done internally, resulting in no client redirection. Evaluating \( T_1 \), \( c \)’s query can be answered, but since box \( B_1 \) is no longer active, it must be answered by one of its successors. In this case by successor \( T_3 \), which is located on site \( S_3 \). Hence, a redirect to \( c \) for site \( S_3 \) is sent. As obvious from the previous case, at \( S_3 \), \( c \) retrieves the answer to its query.

Finally, \( c \) can decide to connect to \( S_2 \). At \( S_2 \), the trail \( T_2 \) is available. This trail does not cover the query \( \ldots \), so neither would its successors do, if any. Hence, a redirect to the predecessor box is sent. This box, the origin \( B_0 \), is located on \( S_1 \). Since \( S_1 \) does not (have to) know that \( c \) was redirected for box \( B_0 \), it just evaluates \( c \)’s query like it did in the case above, with the same result.

So far we only considered a query which was fully contained in a single box: the lookup of the value 42. Instead of this point query, a range query could be issued by \( c \), that possibly spans multiple boxes. Consider query \( \ldots \) which describes a range \([10\ldots20]\). Like in the previous cases described, client \( c \) will end up at sites \( S_1 \) and \( S_3 \). Both sites will be able to return a partial answer to the query and an additional redirect in order to get the remainder of the answer. Here, the client has to deal with the data being spread over two sites.

It must be noted that for this example we choose to have three different physical sites. This is merely for explanatory purposes. It is very well possible for every box to be on its own site, or for all boxes to be on the same site. There are no inherent restrictions in the Armada model as to where boxes are hosted.

### 3 Architecture Overview

In this section, we illustrate the formation of an Armada, the role of the database schema, and decisions taken by the sites regarding responsibilities for data management.
3.1 The Alliance

A collection of sites $S$ provides the context to distribute the database. At any time, only a subset of $S$ is actually involved in the Armada, i.e., those sites containing boxes and associated lineage trails. The subset of participating sites is called the *Armada alliance*, or $\mathcal{A} \subseteq S$. The starting point for an alliance is a single site $\mathcal{A} = \{S_0\}$, i.e., the origin.

The alliance $\mathcal{A}$ should be extended before a clone, chunk, or combine operation can deposit the result boxes at a new site. A site $Y$ can only be invited to join the alliance if a member of the alliance is willing to co-operate with $Y$. For that, $Y$ should adhere to the Armada’s *Code of Conduct*:

**DEF. 9** A site $Y$ becomes a member of the Armada alliance $\mathcal{A}$ iff

- it is nominated by an existing member,
- it donates resources to manage boxes,
- it keeps a permanent record of its lineage trails,
- it co-operates and faithfully answers queries,
- its existence may be published to other members.

Admission of sites is broadcasted to all members asynchronously or on a need-to-know basis. Due to update propagation delays, at any time a member only knows a portion of the alliance $\mathcal{A}$ by inspection of the lineage trails it receives together with new boxes.

A technical issue is to publish the site identities $S$ of possible new members. In line with the dynamic nature envisioned for Armada, every site $X$ knows only a fraction $S_X \subseteq \mathcal{A}$ and should be told about possibly new members explicitly by intervention from an outside authority.

Selection of candidate sites to join the Armada is initiated by a member when it can no longer fully co-operate due to resource constraints. Given the Code of Conduct, an open call is issued to sites $S'$ for bidding on solving a quantified resource problem, e.g., lack of storage or CPU units.

**DEF. 10** A bid($X$,CPU,MEMORY) is an operation executed by site $X \in S$ and returns $\{t \in \mathbb{R}|0 \leq t < 1\}$, a positive real number representing the value (eagerness) of $X$ to participate with a minimum of CPU and MEMORY units.

**DEF. 11** Let $X \in \mathcal{A}$ be a site with a wish to offload (CPU, MEMORY) units. It issues bid requests to $S' \subseteq S$ and grants the bid from site $Y \in S'$ which supplies the most satisfying bid.

Calling for additional help by a site $X \in \mathcal{A}$ may lead to a situation that no other site $Y \in S$ is willing or able to make resources available, effectively passing back the problem to the site $X$. This failure cannot be resolved at $X$. Instead, the process of finding, negotiating

---

4 New members can be searched using a client application or could be hardwired in the implementation, e.g., using an IP-range.
and bidding is then pushed back to the client by rejecting queries due to resource overload. The client could attempt alternative sites to receive the attention needed, or should contact an outside authority to increase the basis from which Armada members are recruited. In our analogy, any Armada is limited in the ships it can deploy. An extension requires a governmental approval (and new taxation).

### 3.2 Chunk Functions

Extending the Armada (or offloading work to others) is grounded in the ability to fragment and replicate portions of the database. The Armada model captures the offloaded work in the lineage trail as the clone, combine and chunk functions. The operations are precisely administered, such that at any time the lineage trail can be re-interpreted to assess the past decisions.

The chunk function should satisfy the correctness criteria for distributed relational systems [OV99]: the function $f$ is lossless, i.e., each possible data element can be associated with either box $B^f$ or $B^{f'}$, it should designate disjoint portions $B^f \cap B^{f'} = \emptyset$, and the original box can be reconstructed from its components.

Note that a chunk function can be generalised to partition a space into multiple disjoint components. This way it encompasses all known techniques from physical distributed database design. The function $f$ could be a simple hash or range distribution function, which derives the destination boxes based on the key value. Scalable distributed data structures have been developed to support the evolutionary growth as well [L+04]. Alternatively, the function $f$ is a deterministic algorithm, solely based on time and location invariant data properties.

It should be stressed that the nature of the chunk function can be decided upon at each site autonomously and it may differ for each box being considered. The consequence is that chunking or cloning a box leads to a local datamanagement optima, ignoring the goal of an Armada at large to form a coherent and effective distributed system. Autonomy in this respect calls for a brokerage service, e.g., a client application, to mediate between sites to balance their tasks.

### 3.3 Box Updates

A client application $c$ interacts with the Armada sites on a one-by-one basis. It is told the identity of at least one Armada site $X$ using publicly known information or through an authoritative outsider. There is no a priori relationship; a client may pick any member from the alliance.

The client $c$ can issue updates to the Armada using site $X$ as a starting point. If $X$ contains the boxes holding the data of interest, updates follow the traditional local database patterns. However, if the lineage trails carried at $X$ denote existence of cloned versions or
the updates have effect on remote boxes, it tells the client. Unlike traditional distributed
systems, it does not mediate directly in propagation of the update requests.

For example, for cloned boxes it returns a list of sites the client should contact to ensure
global consistency in due time. To implement this policy any of the known database update
replication schemes can be used. They can even be implemented with a few a priori defined
Armada agents, who take over the role of the clients’ responsibility.

Beware that this client-Armada relationship is built on mutual trust and persistency. If
the client forfeits its duty to forward updates to related sites, the outcome could be an
inconsistent database. It does not render the Armada useless, but may affect local decisions
taken in the future. It mimics reality where decisions are mostly based on locally consistent
information only.

3.4 Armada Heterogeneity

A real Armada consists of different types of ships. To name a few famous Dutch kinds, the
galjoten, hoekers and spiegelschepen all have different capabilities on storage, speed or
defence. As such, different ships have different functionalities and responsibilities within
the Armada. Similarly, the sites in our Armada, can be of different types. Not only their re-
sources differ, but also their connection to the rest of the Armada, and the kind of data stor-
age engine they run. Actual storage may be done in a flat (log) file, or in an SQL database
such as PostgreSQL, IBM DB2, Oracle, MySQL, etc., which gives different properties to
the sites.

The analogy goes further when it concerns the data boxes. Much like ships carry both bar-
rels, boxes, and crates, an Armada site carries data boxes of different flavours. Some might
be designed to store particular data items, e.g., blobs, multimedia images or structured
documents, while others are organised around the physical boundary conditions, e.g., disk
block sizes. In all situations, the amount of lineage trail information is considered small
compared to the box itself. Theryby avoiding a bureaucracy.

4 Query Processing

The loose affiliation and strong autonomy of sites, combined with the pivotal role of the
client calls for a fresh look on distributed query processing. In such a vision, any form of
system induced centralised control over query execution is ideally removed.

In this section, we describe the mapping of the Armada reference model to a relational
context and illustrate the challenges for query processing.

5See http://www.holland.com/voc/gb/fleet/ships/index.html for more information on these ships.
4.1 Relational Lineage Trails

The Armada reference model does not a priori prescribe the data model and query language. However, once we deploy it in the context of a real application setting, it has to be fixed to delimit the scope. The first refinement of the Armada model is geared towards relational systems, which calls for a redefinition of boxes and lineage trails.

DEF. 12 A relational box \( B_i \) in a lineage trail \( T_i \) is a box whose content is covered by the relational schema \( DB \) attached to the origin site \( DB = \text{schema}(\text{origin}(T_i)) \).

The data in the box should satisfy traditional key and domain constraints. However, the role of referential and table constraints should be reconsidered in the light of the Armada autonomy. A discussion on this topic is left out of this paper for space reasons.

DEF. 13 A relational lineage trail is an Armada lineage trail \( T_i \) whose chunk, clone, and combine functions \( f, r, m \) are limited to relational algebra queries over the schema \( DB \) attached to the origin site, \( DB = \text{schema}(\text{origin}(T_i)) \).

This definition emphasises the role of the origin site. Its schema determines the scope of the data space managed. All boxes managed by the Armada can be phrased as relational queries, but care should be taken to limit the expressiveness of the query language to also ensure a lossless Armada. The chunk function \( f \) is a simple SELECT-FROM-WHOLE query, such that the key attributes are retained in the derived boxes.

A relational lineage trail can be seen as a small snapshot of a relational catalog. It describes the data retained in boxes in terms of a compound view over the origin schema. Furthermore, the relational trails contain descriptions of database views once managed at remote sites. It forms a roadmap for referral queries and decomposition into a distributed query.

4.2 Single Box Queries

Finding a box with data of interest in the Armada remains the most important query. However, unlike P2P schemes, the whereabouts of the relevant box are not administered centrally; its location may even frequently change.\(^6\)

A client can send a query \( Q \) to any participating site where it can be validated against the database schema for correctness using any of the relational trails. Subsequently, the query is replaced by a union-query \( Q = Q_0 \cup \cdots \cup Q_k \) such that term \( Q_i \) represents a sub-query to be solved by site \( S_i \) in the Armada. Splitting is based on all lineage trails known locally. Algorithmically it requires a search for the union query with all known sites holding boxes.

\(^6\)We consider the hash function in a Chord to be globally known, hence based on a central copy of this function.
of interest. Given the nature of the successor trails known at the boxes, a query might be broken up again when it turns out that the box it refers to was chunked afterwards.

Unlike traditional distributed databases, the subqueries $Q_1, \ldots, Q_n$ are not immediately forwarded to their destination site. Instead a query referral list $Q_l = [(Q_i, S_i)]$ is built and sent back to the client for further consideration. To retrieve the answer the client should explicitly ask the sites for their result sets for the given query $Q_i$. It may iterate through the referral list, asking each site in turn to deliver it, or it may broadcast the complete list at once. This control also gives the opportunity to the client to abort query processing after each sub query issued. In all cases, the client is responsible to merge the results obtained and to deal with the interaction of the sites. Connections to sites may time out. Sites may appear to be unreachable, reject connections or tell they are too busy. In case of clones, this may even result in a redirect to one of the other clones. It is the client’s task to prevent an endless loop to arise if both clones appear to be too busy to handle the request of the client.

As a remedy against unavailable or too busy sites, a client might inspect the lineage trails to see if there are any clones in the lineage. Finding a clone may result in getting the data from another ‘branch’ in the Armada, if available.

The autonomy of the Armada sites and its evolution complicate this scheme. Consider query $Q_i$ arrived at site $S_i$ for evaluation. Then a few cases should be considered.

1. The site $S_i$ accepts the query and can handle it locally. A result set is prepared and shipped to the client.
2. The site $S_i$ accepts the query, but produces a partial answer and an update for the query referral list.
3. The site $S_i$ runs out of resources and is not able to respond to the query request. It returns the query to the client, which should decide on what to do. If the site knows about other sites that could possibly handle the query (partially), it also sends an alternative update for the referral list. Especially for clones this is a standard procedure.
4. The site $S_i$ detects that the boxes of interest have been relocated. It sends a new query referral list back to the client.
5. The site $S_i$ runs out of resources and decides to expand the Armada with new sites. The query is replaced by a new query referral list afterwards and sent back.
6. The site $S_i$ breaks the connection after a partial result has been shipped. The client should re-submit the query.
7. The client breaks the connection with site $S_i$, which triggers a local transaction abort.

The scheme proposed shifts the burden of distributed query processing partly to the client. The rationale is that the client ultimately resolves conflicts, e.g., time to wait for an answer and ‘money’ to spend. An actual client implementation may be based on a library with a priori defined scenarios for dealing with the query referrals.
A tricky part is detection of duplicate results, for it may potentially call a very large local memory at the client side. The solution sought is based on keeping the lineage trails attached to the referral list. It can be used to identify boxes with duplicate information (clones). Judicious execution of the referral queries and early diversion of result sets that are known to hold duplicates are the tools for the client to deal with this problem.

4.3 Query Evaluation

Query optimisation within Armada takes on a different flavour as well. Known techniques for semantic and symbolic query optimisation still apply. However, reducing the amount of data shipped or minimising the response time cannot a priori be the prime target. When a task is taken from the referral list there is still no guarantee on the responsiveness of the site being addressed. Therefore, ruling out many query execution plans upfront, based on cost estimations, is not an option.

The approach taken is based on the rationale that subqueries can only be solved if the (partial) input data resides at a single site. It leads to the refinement of the query referral list into a dependency graph, which captures the processing dependencies. The client should obey these dependencies during query evaluation.

At each site, a subquery is evaluated and/or preparation steps are taken to bring boxes together for the next step. Preparation involves a decision on cloning, chunking and combining pieces at a site with ample resources available. It is a variation on our initial bidding process. For querying we are interested in temporary resources only. After the result has been produced and shipped to the client, the storage could become available for re-use.

The query bid request \( \text{ask}(Q_i, T_i) \) involves the subquery \( Q_i \) and associated lineage trails \( T_i \). It is sent to sites of interest for a quote on its fictive cost.

A site \( S \) can respond in different ways. It may accept the task and reserve resources for the duration of the query. Or, it may propose to initiate an Armada re-organisation first, e.g., cloning, chunking and combining the operands. And finally, it may simply opt out.

The effect at the client is that query evaluation becomes highly dynamic and unpredictable. Just-in-time decisions are taken on where data should be sent and what order of evaluation is most effective. The benefit is that the query plan can be stopped at any time to avoid spending resources on less interesting results.

Furthermore, conceptually the result sets remain at their site of origin until the client explicitly releases the resources. The same holds for all intermediate results. The global effect is that the Armada becomes polluted with temporary results. However, a site’s autonomy will permit unilateral disposal, provided the box discarded can be reconstructed or a referral query can be issued to recover it from a dump site.
5 Related Research

Research on federated/distributed database architectures has a long history. All major DBMS suppliers provide technology to realise a distributed database. They are also optimised for a limited number of servers, e.g., running on a cluster computer composed of several tens of processors with a NAS service. Wide area distributed databases benefit from a plethora of publish/subscribe techniques, e.g., Oracle Streams [Tum04] and Microsoft’s Message Queues [B04b].

Close to Armada’s objectives is Mariposa [S+94]. This system aims for a distributed setting based on fragments of data among autonomous systems. Unlike the envisaged client interaction in Armada, Mariposa passes queries or data on to other sites it knows on behalf of the actual client, resulting in a chain of dependent systems. Further on, location of fragments is not really specified, whereas Armada has this embedded in its lineage trails. The lineage information in Mariposa is used mainly for merging back previously split fragments. Armada on the other hand, allows merging of any two or more boxes.

In recent years, two research trends in distributed databases have emerged: sensor network databases and P2P systems. Sensor network databases are characterised by a large number of resource limited receptors at the edge of a network to collect mission critical data. Prototypical building blocks are small ‘Motes’, a single-board-computer (SBC) equipped with a limited memory, limited network capabilities, and limited energy, glued together to realise a distributed information system to feed the upstream applications. On each site, we find one or more sensors and an embedded SQL database engine for storage management and query processing [M+05, F+05, B+04a, A+05]. However, their underlying architectures ignore the autonomy target set for Armada. In essence, they are built from functionally scaled-down versions of relational database systems.

The focus of Peer-to-Peer systems is efficient query routing and localisation [P+04, MM02]. Armada differentiates from this approach in having a data centric view: the data, in terms of boxes, filled with relations are aimed at evolutionary growth starting from a single node. P2P techniques assume the data is already in place and numerous, usually in the form of files, like in PIER [H+05]. Unlike P2P, Armada has functions that define how data is split over a number of boxes, which allow for concise localisation of data.

Scalable distributed data structures (SDDS), a predecessor of P2P systems, use globally known, but locally adaptive partitioning functions [L+04, KK00]. Also the client behaviour in SDDS implementations bears some similarity with the Armada approach. They manage a cache with metadata to direct data lookups. The main difference with the Armada vision is its level of abstraction. SDDS solutions are focused on single key-based retrieval. In our model, we extend the scope to the complete functionality of a database system. Furthermore, the lineage trails capture the complete history of a box, something not considered in an SDDS. It maintains the latest, locally consistent distribution status.

Over their life span, database systems experience a continuous change (usually growth) of the amount of data stored. Likewise, usage patterns and workloads keep on changing. For example, more recent data is often accessed more frequently than older data, creating a “continuously moving access hotspot”. Classical distributed database architectures do
not provide any means to adapt to these changes automatically. Rather, increasing the systems capacity (by adding additional nodes) and re-distributing the data to balance the load are measures that have to be initiated and executed by some human DBA [OV99]. Additionally, client/server settings form the base of dealing with the work, thereby greatly reducing the autonomy of the participating servers.

The area of self-managing and self-tuning databases limits itself by only advising the DBA [R+02, Z+04] or only dealing with indices and materialised views [A+04] — the metadata. Combinations of replication and fragmentation are not supported, and only on the whole table data, where fragmentation is only horizontally applied. Armada, on the other hand, can be considered a self-adaptive model to meet the environment requirements and reconfigure when they change.

6 Conclusions and Outlook

Emerging applications based on large numbers of autonomous systems challenge the assumptions of underlying traditional distributed database technology. The storage and processing requirements encountered are often modest compared to the servers on which commercial databases run. Instead, they stress the need for autonomy in managing a portion of the database in a co-operative or P2P setting. The volatility of devices joining and leaving the ensemble, calls for a fresh look on metadata management, query processing and transaction semantics.

In this paper, we introduced Armada, a reference model and system architecture for distributed datamanagement. The research methodology is purposely focused on the introduction of a concise model based on lineage trails. The exploratory description of the envisioned architecture charters a rich research landscape ahead. The analogy of a real-world Armada, a fleet of autonomous ships sailing under authoritative goal and charter, provides the necessary insight in alternative solutions herewith no-go areas for distributed database systems.

A simulator for the Armada model has been developed to experiment with large examples and study the effect of lineage trail management. Its next incarnation provides quantitative data on the robustness of the model against (deliberately) unavailability of physical sites.

Other priorities on the Armada research agenda include development of economic models to steer the interaction between clients and Armada sites and fleet formation reorganisation. Finally, a real Armada system implementation based on existing database technology should prove that the database community is ready to take its role in emerging domains.
References

[A+03] E. H. L. Aarts et al., editors. First European Symposium on Ambient Intelligence (EUSAI), volume 2875 of LNCS, 2003.
Abstract: We study an instance-based approach for matching hierarchical ontologies, such as product catalogs. The motivation for utilizing instances is that metadata-based match approaches often suffer from semantic heterogeneity, e.g. ambiguous concept names, different concept granularities or incomparable categorizations. Our instance-based match approach matches categories based on the instances (e.g. products) assigned to them. This way we partly translate the ontology match problem into an instance match problem which is often easier to solve, especially when instances carry globally unique object ids. Since concepts of different ontologies rarely match 1:1 we propose to determine correspondences between sets of concepts. We experimentally evaluate the match approaches for real product catalogs.

1 Introduction

Ontologies become increasingly important in both commercial and scientific application domains. Relevant objects of such domains, e.g. products, genes, etc., can be semantically described and categorized by ontologies. Typically, such ontologies use a controlled vocabulary for the naming of concepts. Concepts can be organized within several generalization/specialization hierarchies (is-a relationships) and be interconnected by additional relationships. Some ontologies, e.g. in life sciences, aim at providing a shared and standardized description of concepts of a community to help exchange and integrate data from different sources [DH05, WVV+01].

Unfortunately, ontologies also introduce semantic heterogeneity since many independently developed ontologies are now in common use. This is especially the case for organization-specific ontologies such as product catalogs, which are typically designed for a specific purpose. Hence ontologies of different organizations may widely differ even if they address the same application domain. As an example, Figure 1 shows portions of two product ontologies of the e-shops Amazon\(^1\) (left side) and Softunity\(^2\) (right side). Users can browse through the concepts (categories) of such product catalogs to find the associated products, e.g. software products such as "Windows XP Home" and "SuSE Linux 10.1". Product information is typically structured according to a database schema using product- and shop-specific attributes, such as id, title and price. As the example shows, both ontologies are differently organized. Unlike Softunity, the Amazon ontology consists of multiple orthogonal hierarchies, e.g. "by brands" and "by category". Therefore, products such as "Windows XP Home" can be related with multiple concepts.

---

\(^1\) [http://www.amazon.com](http://www.amazon.com)

\(^2\) [http://www.softunity.com](http://www.softunity.com)
Moreover, the Amazon ontology differentiates between "Windows" and "Linux" operating systems while the Softunity ontology only has a single concept "Operating System". Hence, both ontologies are of different granularity.

An ontology mapping can bridge the semantic heterogeneity of different ontologies and thus help to search or query data from different sources, e.g. to compare or recommend similar products offered in different e-shops. Previous approaches to determine a mapping or match result between ontologies mostly utilize metadata like the concept names, concept descriptions or structural context information. However, the usefulness of such approaches is often limited due to the semantic heterogeneity problems discussed, e.g. ambiguous concept names, different concept granularities or incomparable categorizations.

We therefore advocate for a simple instance-based match approach which matches concepts (product categories) based on the instances (e.g. products) assigned to them. This is motivated by the assumption that the real semantics of a concept is often better defined by the actual instances assigned to the concept but by metadata like the concept name. To determine matching concepts using instances we need to find matching instances between the ontologies, i.e. we partly turn the ontology match problem into an instance (object) match problem. Instance matching is based on specific data values and thus often easier to solve than matching abstract metadata. An ideal case for instance matching is given when instances carry globally unique object ids. For example, many e-shops use unique product ids, so-called EANs (European Article Number). In the example in Fig.1, the EAN values allow us to find the two shown instance (product) correspondences for the Linux and XP products. These instance correspondences in turn can be used to determine matches between the associated product categories, e.g. we can find out that the Amazon categories "Microsoft" and "Windows" both match the Softunity category "Operating System". Obviously such an instance-based match approach is the more promising the higher the instance overlap of the ontologies.

Figure 1: Portions of two application-specific ontologies with associated objects
Previous match approaches often restrict themselves to mappings of 1:1 and N:1 cardinality. For schema matching such mappings are needed for data exchange between a source and a target schema where each target attribute value must be uniquely derived from one or several source attribute values. Ontology mappings of cardinalities 1:1 and N:1 are sufficient to express equivalence and subset relationships between concepts of different ontologies. However, we find that concepts like product categories of different ontologies may overlap in almost arbitrary ways so that there is a need to support N:M match relationships. We thus propose to use instance matches for determining correspondences between sets of concepts and support 1:1, N:1 and N:M mapping cardinalities. The coarser N:M ontology mappings are still useful for important applications, e.g. ranked keyword queries or product recommendations from related categories at a different e-shop.

The rest of this paper is organized as follows. In the next section we briefly discuss some additional related work. Section 3 describes how to determine instance-based ontology mappings and presents an experimental comparison of its effectiveness with a name-based match scheme. In Section 4 we illustrate and evaluate set correspondences. Section 5 concludes.

2 Related Work

There is a big literature on algorithms for schema matching and ontology matching [RB01, KS03, AGY05, DH05, SE05]. The approaches can be roughly classified as metadata-based, instance-based or mixed forms. Metadata-based match algorithms, e.g. [MS02, ELT+03, NM03, MB04, ADMR05], utilize concept names, concept descriptions or definitions (if available) and the ontology graph structure. However, concept names in e-Business are often short and ambiguous. For instance, concept names, such as "miscellaneous", "collections" and "accessories", are often used in different contexts within the ontology. To make concept names more meaningful, they can be concatenated along the path from the ontology root to the concept node. However, using such path names is not always effective since concepts can be differently arranged in different ontologies by incomparable classification criteria.

Some instance-based schema matching approaches utilize previously identified duplicate instances between overlapping sources, e.g. [PE95, CCL03, BN05]. While we use instance matches to derive category matches these approaches focus on the use of duplicates for matching the attributes of the instances. Moreover, these approaches consider 1:1 and 1:N/N:1 match cardinalities whereas our ontology matching approach also detects N:M match relationships.

Instance-based ontology matching is investigated in [AS01, ITH03, DMD+03, HYN+04] using different statistical or machine learning approaches. [AS01, DMD+03] utilize a Naïve Bayes classification approach to assign source concepts to the concepts of a master catalog; the instance mapping is used to improve the classification accuracy. [ITH03] matches categories between two internet directories based on their containing web links (instances) but apply a metric that is different from ours. [HYN+04] compares feature vectors for each concept pair using keywords found in the instances and then determines similar feature vectors by a structural matcher. The ontology mappings gen-
erated by all these instance-based approaches only consist of single concept correspondences but not set correspondences.

The evaluation of match algorithms typically requires generated mappings to be compared with a perfect, manually determined match result by using information retrieval metrics such as precision and recall. However, creating such a perfect mapping for large real-world ontologies is extremely labor-intensive. Furthermore, it is often difficult to clearly decide when two concepts should match due to the mentioned problems of semantic heterogeneity. Therefore, we do not try to derive a perfect mapping for our evaluation but compare the result sets of different algorithms with each other, similar to \[\text{BAB05, MTM+06}\].

3 Instance-based Matching of Ontologies

For our study, an ontology consists of a is-a hierarchy of concepts. Concepts can have multiple associated instances, i.e., objects that are described or classified by the concept. An instance can be associated with multiple concepts, e.g. when the ontology contains concepts of orthogonal aspects. Moreover, an instance may be assigned not only to leaf-level concepts but also to inner concepts of the ontology.

The key idea of our approach is to derive the similarity between concepts from the similarity of the associated instances. Determining such instance matches is easy in some domains, e.g. by using the non-ambiguous EAN in e-commerce scenarios. Moreover, instance matches may be provided by hyperlinks between different data sources and, thus, can easily be extracted. In the absence of unique identifiers, instance matching can be performed by general object matching (duplicate identification) approaches, e.g. by comparing attribute values.

An important advantage for instance-based ontology matching is that the number of instances is typically higher than the number of concepts. This way, we can determine the degree of concept similarity based on the number of matching instances. Furthermore, the match accuracy of the approach can become rather robust against some instance mismatches.

In the following we first introduce three metrics to determine an instance-based similarity between concepts. Afterwards we present the metrics used for evaluating the ontology match approaches. Section 3.3 evaluates the approaches for matching two real-world product catalogs.

3.1 Similarity metrics

In this paper we study three metrics for determining the instance-based similarity between concepts $c_1$ and $c_2$ of different ontologies, namely the dice similarity $\text{Sim}_{DICE}(c_1, c_2)$, the minimum similarity metric $\text{Sim}_{MIN}(c_1, c_2)$ and the base similarity metric $\text{Sim}_{Base}(c_1, c_2)$.

The dice similarity metric \([\text{Rijs79}]\) between two concepts $c_1$ and $c_2$ of the concept sets $C_{O1}$ and $C_{O2}$ of two ontologies $O_1$ and $O_2$ is defined as follows:

$$\text{Sim}_{DICE}(c_1, c_2) = \frac{2 \cdot |I_{c_1} \cap I_{c_2}|}{|I_{c_1}| + |I_{c_2}|} \in [0..1], \forall c_1 \in C_{O1}, c_2 \in C_{O2}$$
In the formula, \(|I_{c_1}|\) (\(|I_{c_2}|\)) denotes the number of instances that are associated to the concepts \(c_1\) (\(c_2\)). \(|I_{c_1} \cap I_{c_2}|\) is the number of matched instances that are associated to both concepts, \(c_1\) and \(c_2\). In other words: the similarity between concepts is the relative overlap of the associated instances.

The dice similarity values do not take into account the relative concept cardinalities of the two ontologies but determine the overlap with respect to the combined cardinalities. In the case of larger cardinality differences the resulting similarity values thus can become quite small, even if all instances of the smaller concept match to another concept. We therefore additionally utilize the minimal similarity metric which determines the instance overlap with respect to the smaller-sized concept:

\[
\text{Sim}_{\text{min}}(c_1, c_2) = \frac{\min(|I_{c_1}|, |I_{c_2}|)}{|I_{c_1} \cap I_{c_2}|}
\]

For comparison purposes we also consider a base similarity which matches two concepts already if they share at least one instance.

\[
\text{Sim}_{\text{base}}(c_1, c_2) = \begin{cases} 1 & \text{if } |I_{c_1} \cap I_{c_2}| > 0 \\ 0 & \text{if } |I_{c_1} \cap I_{c_2}| = 0 \end{cases} \quad \forall c_1, c_2 \in C_{O_2}
\]

Obviously it holds for all correspondences between concepts \(c_1\) and \(c_2\):

\[
\text{Sim}_{\text{base}}(c_1, c_2) \leq \text{Sim}_{\text{dice}}(c_1, c_2) \leq \text{Sim}_{\text{min}}(c_1, c_2)
\]

We may also apply other similarity metrics, e.g. an asymmetrical metric such as \(\text{Sim}(c_1, c_2) = \frac{|I_{c_1} \cap I_{c_2}|}{|I_{c_1}|}\). We leave the analysis of other metrics as a subject for future work.

### 3.2 Evaluation metrics

The standard metrics for evaluating the effectiveness of match approaches, recall and precision, require that the perfect match result is known. However, this perfect match result is generally unknown for difficult real-life match problems, especially for large heterogeneous ontologies. Fortunately, for our instance-based match approaches we can use the base similarity metric as a yardstick for evaluating alternate match approaches. This is because a baseline matcher using this similarity metric achieves the maximal possible recall for instance-based ontology matching. On the other hand, its precision is likely to be very low because it matches two concepts already if they share only one instance, i.e., even for low concept similarity. Other instance-based approaches (like using the dice or minimum similarity metrics) yield subsets in both the set of matching categories and the correspondences, i.e. lower recall, than the baseline matcher. However, these alternatives are likely to be more precise than the baseline matcher since they restrict themselves to category correspondences with a larger instance overlap.

For measuring the recall of a match approach we thus propose to use a relative Match-Coverage metric w.r.t. to the baseline matcher. Let \(C_{O_1 \cup O_2}\) be the number of determined correspondences between ontologies \(O_1\) and \(O_2\) for a given match approach. \(C_{O_1}\) \((C_{O_2})\) denotes the set of matched \(O_1\) \((O_2)\) concepts, i.e., the set of concepts having at least one correspondence. We then define match coverage as follows:

\[
\text{MatchCoverage} = \frac{|C_{O_1}| + |C_{O_2}|}{|C_{O_1 \cup O_2}|}
\]
In the formula, $C_{\text{Base-O1}}$ ($C_{\text{Base-O2}}$) is the set of matched O1 (O2) concepts using the baseline approach.

For estimating the precision of a match approach we determine the so-called \textit{MatchRatio} metric, i.e., the ratio between the number of found correspondences and the number of matched concepts:

\[
\text{MatchRatio}_{O1} = \frac{|\text{CorrMatch}_{O1}|}{|C_{O1}|} \quad \text{MatchRatio}_{O2} = \frac{|\text{CorrMatch}_{O2}|}{|C_{O2}|}
\]

The intuition is that the value (precision) of a match result is better if a concept is not loosely matched to many other concepts but only to fewer (preferably the most similar) ones. The match ratio for the baseline matcher is expected to provide a worst-case value for instance-based matching.

### 3.3 E-Commerce scenario

Our experimental evaluation uses the real-world product catalogs and instance data of Amazon.de and Softunity.com. The catalogs are restricted to the area of software and games. Table 1 summarizes their characteristics. The comparison of Amazon and Softunity shows a significant difference in both the number of instances and the number of concepts. Note that, unlike Softunity, Amazon products are on average directly associated to 1.4 concepts. Only 36% of all Softunity concepts have directly associated products but almost 93% of Amazon concepts do so. Obviously, Amazon frequently associates products to inner concepts that are less related with their descendants in the hierarchy. Note that concepts also have \textit{indirectly associated products}, i.e. the products which are directly assigned to at least one of their descendants.

The underlying (perfect) instance match is determined by matching products having the same EAN. It contains 1872 matches and cover about 73% of the Softunity products. Using the perfect instance mapping we determine correspondences based on the introduced similarity metrics. Table 2 shows the results for the baseline matcher; Table 3 and Fig. 2 show results for the Dice and Minimum similarity metrics for different similarity thresholds. In all cases, we distinguish between direct associations (concept similarity based on overlap of directly associated instances), and indirect associations that also consider instance associations from sub-concepts of the is-a hierarchy. For indirect associations we eliminate trivial concept correspondences, i.e., given a correspondence between two concepts we remove all correspondences between their ancestors that do not have a greater similarity. For a given threshold, the usage of indirect associations will increase the number of correspondences because additional match candidates are considered. This extension is also beneficial to handle different concept granularities. For the
starting example in Figure 1, indirect associations can help match the Operating Systems concepts, although the Amazon concept has no directly associated products.

Table 2 indicates that the baseline matcher finds correspondences only for a minority of the concepts, namely 28% (34%) of the Softunity and 18% (20%) of the Amazon concepts using direct (indirect) associations. The match ratios are rather high; using indirect associations almost triples the match ratios, i.e. the number of matching concepts per matched concept.

Table 3 confirms that dice similarity is very restrictive making it difficult to obtain high concept similarities. Hence only few correspondences are achieved for direct associations and only few concepts can be matched (low recall). As shown in Fig. 2, for all similarity thresholds the match coverage is less than 30% compared to the baseline matcher. On the other hand, the quality of the correspondences is quite good. For example, with a 50% similarity threshold we obtain 71 correspondences covering 60 (68) different Softunity (Amazon) concepts leading to a very good match ratio of 1.2 (1.0). The baseline approach, on the other hand, uses the ten-fold number of correspondences for matching about twice the number of Softunity concepts (ratio 5.4) and five times the number of Amazon concepts (ratio 2.1). Indirect associations help to slightly improve the match coverage for dice without impairing the match ratios. In section 4 we analyze how the match coverage can be further extended by considering set correspondences.

The minimum similarity metric is less restrictive than dice similarity and determines many more correspondences. Furthermore, many more concepts can be matched (Figure 2) so that match coverage is improved significantly for our test data. Even for a similarity threshold of 1 (100%) a match coverage of up to 80% is achieved. This good coverage is obtained with many fewer correspondences than in the baseline case (ratios of about 2.7 for Softunity and 1.1 for Amazon). Compared to dice similarity the much improved recall is achieved with a similar good precision for Amazon concepts. The higher ratio for Softunity is influenced by the much higher number of Amazon concepts so that more correspondences are needed per Softunity concept to match most instances. In

Table 3: Number of concept correspondences for instance-based matching

<table>
<thead>
<tr>
<th>Association</th>
<th>Metric</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
<th>100%</th>
<th>Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>Dice</td>
<td>71</td>
<td>40</td>
<td>21</td>
<td>17</td>
<td>13</td>
<td>11</td>
<td>711</td>
</tr>
<tr>
<td></td>
<td>Min</td>
<td>389</td>
<td>308</td>
<td>255</td>
<td>233</td>
<td>213</td>
<td>208</td>
<td></td>
</tr>
<tr>
<td>Indirect</td>
<td>Dice</td>
<td>90</td>
<td>62</td>
<td>34</td>
<td>30</td>
<td>23</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Min</td>
<td>500</td>
<td>425</td>
<td>385</td>
<td>364</td>
<td>346</td>
<td>335</td>
<td>2,251</td>
</tr>
</tbody>
</table>
summary, using the minimum similarity is the best match approach for the considered e-commerce scenario and more appropriate than dice.

### 3.4 Comparison between metadata- and instance-based matching

To compare the instance-based approaches with metadata-based ontology matching, we applied different name matchers on the product catalogs. Several name-based mappings are determined by using the trigram string similarity between the concept names of Amazon and Softunity. The mapping NAME-SU determines for each Softunity (SU) concept the Amazon concept with the most similar name; a correspondence is only assumed if the similarity values exceed a minimal similarity of 80%. The mapping NAME-AM analogously determines the correspondences for Amazon (AM) concepts. The symmetrical mapping NAME-SUAM only selects correspondences fulfilling a “stable marriage”, i.e., the best matching Amazon concept for a given Softunity concept has the same Softunity concept as the best match, too. Three additional name mappings are determined which concatenate the concept names with the names of all parent concepts (Path matcher). This way names become less ambiguous and reflect the structural position of a concept within the ontology. Due to the high diversity of path names, we use the best correspondences for each Softunity (Path-SU) and each Amazon (Path-AM) concept respectively without checking for a minimal similarity value. Similar to the name matcher Path-SUAM only selects correspondences fulfilling a “stable marriage”.

Table 4 summarizes our results. The first observation is that the simple name matchers match relatively few concepts (31% for Softunity; 9% for Amazon) but determine correspondences with a rather high match ratio (4.0 – 4.7). The reason is that many concepts have equal or similar names (e.g., "miscellaneous") but are not related to each other. This ambiguity is reduced when using the path name instead of concept name only. The symmetrical path matcher Path-SUAM seems most successful as it achieves a perfect match ratio of 1 for both ontologies. Moreover, Path-SUAM achieves a comparable number of matched concepts than the name matchers but with only a fraction of correspondences.
Comparing the number of matched concepts of the baseline approach (Table 2) with the metadata approaches (Table 4) we see a similar match coverage for Softunity. On the other hand, the metadata-based approaches match only half of the Amazon concepts (with the exception of Path-AM). However, a similar number of matched concepts does not mean that the same concepts are matched by the different approaches. We therefore determine the overlap of the metadata-based and instance-based matching using the baseline scheme as well as the dice and minimal similarity metrics (similarity threshold of 50%). Table 5 shows the number of shared correspondences for the different approaches. For example, the Path-SU matcher determines 492 correspondences whereas the instance based matcher using the dice similarity metric and direct associations determines 71 correspondences. But only 20 correspondences can be found in both match results.

Table 5 reveals a very small correspondence overlap between the metadata-based and instance-based matchers for both direct and indirect associations. The path matchers return a much higher overlap than the name matchers underlining their superiority. The highest relative overlap is achieved for Path-SUAM for which almost 30% of the correspondences are also obtained by the baseline instance matcher. For the instance-based matchers the dice similarity metric obtains the smallest overlap, while the minimum similarity achieves about 80% as many overlapping correspondences as the baseline matcher. Interestingly, for the minimum similarity there is hardly any difference in the overlap between direct and indirect associations although the latter generates significantly more correspondences. The results show that the metadata-based matching approaches miss many concept correspondences with a significant instance overlap. On the other hand, name-based matching identifies many correspondences without instance overlap. Note that these correspondences are not necessarily wrong but can be useful to
find related products even in the absence of matching instances, e.g. when stores have similar but different products (e.g. equivalent products from a different manufacturer). Altogether the experiment clearly shows the need for both approaches, instance- and metadata-based matching.

4 Set Correspondences

The correspondences considered so far related single concepts. Set correspondences relate sets of concepts between two ontologies. We motivate the use of set correspondences, explain their calculation and evaluate them for our test data. Throughout this section we focus on the restrictive dice similarity and direct associations which were shown to determine high quality correspondences but need recall improvements to match more concepts.

4.1 Motivating example

Figure 3 illustrates that set correspondences may express semantic relationships better than single correspondences. For example, we assume that none of the two highlighted Softunity concepts (Adventure Games, Educational Software) corresponds to only one of the highlighted Amazon concepts (Children & Family, Edu- & Infotainment). Hence to accurately describe such a N:M relationship between concepts we should be able to use one correspondence between concept sets rather than only correspondences between single concepts.

We therefore generalize the dice similarity for set correspondences. Given two concept sets $C_1$ and $C_2$ as subsets of all concepts $C_{O1}$ and $C_{O2}$ of two ontologies we define

$$Sim_{dice}(C_1, C_2) = \frac{2 |I_{C_1} \cap I_{C_2}|}{|I_{C_1}| + |I_{C_2}|} \in [0...1], \forall C_1 \subseteq C_{O1}, C_2 \subseteq C_{O2}$$

Analogously, $I_{C1}$ and $I_{C2}$ are the union sets of associated instances to concept sets $C_1$ and $C_2$, respectively, whereas $I_{C1} \cap I_{C2}$ denotes the matching instances to both concept sets.

Figure 4 illustrates the use of the generalized dice similarity metric for a more abstract example with two matching concept pairs {A, B} and {A’, B’}. The circles denote instances that are associated with concepts. For example, the left-most instance (circle) is assumed to be associated to both concepts A and A’. The computation of the instance-based dice similarity for single correspondences leads to the result given in the table.

![Diagram](image_url)
(assuming cardinalities 3, 2, 3, and 3 for concepts A, B, A’ and B’, respectively). On the other hand, the generalized dice similarity for the set correspondence \{A, B\}-{A’, B’} is \(2*5/(5+6) \approx 0.9\) and therefore higher than for all considered single correspondences. The example demonstrates that set correspondences may have much higher similarity values (instance overlaps) than single concept correspondences and are therefore useful for representing relationships between concepts.

### 4.2 Determining Set Correspondences

Set correspondences are established during an iterative process based on the single correspondences that are a special case of set correspondences. Concepts are successively added to the sets on both sides of the correspondence. It is important to note that the extension of a concept set by one concept must improve the correspondence similarity to avoid trivial set correspondences. Therefore no concepts are added that do not strengthen the correspondence. Hence, we require that for all concept sets A and B it holds:

\[ A' \subseteq A \land B' \subseteq B \land (A' \neq A \lor B' \neq B) \rightarrow \text{Similarity (A-B)} > \text{Similarity (A'-B')} \]

### 4.3 Experimental evaluation

In the following experiment we start from the single correspondences using direct associations and the dice similarity metric. We generate concept sets step-by-step up to a maximum of three concepts per set and count the number of resulting correspondences with at least 50% similarity. Table 6 shows the number of correspondences w.r.t. the size of the concept sets, e.g., we count 30 correspondences between sets of two Softunity concepts and one Amazon concept.

The comparison of Softunity and Amazon shows a different development for the number of correspondences when extending the concept sets. The number of new correspondences increases when considering more Amazon concepts but decreases for Softunity. One reason is that Amazon has many more concepts so that the associated products of one Softunity concept are distributed over multiple Amazon concepts.

The example of Section 4.1 illustrates that set correspondences may involve concepts
Conclusions & Future Work

We showed that instance matching can effectively be used for matching hierarchical ontologies such as product catalogs. Instance-based matching considers the extensional overlap of concepts and is thus able to find concept correspondences even in the presence of high degrees of semantic heterogeneity, e.g., different concept names or incomparable categorizations. Our experimental evaluation demonstrated the value of the instance-based approach over metadata-based matching which missed many correspondences between concepts sharing the same instances.

To summarize our results for instance-based matching Figure 6 illustrates the number of found concept correspondences (e.g., 335 for the minimum similarity approach using a 100% threshold) as well as the number of used instance correspondences (849) for different match strategies. Fig. 6 also presents the match coverage (90%) as well as the match ratios for Softunity and Amazon concepts (2.7 and 1.1). The comparison indicates the high usefulness of the minimum similarity and the recall improvements using concept sets. We showed that the minimum similarity nearly achieves the same coverage (Fig. 6b) like the baseline approach (Fig. 6a). Moreover, this coverage is achieved by less than 50% concept correspondences (335 of 711) resulting in much improved match ratios.

Comparing Figure 6c and 6d illustrates that set correspondences are able to match significantly more concepts by using a higher number of instance correspondences (535 vs. 842). This underlines our assumption that 1:1 correspondences are often not sufficient for matching ontologies. However, the number of concept correspondences increases as well resulting in rather poor match ratios, e.g., compared to the minimum approach (Fig. 6b). This suggests that many set correspondences do not actually improve match coverage because they only combine already matched concepts.

Figure 6: Comparison of different matching strategies
In future work we will therefore further investigate set correspondences to eliminate such useless set correspondences and improve precision. Furthermore, we plan to apply instance-based matching in different domains, such as life sciences. We also want to further analyze possible combinations of instance- and metadata-based ontology matching.

References


448
A Classification of Schema Mappings and Analysis of Mapping Tools

Frank Legler
IBM Deutschland Entwicklung GmbH
flegler@de.ibm.com

Felix Naumann
Hasso-Plattner-Institut, Potsdam
naumann@hpi.uni-potsdam.de

Abstract: Schema mapping techniques for data exchange have become popular and useful tools both in research and industry. A schema mapping relates a source schema with a target schema via correspondences, which are specified by a domain expert possibly supported by automated schema matching algorithms. The set of correspondences, i.e., the mapping, is interpreted as a data transformation usually expressed as a query. These queries transform data from the source schema to conform to the target schema. They can be used to materialize data at the target or used as views in a virtually integrated system.

We present a classification of mapping situations that can occur when mapping between two relational or nested (XML) schemata. Our classification takes into consideration 1:1 and n:m correspondences, attribute-level and higher-level mappings, and special constructs, such as choice constraints, cardinality constraints, and data types. Based on this classification, we have developed a general suite of schemata, data, and correspondences to test the ability of tools to cope with the different mapping situations. We evaluated several commercial and research tools that support the definition of schema mappings and interpret this mapping as a data transformation. We found that no tool performs well in all mapping situations and that many tools produce incorrect data transformations. The test suite can serve as a benchmark for future improvements and developments of schema mapping tools.

1 Schema Mappings, Data Exchange, and Mapping Tools

The problem of information integration, i.e., enabling access to multiple distributed, autonomous, and heterogeneous data sources through a common interface (common schema, common query language), is eminent in database research and information systems development. Apart from technical challenges the problem of schematic heterogeneity is particularly obtrusive. Even when using a common data model there are many different ways of modeling the same real world entities and relationships. In this article we focus on the technique of schema mapping to describe the relationships between two schemata in the context of data transformation. Schema mappings and their use for data exchange has recently become a popular notion as exemplified in [AL05, FKP05, MBHR05] and many other projects and tools. Schema mappings have a wide array of further usages, such as schema translation [SL90] and schema integration [BLN86], not considered in this article.

Regard the simple example of Fig. 1. It shows two nested schemata, both modeling persons and their membership in teams. In the first schema, the membership of a person to a
team is modeled as a foreign key constraint, in the second, the membership is modeled by nesting person elements under team elements. Also observe that different facts are represented in the different schemata. The first schema models the address of a person while the second schema does not. And vice versa, the second schema models the person’s date of birth (DOB) while the first does not. Even when elements with the same meaning are included, their labels need not coincide. For instance, teamURL and website have the same semantics but different labels.

Figure 1: A mapping between two nested schemata

Given these and many more heterogeneities among schemata, schema mapping describes the general technique of relating elements between two heterogeneous schemata. The relationships are based on some semantic similarity of the elements and are usually called correspondences; graphically, correspondences are modeled as arrows from an element in a source schema to an element in a target schema; conceptually, a correspondence states that data for the target element can be obtained by fetching the data stored at the corresponding source element. Formally, a correspondence is a relation between a set of elements in the source schema and a set of elements in the target schema. This relation is identified by a transformation function that transforms source data into target data or a filter that selects elements of the source schema [Leg05]. This definition also allows m:n correspondences. A schema mapping is a set of correspondences between a source schema and a target schema. Schema mappings are used to transform data stored under the source schema so that it conforms to the target schema. In a typical situation, the source schema might be that of a data source and the target schema is the federated schema of an integrated system. In a data exchange scenario the two schemata are those of peers willing to exchange data.

Figure 1 shows a schema mapping that relates attributes of the left-hand schema, the source schema, with the right-hand schema, the target schema. This situation raises two important questions that have been extensively dealt with in recent literature and in recent products: (i) How can one obtain the correspondences between two schemata and (ii) how can one interpret a set of correspondences to actually transform source data so that it conforms to the target schema. In this article we focus on the second question and assume that the correspondences have been established.
Informally, the interpretation of a schema mapping faces several difficulties:

**Source schema interpretation:** Associations of data elements (relations, nesting, foreign keys) in the source schema should be recognized and preserved during transformation.

**Target schema conformance:** The result of a transformation should conform to the target schema.

**User intention:** The expert users merely provide informal correspondences ("arrows") between the schemata. Even under the demand that an interpretation of such a mapping should cover all correspondences there usually remain several alternatives. Correctly guessing the intended one can rely only on suitable heuristics.

In this article we classify these difficulties and evaluate several schema mapping tools in their ability to overcome them. We observe that surprisingly not even the first two difficulties are adequately addressed in many of the tools. We limit the scope of our evaluation to the relational and XML data models, because of their widespread usage and the fact that they are supported by most schema mapping tools. Hence, we also limited the number of query languages that are used to transform data from the source to the target. For simplicity in this article we restrict ourselves to a graphical notation for schemata and mappings. Even though we only consider the relational and XML data model in this article, the basis for our research is a schema definition language that is independent of the data model. Both the relational data model and the XML data model can be transferred into this schema definition language. The schema definition language, the transformation of the relational and the XML model into the schema definition language, and the formal definition of each mapping situation can be found in [Leg05].

2 A Classification of Mapping Situations

Figure 2 shows our classification of schema mapping situations, distinguishing three main classes: mapping situations related to *missing correspondences* displayed in the left subtree, mapping situations related to *single correspondences* in the middle, and mapping situations belonging to *multiple correspondences* on the right. The following sections introduce each of the classes and outline possible ways for mapping tools to interpret them.

A classification similar to ours was presented by Kim et al., who classify conflicts between relational schemata based on their structure [KS91]. We used this classification as a basis for ours and extended it to include non-relational features. Previous research in schematic heterogeneity yielded several classifications of correspondences and mapping conflicts. Batini et al., for example, distinguish four types of semantic relationships: identical, equivalent, compatible, and incompatible [BLN86]. Every non-identical relationship implies a conflict. Their classification requires the existence of semantic knowledge about the participating schemata. This requirement means that users know exactly which real world object is modeled by which schema element. Our classification, on the other hand, just utilizes the structural information given by the schemata and the mapping between them.
Spaccapietra et al. classify conflicts between schemata into semantic conflicts, descriptive conflicts, heterogeneity conflicts, and structural conflicts [SPD92]. Because our classification assumes a given schema mapping, we assume both semantic conflicts and descriptive conflicts as resolved. The two remaining classes contain the conflicts that are interesting for our purpose, but were not sufficiently analyzed.

2.1 Missing correspondences

This class contains mapping situations that occur if leaf nodes or inner nodes of the schema are not part of the schema mapping, i.e., no correspondence is connected to the nodes.

If leaf nodes of the source schema are not part of the mapping, the only important consideration is a loss of information, meaning that the source data cannot be recreated from the target data after the transformation. On the contrary, if leaf nodes of the target schema are left out of the mapping, constraints of the target schema could be violated:

Figure 2: Classification of schema mapping situations
• If the node is part of a **key or unique constraint** (not null) a mapping tool could automatically generate a value, as Popa et al. demonstrate \[PVM^*02\]. Another possible solution is to reject the mapping and ask the user for manual resolution.

• If the node is part of a **foreign-key constraint**, a tool could automatically detect a connection between the key and the foreign-key using the information from the source schema. If that is not possible, a manual resolution is necessary.

• If the node is **mandatory** and a default value is given, a tool should use this information. Otherwise, a manual resolution would be the best option. A random value is also an acceptable but not optimal solution. The fact that the value is required implies that it is relevant for the application; inserting a random value contradicts this intuition.

• If the node is **not mandatory**, a tool can neglect it, preferably with some warning.

Whether a missing correspondence to and from inner nodes of a schema tree influences the produced transformation query or not depends on the definitions of correspondences and mappings. Problems occur depending on whether a tool allows and interprets correspondences between inner nodes or not (see later).

### 2.2 Single correspondences

Mapping situations related to single correspondences are outlined in this section. The classification is based on the properties of correspondences along different dimensions:

1. **Cardinality of the correspondence.** According to the number of the participating schema elements in a correspondence, 1:1, 1:n, n:1, and n:m correspondences can be identified. Only 1:1 correspondences are further analyzed in this section. The remaining three types are analogous to problems at inner node level: the question on how to combine multiple input values and how to produce multiple output values are similar to the questions discussed for multiple correspondences (see Sec. 2.3).

2. **Type of the participating schema elements.** Elements associated with a correspondence can either be leaf nodes or inner nodes.

   • Correspondences between **leaf nodes** are the most common kind of correspondences and are supported by every schema mapping tool. Their purpose is to define which source data to transform into which target data.

   • Correspondences between **inner nodes** are not supported by all mapping tools. If they are supported, then for every element in the source instance an element in the target instance could be created.

   • When connecting **inner nodes with leaf nodes**, the data and metadata levels are mixed. If no additional transformation function is given, a mapping tool should reject this correspondence. Figure 3 shows an example of metadata in the source schema corresponding to data in the target schema.

3. **Properties of the participating schema elements and constraints.** Correspondences can also be classified according to the properties that participating schema elements have (see bottom left of Fig. 2).
There are several **cardinality-related situations**, which can be classified as situations with a **loss of information** (when source data cannot be transferred into the target schema due to a higher cardinality in the source schema) and situations with a **lack of information** (when the source instance does not include enough information to build a valid target schema instance due to a lower cardinality in the source schema). Both situations should be recognized by a mapping tool and treated accordingly. For example, if in a 1:1 correspondence the source node is nullable and the target node is mandatory with no default value, a mapping tool should ask for a manual resolution for cases where the source value is null.

Conversion tables or cast tables define how **data types** can be transformed into each other (also spanning different data models). They can be used to resolve mapping situations related to different data types. According to these tables there are three classes of compatibility: **compatible**, **partly compatible** (depending on the concrete value), and **incompatible data types**. For compatible data types, mapping tools could insert casts according to these conversion tables, and warn users if data types are not or only partly compatible.

### 2.3 Multiple correspondences

This class comprises mapping situations related to multiple correspondences. The categories in this class are based on the structure of the source and target schema rather than on the properties of individual elements.

Research projects addressing this topic developed sophisticated algorithms to discover clusters of semantically connected schema elements (e.g., [MHH00, PVM+02]). To be able to better comprehend and compare the tests and results, a simplified approach was chosen. According to this approach, three different kinds of relationships (associations) between elements of a schema can be distinguished:

- Elements have a **structural connection** if they have at least one common ancestor, whose cardinality is greater than one. A mapping tool should recognize *structurally connected elements in a source schema* and maintain their semantic relationship. This can be done by transforming structurally connected elements simultaneously, e.g., leaf nodes with the same parent node. Furthermore, a mapping tool should be able to combine nested elements by un-nesting them where appropriate.

  Regarding *structurally connected elements in the target schema*, a tool has the
choice whether to group the data or not. This question also leads to the problem whether an aggregation of the data is possible. Figure 4 shows an example where a grouping of families under the same lastname and an aggregation of the family-income is desirable.

![Figure 4: Grouping and aggregation](image)

- A **foreign-key based connection** between elements is given if the elements reside in subtrees, that are interconnected with a foreign-key relationship. This class also includes explicitly defined joins between subtrees, which can be captured with some mapping tools. A mapping tool should recognize *source schema elements, that are connected by a foreign-key* and maintain their semantic relationship. Hence, a transformation query should contain a (outer-)join over the connected subtrees. For *target schema elements with a foreign-key connection* two situations can be distinguished: Either the key and foreign-key elements are part of the mapping or they are not part of the mapping. In case they are part of the mapping, the user already provides the information on how to establish a semantic relationship between the target values. If they are not part of the mapping, a schema mapping tool could automatically create key/foreign-key pairs for related elements. This can be realized with skolem-functions (see [PVM+02]).

- Nodes that are connected neither structurally nor foreign-key-based are **connectionless nodes**. Connectionless in this context means that they can be arbitrarily combined in a transformation from the source to the target. Connectionless nodes could, for example, be combined with a Cartesian product, an outer join, or listed in the order of their appearance in the instance of the source schema.

Additionally, we classified mapping situations, that result from **choice constraints**. A choice constraint specifies that in an instance a node might have only one child node from the set of nodes listed in the schema. A schema mapping tool should treat *source schema nodes with a choice constraint* similarly to nodes that are not mandatory. Unless all child nodes of a source node with a choice constraint are associated with the same target schema node, it cannot be guaranteed that a value is transferred into the target. The problem is then a lack of information, similar to the cardinality-related situations where the cardinality of the element in the source schema is lower than the cardinality of the element in the target schema. A schema mapping tool should recognize these situations and either reject the mapping or inform the user about the problem.

For a **target schema node with a choice constraint** it must be guaranteed that exactly one child node is produced. Accordingly, a mapping tool should recognize situations, where no
child node or more than one child node could be created and either reject those situations or warn the user.

A characteristic that is independent of a certain mapping situation, is the production of duplicates. A tool can either produce duplicate values or produce distinct values. Even though both solutions are correct, it would be desirable to give the user the choice, whether a tool should produce duplicates or not. For brevity, we foreclose the results of this aspect of schema mapping: All tools, except for one, produce per default duplicate values. Only two tools give the user the choice, whether to produce duplicates or not. One of them provides a special operator, whereas the other one requires the manual insertion of script.

3 Schema Mapping Tools

In this section we briefly describe the selection of six schema mapping tools from research and industry that we have evaluated. All tools feature a graphical user interface displaying source and target schema as a tree and allow to draw lines between elements of the schemata. The evaluation with respect to their capability to interpret schema mappings in the next section is anonymized, so this section serves only as a pointer to different tools and a comparison of their general capabilities.

**BizTalk Mapper 2004** is part of Microsoft’s BizTalk Server to manage business processes ([http://www.microsoft.com/biztalk/](http://www.microsoft.com/biztalk/)). Messages and data from different sources are converted to XML and the BizTalk mapper allows to specify transformations on this data. In addition to the pure mapping functionality, BizTalk Mapper offers a large library of “functoids” to transform data values. The mappings are interpreted as XSLT scripts.

**Clio** is not a commercially available tool, but a research prototype developed at IBM’s Almaden Research Center ([http://www.almaden.ibm.com/software/km/ клио/](http://www.almaden.ibm.com/software/km/ клио/)). It was one of the first and most sophisticated tools motivating many theoretical and practical research results. The product version of Clio is now part of Rational Data Architect.

**MapForce 2005** is part of Altova’s XML suite of tools to help users develop XML applications, which also includes XMLSpy ([http://www.altova.com/products/mapforce/](http://www.altova.com/products/mapforce/)). Like Clio, MapForce is a tool developed solely for the purpose of schema mapping and deriving transformation queries.

**Oracle Warehouse Builder 10g Release 1** is a tool to develop data warehouses based on the Oracle 10g database system ([http://www.oracle.com/technology/products/warehouse/](http://www.oracle.com/technology/products/warehouse/)). Part of this development is the ETL process (Extract, Transform, Load), which in turn includes a schema mapping step, among many others. We discuss this tool as a representative for many ETL tools offered by database and data warehouse vendors and companies specializing in ETL.

**Stylus Studio 6** is an XML integrated development environment by Progress Software specializing on XQuery / XSLT creation and visualization ([www.stylusstudio.com](http://www.stylusstudio.com)). With its XQuery Mapper developers can create and visualize mappings between XML schemata, which are in turn interpreted as XQueries or XSLT transformation queries.
**WebSphere Studio Application Developer Integration Edition** is IBM’s integrated development environment including a collection of tools to develop Web Services, Portals, etc. ([http://www.ibm.com/software/integration/wsadie/](http://www.ibm.com/software/integration/wsadie/)). One feature is a tool to map between XML Schemata, namely the XML-to-XML Mapping Editor.

Schema mapping is a research and development field of growing importance. Apart from the tools described here, there are several others including most ETL tools, the schema management prototype Rondo [MRB03], the SAP Exchange Infrastructure, etc. We also note that the tools are also still under development or are research prototypes that do not aim at full coverage of all possible situations. We have noted the versions we used and are aware of the fact that many of the observed problems are in fact bugs and are likely to be corrected in later versions. However, other problems reach deeper into the semantics of mapping and there is no easy or obvious “fix”. We believe that the set of experiments described in the next section are well-suited not only to document the state-of-the-art, but also to track progress in their development.

### 4 Evaluation

This section presents a set of tests to evaluate a schema mapping tools capability to cope with the mentioned situations. Section 4.1 introduces the overall evaluation procedure and criteria. Sections 4.2 – 4.4 then present a brief overview of the evaluation results. The evaluation is anonymized for legal reasons. In Sec. 4.2 schema mappings and evaluation results are exemplarily shown in more detail for situations where target leaf nodes are not part of the mapping. The schema mappings and detailed evaluation results for all other mapping situations are listed in [LN06].

#### 4.1 Evaluation procedure

To avoid side effects and to increase understandability, all mapping situations are tested in an as isolated as possible way. I.e., we have taken care that the situation under scrutiny is the only difficult situation in the schemata. To this end we developed 43 small schemata and mappings between them. All tools were exposed to these situations, if the situation was applicable to the features they support. For instance, the Oracle Warehouse Builder supports only relational data, so tests regarding nesting were not performed for that tool.

For each tool and each situation we loaded the two schemata, manually entered the appropriate correspondences, let the tool produce a transformation query, evaluated the query, and regarded the query result. To increase the comparability between the different mapping tools, all but the Oracle Warehouse Builder were tested using XML. The query language of the transformation query depends on the features of the tools. Most tools support more than one query language to express a transformation query. Wherever available we used XQuery.

We evaluate the results of the different tools with “+” for a satisfactory outcome, “o” for
an acceptable outcome, and “–” for an insufficient or incorrect outcome. Depending on the individual tests the scores were given based on different criteria: In general we favored results where little user-intervention was necessary. Tools that use all available schema and mapping information and generate a correct outcome are marked with a “+”. In many situations tools unnecessarily ask for user input, for instance asking how to join two elements despite an existing foreign-key relationship. In such cases, and when tools generate syntactically correct but semantically questionable results, we noted a “o”. Finally, incorrect results, i.e., transformed data that does not conform to the target schema, were marked with “–”.

4.2 Missing Correspondences

This class was tested with two sets of mappings: the first leaving out leaf nodes of the source schema (Fig. 5a) and the second leaving out leaf nodes of the target schema (Fig. 5b-5d). For the second set we present the full set of results as an example of the level of detail and the types of outcomes. Missing inner nodes were evaluated as part of the tests in Section 4.3. This section summarizes the results.

1. **Leaf node of the source schema is not part of the mapping.** As expected, all tools (correctly) ignore such leaf nodes if those nodes do not participate in the mapping.

2. **Leaf node of the target schema is not part of the mapping.** Such mapping situations were tested using three different schema mappings as shown in Fig. 5. The mapping in Fig. 5b was used to test mapping situations with key constraints, unique constraints, default values, and target leaf nodes that are not mandatory. The two other mappings in Fig. 5c and 5d were used to test mapping situations, where target leaf nodes with a foreign-key constraint are not part of any correspondence. The evaluation results are listed in Tab. 1. To summarize, only a single tool (Tool 1) performs well in all situations. This tool follows a passive conflict solving strategy by giving warnings to the user for almost all critical situations.

A second tool (Tool 2), on the other hand, follows an active conflict solving strategy by automatically solving the issues. However, its solutions are not always perfect. In particular, an available default value was ignored (see Row 4a in Tab. 1). The remaining four tools ignore the target schema attributes and create transformation queries that produce data not conforming to the target schema. They either do not produce the node at all (Tools 3 and 4) or produce it with a NULL value (Tools 5 and 6).

**Inner node is not part of the mapping.** Whether problems arise if inner nodes of a schema are not part of the mapping, mainly depends on the way a tool defines correspondences and interprets them.

Two of the tools do not allow inner nodes to be part of the mapping at all. Whereas three tools require correspondences between inner nodes in order to transform all instances of repeatable elements. If repeatable inner nodes are not connected, these tools transform only the first instance of this node from the source to the target.
4.3 Single correspondences

For the remainder of the evaluation we only briefly describe experiments and results here. The full evaluation results are listed in [LN06].

1:1 correspondences. The tests in this category are not exhaustive, because the number of all possible combinations between the different properties of a schema element is very high. Therefore we chose to test cardinality-related situations with correspondences only between leaf nodes and include correspondences between inner nodes only when required by the tool to properly generate results. The same was done with data type-related situations, because only leaf nodes can contain data. Furthermore, transformation functions were included only when testing data type-related mapping situations.

1. Type of participating schema elements. Of course, all tools support correspondences between leaf nodes of the source schema and the target schema. This is not true for correspondences between inner nodes of the participating schemata. As mentioned earlier, three tools require these kinds of correspondences in order to produce the correct result. The other three tools do not require them (two do not even allow them). The interpretations of correspondences between inner nodes in turn are very varied: One tool warns the user, if no child-element of the inner node is part of the mapping. Another tool forces the user to connect the child-leaf nodes in this case. Two tools produce subtrees without content in the target schema and the last tool just produces an instance of the inner node, without adhering to the target schema.

The connection of data level and metadata level via correspondences between inner
nodes and leaf nodes or vice versa is not supported by two schema mapping tools. All other tools are able to model and interpret the example shown in Fig. 3.

2. **Cardinality of the participating schema elements.** All tested tools have severe problems with cardinality-related mapping situations. Surprisingly, two schema mapping tools are not even aware of cardinalities. They produce the same transformation query regardless of possible conflicts. The other four tools recognize only few of the conflict situations.

3. **Data type of the participating schema elements.** The evaluation reveals that three schema mapping tools are wholly unaware of data types. Hence, they do not recognize problems with incompatible and partly compatible data types. The tools that are aware of data types in turn do recognize most conflict situations.

1:n, n:1, n:m correspondences. The evaluation was limited to n:1 correspondences, because none of the tools supports more than one target schema element as part of a single correspondence. For n:1 correspondences the main question is, how to combine the input values of the correspondence. The interpretations provided for these situations are similar to the interpretations given for multiple correspondences. They depend on the kind of connection between the participating source schema elements, which can be distinguished in structural connection, foreign-key-based connection and no connection (see Sec. 2.3).

Only one tool provides an algorithm that automatically detects foreign-key relationships. With another tool the user is forced to manually join the related subtrees. Three tools ignore the relationship and the last tool at least warns the user.

An automatic un-nesting of relationships between elements, that are structurally connected is performed by three tools where necessary. The other tools allow to manually un-nest these subtrees. For example, this can be done by connecting inner nodes in the source

<table>
<thead>
<tr>
<th>Situation</th>
<th>Tool 1</th>
<th>Tool 2</th>
<th>Tool 3</th>
<th>Tool 4</th>
<th>Tool 5</th>
<th>Tool 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (Key)</td>
<td>(+)</td>
<td>(+)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td></td>
<td>warning</td>
<td>produced with skolem-function</td>
<td>no node produced</td>
<td>no node produced</td>
<td>node with NULL value</td>
<td>empty node</td>
</tr>
<tr>
<td>2 (UNIQUE)</td>
<td>(+)</td>
<td>(+)</td>
<td>(+)</td>
<td>(+)</td>
<td>(+)</td>
<td>(+)</td>
</tr>
<tr>
<td></td>
<td>warning</td>
<td>produced with skolem-function</td>
<td>no node produced</td>
<td>no node produced</td>
<td>node with NULL value</td>
<td>empty node</td>
</tr>
<tr>
<td>3 (Foreign key)</td>
<td>(+)</td>
<td>(+)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td></td>
<td>warning</td>
<td>produced with skolem-function</td>
<td>no node produced</td>
<td>no node produced</td>
<td>node with NULL value</td>
<td>empty node</td>
</tr>
<tr>
<td>4a (Mandatory with default value)</td>
<td>(+)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td></td>
<td>used default value</td>
<td>produced with skolem-function</td>
<td>no node produced</td>
<td>no node produced</td>
<td>node with NULL value</td>
<td>empty node</td>
</tr>
<tr>
<td>4b (Mandatory without default value)</td>
<td>(+)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
<tr>
<td></td>
<td>warning</td>
<td>produced with skolem-function</td>
<td>no node produced</td>
<td>no node produced</td>
<td>node with NULL value</td>
<td>empty node</td>
</tr>
<tr>
<td>5 (Not mandatory)</td>
<td>(+)</td>
<td>(+)</td>
<td>(+)</td>
<td>(+)</td>
<td>(+)</td>
<td>(+)</td>
</tr>
<tr>
<td></td>
<td>no node produced</td>
<td>no node produced</td>
<td>no node produced</td>
<td>no node produced</td>
<td>node with NULL value</td>
<td>empty node</td>
</tr>
</tbody>
</table>

Table 1: Target leaf node not part of the mapping
schema with inner nodes in the target schema. This way some tools are forced to generate for each node in the source instance a respective node in the target instance.

Not all tools support n:1 correspondences in which the participating source schema nodes are *not obviously connected* (see sample mapping in Fig. 6). Additionally, the tools that allow these correspondences do not provide satisfying solutions. They either produce non-target schema compliant data or do not transfer all source data into the target data.

**Figure 6: A 2:1 correspondence with not obviously connected source schema nodes**

### 4.4 Multiple correspondences

For mapping situations with multiple correspondences we distinguish structurally connected, foreign-key-based connected, and connectionless elements in source and target schema. For reasons of simplification we do not evaluate situations, where these kinds of connection between nodes are nested into each other. Nevertheless, even with these simple situations no schema mapping tool performs well.

The evaluation shows, that the combination of input values for multiple correspondences is almost identical to the results for n:1 correspondences (with a transformation function). Therefore this section regards only the combination of output values.

1. **Structurally connected target schema elements.** Only one tool gives the user the choice to decide whether to group the target data or not. One tool automatically groups the target data and three tools do not group the data, thus losing associations among the data values. The aggregation shown in Fig. 4 (Page 7) could not be modeled by any of the tools.

2. **Foreign-key-based connected target schema elements.** If the target nodes are connected with a foreign-key relationship and both the key and the foreign-key are not mapped, only one tool automatically creates the key/foreign-key pair. Another tool warns the user, that some information is missing to create valid target data. All other tools ignore the relationship and create data that is not compliant with the target schema.

3. **Connectionless target schema elements.** All tools treat connectionless target schema elements independently and transfer all source data into each of the target schema elements (according to the correspondences).

4. **Choice constraint.** Only two of the five mapping tools that support XML are aware of the choice content model defined in XML Schema. The other tools create the same transformation query for the choice model and the sequence model and therefore do not recognize possible issues.
5 Discussion and Outlook

In this article we presented two main contributions: A classification of schema mapping situations for relational and nested schemata and a comparison of a set of current schema mapping tools from research and industry using a benchmark-suite of schemata and correspondences. While this is only a snapshot of the state of the art of schema mapping and new versions of tools have already appeared during our study, the contributions have a lasting character and can be used as reference for future schema mapping tool development and evaluation.

At this point it is worthwhile to discuss the broad range between users creating a mapping by editing correspondences and the applications "intelligently" guessing what kind of mapping the user might intend. This trade-off is best exemplified in a mapping from two relations to a single relation in the target. Some tools can guess and suggest to join the two source tables using a known foreign key constraint between the two. Other tools might deliberately refrain from such interference and let only users explicitly state joins between tables. Therefore, the question is how much intelligence and automation is needed in a schema mapping tools.

In general, tools provide three methods for automation and thus to ease the burden of a user: First, schema matching techniques can suggest correspondences between source and target schema. Despite much research in the field of schema matching, the automatically detected correspondences can merely be suggestions to be confirmed or rejected by the user. Second, tools interpret a set of correspondences as a mapping making decisions whether to join two elements, whether to use inner or outer joins, how to group elements etc. I.e., the tools help specify the mapping in more detail, again allowing users to interfere and change this specification. Third, the tools automatically compile this specification into a transformation query in some query language. This compilation is always fully automatic – users can only manually adapt the resulting query if necessary.

We believe that for scenarios with very large schemata or with very many schemata a high degree of automation is indispensable. This automation must be combined with a high degree of flexibility, i.e., users must be able to correct and adapt the mapping at any point and time. With a schema mapping debugger recent work already points in this direction [CT06]. In this article we have highlighted the necessary abilities of tools to enable this degree of automation and have pointed out the current limitations.

Acknowledgments. This research was supported in part by the German Research Society (DFG grant no. NA 432).

References


463
Dissertationspreise
Pathfinder: XQuery Compilation Techniques for Relational Database Targets

Jens Teubner
Technische Universität München, Institut für Informatik
jens.teubner@in.tum.de

Abstract: Relational database systems are highly efficient hosts to table-shaped data. It is all the more interesting to see how a careful inspection of both, the XML tree structure as well as the W3C XQuery language definition, can turn relational databases into fast and scalable XML processors.

This work shows how the deliberate choice of a relational tree encoding makes the XML data model—ordered, unranked trees—accessible to relational database systems. Efficient XPath-based access to these data is enabled in terms of staircase join, a join operator that injects full tree awareness into the relational database kernel. A loop-lifting compiler translates XQuery expressions into purely algebraic query plans. The representation of iteration (i.e., the XQuery FLWOR construct) in terms of set-oriented algebra primitives forms the core of this compiler. Together, the techniques we describe lead to unprecedented XQuery evaluation scalability in the multi-gigabyte XML range. Pathfinder is an open-source implementation of a purely relational XQuery processor.

1 Introduction

The ubiquitous use of the XML file format to store, interchange, and process data raises an increasing demand to manage these data in a scalable manner. Not only since the initiative of the W3C to develop XQuery as a standard query language for XML, researchers around the globe ambitiously started to develop novel database techniques that can efficiently handle semi-structured data. The approaches pursued have been as diverse as the XML data themselves: new storage layouts can natively handle tree-structured data (e.g., [FHK+02, NvdL05]), new algorithms and index structures accelerate XPath navigation primitives (e.g., [BKS02, CSF+01]), and tree algebras reflect the intricate semantics of the XQuery language (e.g., [JLST01]).

In this work, we want to assess how far we can get without the construction of such new and complex software systems. The processing model of existing relational databases—bulk operations on sets of tuples—proves versatile enough to embrace the semantics of XQuery in a standards-compliant fashion. At the same time, the maturity of existing implementations provides unprecedented scalability with interactive query response times on multi-gigabyte XML instances.

To meet these scalability goals, we contribute the purely relational XQuery processing stack shown in Figure 1 which can turn any RDBMS implementation into a processor for XQuery.
A relational tree encoding, derived from the XPath accelerator encoding by Grust [Gru02], provides a true isomorphism between instances of the XML data model, ordered, unranked trees, and relational tables of tuples. If B-tree indexes on such tables are chosen deliberately, interactive query response times for XML data can be observed even on commodity RDBMS implementations.

The XPath performance of such a system can further be improved if the underlying DBMS kernel is made aware of properties inherent to the used tree encoding. Staircase join encapsulates such knowledge in a single database operator and can accelerate tree navigation by orders of magnitude.

Finally, we extend the processing stack to full XQuery compliance. The loop-lifting compilation procedure trades XQuery’s for iteration primitive for truly bulk-oriented operations in the relational system. By shifting the dynamic evaluation into the DBMS kernel, we make the scalability advantages of modern RDBMS implementations immediately accessible to process XQuery.

The Pathfinder XQuery compiler is a complete implementation of the techniques we describe here. Pathfinder is part of the MonetDB/XQuery system, which is found among the fastest XQuery processors in existence today.

Sections 2 to 4 in the following will sketch the components of the relational XQuery processing stack. We provide performance figures obtained with MonetDB/XQuery in Section 5, before we summarize in Section 6.

2 Relational Storage of XML Data

To losslessly store XML data in a relational system, we use range encoding, a variant of the schema-oblivious tree encoding proposed by Grust [Gru02]. We enumerate all tree nodes according to the XML document order to obtain the preorder rank \( \text{pre}(v) \) for each node \( v \). Further, we maintain \( \text{size}(v) \) as the number of \( v \)'s descendants and \( \text{level}(v) \), \( v \)'s distance from the document root. Two properties \( \text{kind}(v) \in \{\text{elem}, \text{text}, \text{comment}, \ldots\} \) and \( \text{prop}(v) \) (holding \( v \)'s tag name or textual content for text/comment nodes) account for the semantical information of each node. Figure 2 on the left illustrates this encoding for a small sample tree.

On range-encoded data, XPath location steps translate into simple region predicates. To exemplify, the XPath \( \text{descendant} \) axis becomes a range condition on preorder ranks:

\[
v \in c/\text{descendant} \iff \text{pre}(c) < \text{pre}(v) \leq \text{pre}(c) + \text{size}(c).
\]

Figure 2: Sample tree (with \( \text{pre} \) and \( \text{size} \) annotations) and its relational encoding.

---

1Pathfinder is available in open source at http://www.pathfinder-xquery.org/.
Figure 4: Partitioned B-trees provide efficient child navigation performance on range-encoded XML trees without the expensive maintenance of parent/child references (IBM DB2 v8.2 on a 2 × 3.2 GHz Intel Xeon system with 8 GB RAM; path //open_auction/bidder/increase).

The evaluation of such one-dimensional range predicates is well supported by existing (e.g., B-tree) index structures.

2.1 Off-the-shelf RDBMSs are Better at XPath than You Might Expect

Numbering schemes of this kind are known to provide very efficient support for axes with a recursive definition in XPath [Gru02]. At first sight, this does not hold for the important non-recursive axes child and parent, which require an additional predicate on column level to characterize their semantics, e.g.:

\[
v \in c/\text{child} \iff 
\text{pre}(c) < \text{pre}(v) \leq \text{pre}(c) + \text{size}(c) \land \text{level}(v) = \text{level}(c) + 1.
\]

(CHILD)

Earlier work [Gru02] had thus used explicit parent/child references to provide acceptable runtime behavior for non-recursive XPath axes. By using partitioned B-trees [Gra03] to index the relational XML storage, however, we can reach a similar performance without the additional storage overhead. The prepending of the level column to a B-tree on pre (to obtain a concatenated \(\langle\text{level}, \text{pre}\rangle\) B-tree) partitions the resulting B-tree into height(\(t\)) regions as shown on the right in Figure 3 (where height(\(t\)) denotes the total height of the XML document tree).

On such a partitioned B-tree, all children of a given context node appear within a single index partition and in ascending pre-order (i.e., document order). As we see in Figure 4, this leads to efficient child evaluation on range-encoded data without the storage overhead of explicit parent/child references. Similar uses of partitioned B-trees are found to accelerate other XPath idioms as well [Teu06].
3 XPath Evaluation on Relational Back-Ends

Further improvements of an RDBMS’s XPath performance can be reached if we make the system aware of the fact that underlying relational tables actually constitute the encoding of a tree. Staircase join encapsulates such knowledge in a single database operator that may easily be plugged into existing RDBMS implementations. Tuned for the evaluation of XPath, staircase join largely avoids to spend work on irrelevant tuples, which brings execution times down to a minimum.

Figure 5 uses the two-dimensional pre/post plane\(^2\) and the XPath descendant axis to illustrate the three techniques that make staircase join an efficient means to answer XPath queries:

**Pruning.** Since XPath demands the result of location paths to be returned without duplicates, some context nodes may not contribute any new matches to the result set. In the pre/post plane, this surfaces as an overlap of their corresponding query regions. Staircase joins prunes such nodes early from the context set. This may significantly reduce the cost to eliminate duplicates from a path result.

**Partitioning.** After pruning the context set, the resulting query region takes the shape of a staircase in the pre/post plane. Staircase join divides this region into a distinct partition for each remaining context node. Each partition is scanned only once and in pre order. Regardless of the context set, the size of the document relation is now an upper bound for the number of tuples that need to be processed. The production of result tuples in document order obsoletes a subsequent sort operation as it was required in the original query plan (cf. Figure 4(b)).

** Skipping.** Since the pre/post plane actually constitutes the encoding of a tree, we can conclude that some regions in the plane cannot contain any nodes [GvKT03]. Staircase join skips over such regions, which further reduces the number of tuples to be processed from the document relation. The effect of skipping can be substantial: in earlier work [GvKT03], we found staircase join to skip over more than 90% of all tuples.

\(^2\)Note that the range- and pre/post-encodings are isomorph. Concepts equally apply to range-encoded data. In the pre/post plane, all descendants of a node \(v\) are to be found in the quadrant on \(v\)’s bottom-right.
Table 1: Subset of the relational algebra emitted by the loop-lifting compiler. Operator $\rho$ is the equivalent of SQL:1999’s `ROW_NUMBER` operator.

Staircase join encapsulates full tree awareness within a single join operator. This operator easily plugs into any existing relational database kernel. We have shown staircase join’s effectiveness with implementations for the MonetDB and PostgreSQL systems, for which we refer the reader to [Teu06].

4 Loop-Lifting: From XPath to XQuery

We have now seen how one of XQuery’s core data structures, ordered, unranked trees, can suitably be mapped to relational database tables. The second principal data type in the XQuery data model, ordered sequences of items, however, seems quite contradictory to the processing model of relational systems, unordered sets of tuples. Existing systems thus often tend to escape to a programming language outside the database kernel to implement language features that are sensible to this difference.

The loop-lifting compilation technique, in contrast, carries these tasks into the database kernel and leverages any RDBMS implementation to full XQuery support. Our approach remains purely relational: the compiler emits plans of a standard relational algebra (see Table 1 for an excerpt) whose operators are efficiently implementable on, e.g., SQL hosts. Note that this algebra operates on first normal form relations only. No XQuery-specific extensions (such as, e.g., the Map operators in [RSF06]) are required to back our compiler.

4.1 A Relational Representation for XQuery Sequences

The loop-lifting compiler represents any XQuery item sequence in terms of a relational table. The table shown on the right shows the relational sequence encoding of the XQuery sequence ("a", "b", "c", "d"). In this table, sequence order is maintained using column `pos`, while the actual sequence items are stored in column `item`. In line with the XQuery data model, we assume that column `item` can host atomic values as well as references to XML nodes (e.g., in terms of their preorder ranks $\text{pre}(v)$) in a heterogeneous fashion. See [Teu06] for ways to implement such a column.

4.2 Turning Iteration Into Joins

The heart of the loop-lifting compiler is the standards-compliant translation of XQuery’s iteration primitive, the `for-return` construct. This construct successively binds a variable $v$ to the items listed in its `in` part. The `return` body $e$ is then evaluated for each binding
and all sub-results are assembled to form the overall expression result:

\[
\text{for } v \text{ in } (x_1, x_2, \ldots, x_n) \text{ return } e \equiv (e[x_1/v], e[x_2/v], \ldots, e[x_n/v]).
\]

The semantics of this construct remains purely functional: it is sound to evaluate \( e \) for all bindings of \( v \) in parallel. The \( \text{iter|pos|item} \) relation shown here for the variable \( v \) reflects this situation and encodes all bindings of \( v \) in a single relation. This \( \text{loop-lifted sequence representation} \) is pervasive in our approach. Each tuple \( (i, p, x) \) in it indicates that, in the \( i \)th iteration, the item at position \( p \) has the value \( x \) (note that \( \$v \) is a singleton in the above expression, hence, \( \text{pos} \equiv 1 \)).

We can easily derive this representation of the binding variable from the representation of the expression it is bound to:

(i) attach a new \( \text{iter} \) column, consecutively numbered from 1, \ldots, \( n \) in the order given by the \( \text{pos} \) column, and then

(ii) set the \( \text{pos} \) column to constant 1.

The \( \text{row-numbering} \) step (i) is characteristic for this approach and we assume the availability of a respective operator \( \varrho_a((b_1, \ldots, b_n))|\text{c} \) to implement it. For each group identified by column \( c \), operation \( \varrho_a((b_1, \ldots, b_n))|\text{c}(R) \) extends \( R \) by a new column \( a \) that contains consecutive numbers in the order specified by \( (b_1, \ldots, b_n) \). Many RDBMSs readily provide an implementation for \( \varrho_a \). The construct \( \text{ROW\_NUMBER()} \text{ OVER} \ (\text{PARTITION BY} \ \text{c ORDER BY} \ b_1, \ldots, b_n) \), e.g., implements \( \varrho_a((b_1, \ldots, b_n))|\text{c} \) in SQL:1999 \[GST04\].

### 4.3 Independent Iterations

Note how column \( \text{iter} \) in the loop-lifted sequence representation enumerates the iterations performed by the \( \text{for} \) loop. It is a principle idea of our compilation approach that each subexpression is compiled in dependence of all enclosing \( \text{for} \) loops. To encode the latter, we use a unary \( \text{loop} \) relation, a projection of the loop-lifted encoding of the iteration variable on column \( \text{iter} \). The table on the left depicts the \( \text{loop} \) relation that encodes the \( n \)-fold iteration over the loop body \( e \) in the above example.

Once loop has been determined, we can use it to obtain the loop-lifted encoding of a constant subexpression by means of a Cartesian product.

We say that the expression is \( \text{lifted} \) with respect to loop. To illustrate, the table on the right encodes the sequence \( ("a", "b") \) in the loop

\[
\text{for } \$v \text{ in } (10, 20) \text{ return } ("a", "b") .
\]

To ensure compositionality, the full compilation procedure operates on loop-lifted sequence representations only. The compiler is defined in terms of a set of compilation rules, such that the algebraic expressions consumed and produced by each rule evaluate to the loop-lifted encodings of their respective XQuery equivalents, each one associated with a loop relation.
4.4 Compiling Arbitrary XQuery Expressions

The complete procedure to compile arbitrary XQuery expressions into their relational equivalent is beyond the scope of this paper (refer to [Teu06] for an extensive documentation). To provide an intuition of the typical plans emitted by the compiler, let us briefly review the compilation and evaluation of the XQuery expression

\[
\text{for } \$v \text{ in } (3, 4, 5, 6) \text{ return if (} \$v \mod 2 \text{ eq 0) then "even" else "odd".}
\]

\((Q_1)\)

The loop relation associated with the \texttt{return} body of this query is the relation shown on the left in Figure 6. This relation is used to compile the predicate subexpression \(e_1\). We omit the details of this compilation and show its outcome as the relation \(q_{e_1}\) in Figure 6. It contains the loop-lifted representation of a single Boolean value for each of the four iterations (in the third iteration, e.g., \(e_1\) evaluates to false).

Depending on the outcome of the predicate, we need to either evaluate the \texttt{then} branch \(e_2\) or the \texttt{else} branch \(e_3\). Two independent selections compute the respective sets of iter values (\(\sigma_{\text{item}}\) selects all tuples with value \texttt{true} in column \texttt{item}, \(\sigma_{\neg \text{item}}\) selects the complement) which are used to loop-lift the respective branches. Figure 6 shows the two relations \(\text{loop}_{\text{then}}\) and \(\text{loop}_{\text{else}}\). Cartesian products yield the loop-lifted encodings \(q_{e_2}\) and \(q_{e_3}\) of the subexpressions \(e_2\) and \(e_3\), respectively. The result of the \texttt{return} clause is their disjoint union \(q_{e_2} \cup q_{e_3}\) shown on the right in Figure 6.

Observe how the intermediate result \(q_{e_1}\), the loop-lifted encoding of the predicate expression \(e_1\), is consumed by two different sub-plans in Figure 6. This plan sharing is characteristic for query plans emitted by a loop-lifting compiler. The optimizer component of the Pathfinder XQuery compiler has thus been explicitly tuned to handle graph-shaped plans [RTG07].

4.5 Optimizing and Evaluating Loop-Lifted XQuery Plans

The loop-lifting compilation procedure turns arbitrary XQuery expressions into a query plan composed of a rather standard set of algebraic operators (see Table 1). Besides the scalability advantages that result from this approach, the use of relational algebra as an
equivalent representation for XQuery expressions can help to solve a number of problems that proved hard on the level of the XQuery language:

*Indifference of Order.* Different notions of order are wired deeply into the XQuery language (document order, sequence order, and iteration order). In loop-lifted query plans, this surfaces as the maintenance of iter and pos information throughout the plan.

There are many situations in XQuery, however, where order does not matter to the outcome of a query, e.g., in the inputs of existential predicates or the context sets of XPath location steps. In the relational plans, this usually means that an iter or pos column generated for some XQuery subexpression is never inspected by any upstream plan operator. The Pathfinder compiler uses a specific variant of projection pushdown to counter this effect, such that order information is only generated if indeed prescribed by the semantics of the query [GRT07].

*Robust Join Detection.* Since, in XQuery, there is no explicit join construct, the syntactical variations to express value-based joins are quite diverse. Based on the inference and inspection of functional dependencies, the Pathfinder compiler recognizes join situations in loop-lifted XQuery evaluation plans. This recognition is independent of syntactical variations and will detect, e.g., the value-based join in let $d := fn:doc (\cdots)$ for $a$ in $d/a$ return $d//b[@c = a/@d]$ [RTG07].

*Dependable Cardinality Estimates.* The availability of dependable estimates for (intermediate) result sizes can significantly improve query optimization and execution, e.g., to efficiently allocate resources in the physical plan. Unfortunately, the determination of such estimates is hard on the basis of the XQuery language. Existing techniques cover only rather limited subsets of the language.

In contrast, cardinality inference for relational query plans is a well-investigated field in database research. Loop-lifting makes this work immediately accessible to the estimation of result sizes for arbitrary XQuery expressions. Depending on the workload, this approach can be a suitable means to infer cardinality estimates for XQuery [Teu06].

## 5 Experimental Assessment

The prime motivation to re-use relational database technology for XML query processing was the expected scalability that we can inherit from mature RDBMS implementations. Pathfinder is a full implementation of the loop-lifting compilation procedure. Together with a staircase join extension to the MonetDB database kernel, it constitutes the open-source XQuery implementation *MonetDB/XQuery* [BGvK+06].

We used the XMark benchmark [SWK+02] and MonetDB/XQuery version 4.10.2 to verify whether the system indeed meets our scalability goals. At the top of Figure 7, we listed the query execution times (in milli-seconds) required to process the 20 XMark queries on a 111 MB XML instance (the system used for testing was equipped with 2 × 3.2 GHz Intel Xeon processors and 8 GB of main memory).

We further measured execution times on XMark instances of different sizes. Normalized to the elapsed times observed for the 111 MB instance, the resulting figures are illustrated in
Figure 7. MonetDB/XQuery scalability with respect to document size. Figures on top: execution times on a 111 MB XMark instance. Execution times in the graph are normalized to these figures.

Over a large range of document sizes, we see a linear scaling with the document size, the only real outliers being Queries Q11 and Q12. Both queries follow a quadratic scaling that stems from an intermediate result with quadratic complexity. For more in-depth experimental studies on the techniques we described, we refer to the experiments performed in [Teu06].

6 Summary

Our work demonstrates once more the versatility of the relational data model. We have shown how relational database systems can serve as efficient hosts to process XQuery. To suitably store the underlying XML data, we described range encoding, a variant of the XPath accelerator encoding developed in earlier work. A novel join operator, staircase join, provides an efficient implementation for XPath navigation steps over encoded tree data.

Our key contribution that allows the execution of arbitrary XQuery expressions on relational back-ends, however, is the loop-lifting compilation technique. Using a suitable encoding for XQuery’s basic data type, sequences of items, the loop-lifting technique turns the for iteration primitive into a bulk-oriented execution strategy on the relational system.

To demonstrate the effectiveness of the resulting relational XQuery processing stack, we used the software developed in the context of the Pathfinder and MonetDB/XQuery companion projects. We showed how MonetDB/XQuery reaches linear scaling and interactive query response times beyond the gigabyte XML size limit.

References


TopX – Efficient and Versatile Top-k Query Processing for Text, Semistructured, and Structured Data

Martin Theobald  Ralf Schenkel  Gerhard Weikum
{mtb, schenkel, weikum}@mpi-inf.mpg.de
Max-Planck-Institut für Informatik

Abstract: This paper presents a comprehensive overview of the TopX search engine, an extensive framework for unified indexing and querying large collections of unstructured, semistructured, and structured data. Residing at the very synapse of database (DB) engineering and information retrieval (IR), it integrates efficient scheduling algorithms for top-k-style ranked retrieval with powerful scoring models, as well as dynamic and self-throttling query expansion facilities.

1 Introduction

The Web increasingly moves away from the collection of unstructured text it was 10 years ago. Nowadays, the Web is a huge pile of documents that are not only heterogeneous in their content, but also in the level of annotation and structure they provide. Combining effective and efficient search over such heterogeneous collections within a single search engine will remain a major challenge, especially when the structure of documents, like in XML with potentially diverse schemata, hierarchical embeddings, and semantic annotations, should be exploited by queries and be taken into account for result ranking.

The TopX engine aims to solve this issue, addressing and seamlessly integrating recent trends of integrating DB and IR [CRW05]. It is a comprehensive framework for unified indexing and querying large collections of unstructured, semistructured, and structured data, comprising a full-fledged solution for ranked retrieval on desktops or intranets with annotated text or semistructured data, and, ultimately, the Web. It comes with a flexible, yet powerful and self-throttling query relaxation and/or expansion technique as an adequate means for coping with the inevitable diversity when merging various data sources that provides a controlled influence on the result ranking. While the current implementation focuses on IR-style search, the proposed methods and results can be carried over to various application areas like multimedia similarity search on high-dimensional feature vectors of images, music, or video, or preference queries over structured data such as product catalogs or customer support data in a very straightforward way.

TopX seamlessly integrates efficient query evaluation and versatile scoring models for ranked result output residing at the very synapse of database (DB) engineering and information retrieval (IR). As for the DB point-of-view, we aim at providing an efficient algorithmic basis for scalable, top-k-style processing of large amounts of data. Our focus lies on adaptive, disk-oriented cost models for accessing large, disk-resident index structures, with highly developed solutions for storing and efficiently querying large document collections (possibly in the order of Terabytes). As for the IR point-of-view, TopX provides a whole bunch of state-of-the-art, effective scoring approaches for Web IR, multi-attribute structured data, and ranked XML retrieval including XML full-text search. It supports an efficient, self-throttling query expansion mechanism that helps to increase effectiveness for difficult queries in terms of both recall as well as precision at the top ranks.
2 TopX System Overview

Figure 1 depicts the TopX main components. TopX comes with different, general-purpose Web and file crawlers for indexing text and XML data in a generalized schema on top of a relational back-end (which is described in the following for Oracle 10g but could easily be generalized to arbitrary DBMS or even a completely customized index using inverted files). The TopX core query processor is in charge of the bookkeeping of intermediate results and coordinates the sequential and random index list accesses in a multi-threaded architecture. It provides the algorithmic basis for exact and efficient top-\(k\) query evaluations with early threshold termination, with the option of gradually plugging in specialized components for probabilistic candidate pruning and cost-oriented index access scheduling that help to substantially accelerate query executions, as well as dynamic query expansion for IR-style vague search. Different levels of probabilistic extensions can stepwisely be incorporated for the probabilistic components, including basic selectivity estimators, more sophisticated index list histograms, or parameterized score estimators with convolutions for aggregated scores, index list correlations, and structural selectivity estimators for basic XML patterns such as twigs and path structures.

TopX provides both an interactive Web frontend for human users and a Web Service API to be used by other applications. An online live demo of the system is available at http://infao5501.mpi-inf.mpg.de:8080/topx. TopX is completely implemented in Java and deployable as a Tomcat servlet, with a code base of approximately 37,500 lines of code in 198 classes. It is available as a full-featured open-source package from http://topx.sourceforge.net.

3 Querying Text Data

3.1 Index Structures

As for plain text retrieval, queries consist of a set of (possibly weighted) keywords or terms. The result of such a query is a ranked list of documents. TopX precomputes, for each term, an index list by which one can access the document identifiers in descending order of the “local” score with regard to this term, for example, the TF-IDF- or BM25-
based score [GF05]. These lists are stored on disk in a relational Oracle database in a
table TextFeatures (term, docid, score), where docid is a numerical id of documents.
Efficient sequential or sorted access (SA) and random access (RA) on top of the DBMS is
supported by two B⁺-trees on the attributes concatenated in the order (term, score, docid)
for SA and in the order (docid, term, score) for RA. Additionally, optional index list meta
data such as score histograms and term correlation statistics are precomputed and stored
in the database and may be leveraged to accelerate query executions.

3.2 Core Query Engine

In order to find the top-\( k \) matches for multidimensional queries, scoring, and rank-
ing them, TopX adopts and substantially extends variants of the best-known, general-
purpose algorithm for evaluating top-\( k \) queries, Fagin’s family of threshold algorithms
(TA) [FLN01]. These algorithms leverage the observation that sequential disk I/O (i.e.,
sorted index scans) largely benefit from asynchronous prefetching and a high locality in
the hardware’s and processor’s cache hierarchy; so sorted access has much lower amor-
tized cost than random disk access that is inevitably requiring additional index structures
and key lookups for individual object identifiers.

TopX scans all relevant index lists in an interleaved manner; for efficiency, sorted ac-
cesses are performed in batches of fixed size \( b \). In each scan step, when the engine sees
the score for a document in one list, it is hash-joined with the partial scores for the same
document previously seen in other index lists and aggregated into a global score. The al-
gorithm maintains the worst score among the current top-\( k \) results and the best possible
score for all other candidates and documents not yet encountered, by adding to the worst
score the current top scores of all lists where the candidate has not yet been seen. The best
score serves as a threshold for pruning a candidate when its best score does not exceed
the worst score of the currently \( k^{th} \) ranked result; the index scans are stopped when no
candidate exceeds this threshold.

**Efficient Candidate Queuing:** While this focus on inexpensive sequential scans of-
ten results in a good query performance, it leaves uncertainty about the final scores of
candidates and therefore implies some form of bookkeeping or queuing not only for the
intermediate top-\( k \) results, but for all candidates that may still qualify for the final top-\( k \).
We investigated various queuing options for efficient candidate management in real-world,
large collection setups; it turned out that usually, the most effective approach is the combi-
nation of a queue of bounded length with a hash-based cache. Albeit a heuristic, the queue
bound \( q \) may be chosen in the order of the batch size \( b \) which is typically a safe choice.
Then testing the top candidate in the queue with the \( k^{th} \) ranked top-\( k \) item allows for a
lightweight, any-time threshold test for algorithm termination [TWS04].

**Index Access Scheduling:** In addition to the sequential disk accesses for sorted index
scans, the query engine also has the option of performing random accesses to directly re-
solve score uncertainty of documents. Altogether, this entails scheduling for the two types
of disk accesses: 1) the prioritization of different index lists in the sequential accesses,
and 2) the decision on when to perform random accesses and for which candidates. Both
types involve highly specialized probabilistic cost models, thus leading to individual in-
dex access scheduling decisions for result candidates which can substantially improve the
performance of the retrieval engine with no loss in result quality [BMT+06b].

**Probabilistic Candidate Pruning:** Since the user’s goal behind a top-\( k \) query usually
is not to find exactly the \( k \) best data items with regard to some ranking model, but rather
to incrementally explore a topic and to identify one or a few relevant and novel pieces
of information, it is intriguing to allow IR-style, approximate variants of the threshold family of algorithms to reduce runtime costs. TopX provides an optional approximate top-\(k\) algorithm based on probabilistic arguments [TWS04]. When scanning index lists, various forms of convolutions over underlying score distributions and derived bounds are employed to predict whenever it is safe, with high probability, to drop candidates and to prune the index scans for early algorithm termination. These probabilistic candidate pruning techniques can provide up to two orders of magnitude performance gains with a controllable loss in result quality and a very good quality/runtime ratio.

**Multi-threaded Query Processing:** The query processor is organized as a triple-tier, multi-threaded hierarchy consisting of a single main thread that iteratively maintains the data structure for candidate bookkeeping and optionally updates probabilistic predictors for candidate pruning and adaptive scheduling decisions after a batch of \(b\) index accesses; scan threads that continuously read and join input tuples from the inverted list buffers for a batch of sorted accesses, and buffer threads that continuously refill a small buffer cache and control the actual disk I/O for each inverted list. This three-level architecture builds on the observation that candidate pruning and scheduling decisions are computationally expensive and should be done only iteratively, because joining and evaluating score bounds for candidate may incur high CPU load, while the actual sequential index accesses are not critical in terms of CPU load. Synchronization (i.e., object locking) for shared data structures only takes place when a candidate is pulled from the cache and the queue is updated, or when (occasionally) a new candidate is promoted into the top-\(k\) queue.

**Top-\(k\) Shifts:** The TopX query processor supports incrementally increasing the number of query results, \(k\), without the need for restarting the whole query, thus addressing the typical behavior of a human surfer who may first look and digest the first page of 10 results, then eventually browse through the next page, and so on. In this interactive setting, in-memory candidate pruning is disabled; whenever \(k\) is increased, the currently best \(k\) candidates are identified and the scan threads continue scanning until the new top-\(k'\) candidates are found.

4 Querying XML Data

In XML IR, retrievable units are no longer whole documents but individual XML elements. Query languages for XML IR like W3C’s XQuery 1.0 and XPath 2.0 Full-Text or NEXI [TS04] combine content conditions on elements with structural conditions like tag names and paths. A typical example for such a content-and-structure (CAS) query is the NEXI query

\[
//\text{article}*[//\text{section}[about(.,'"XML database'') and about(.,'"architecture'')]]
\]

that searches for article sections about XML databases (which is a content condition) that include a figure about the architecture (i.e., that has this term somewhere in its content). As a special case, content only (CO) or wildcard queries like

\[
//*[about(.,'"XML'')]
\]

do not restrict the tag of target elements – similarly to keyword queries in text retrieval – but continue to return XML elements instead of whole documents. To search a collection with a heterogeneous or complex schema, a user would probably start with a CO query and would then refine it to a CAS query, either manually or with system support.

4.1 Index Structures

Extending the per-term inverted lists from text IR, TopX indexes occurrences of terms in XML elements together with the corresponding tags, i.e., it maintains index lists for combined tag-term pairs [TSW05b]. Such a tag-term list consists of all elements with the
tag that have the search term in their content, together with the corresponding document’s id, some navigational information, the local score, and the maximal local score of any element in the same document within that list. Local scores are computed using a variant of the BM25 scoring model that has been adapted for XML, by considering individual element frequencies and element sizes instead of the document-based counterparts in classic text IR. All elements within the same document are grouped together and form a coherent element block in the inverted lists, which are then sorted by descending maximal element score. As navigational information that supports all XPath axes, we store the pre, post and level numbers using the XPath accelerator technique [Gru02], in order to implement a combined inverted index for XML content and structure in a compact way.

Inside the Oracle database, the tag-term lists are stored in a single index-only table (IOT), using the schema TagTermFeatures(docid, tag, term, score, maxscore, pre, post, level), that allows for efficient random index accesses to all instances of a tag-term pair within a document. For efficient sorted accesses, we build an additional B+-index over the full range of attributes, but in the order (tag, term, maxscore, docid, score, pre, post, level); a sorted scan then corresponds to an index range scan for a given (tag, term) key. As an additional enhancement, we use Oracle’s index key compression option to automatically truncate redundant index key prefixes and skip dispensable key prefix replications at the inner index nodes of the B+-tree structures.

4.2 XML Extensions to the Query Engine

For each tag-term pair occurring in the query, TopX scans the corresponding list in descending max-score order, reading complete blocks instead of single elements. Such a sequential block scan prefetches all tag-term pairs for the same document-id in one shot and keeps them in memory for further processing, which we refer to as sorted block-scans.

To answer a content-and-structure query, we need to find an embedding of the query structure, which may form a directed acyclic graph (DAG), into the document tree. This embedding is not necessarily unique, as there may be multiple valid embeddings per document, because several of its elements may match each of the tag-term conditions. Score bounds are then computed for each document, derived from the score of the best embedding found so far. TopX by default returns documents as entry points to answer a query, with the option to show for each document either only the best embeddings or all embeddings ranked by aggregated element scores. Internally, a hybrid processing technique combines document- and element-specific query processing in a single engine: in document mode the kth document’s score is used for pruning, in element mode the k’th best embedding (obtained from the k’ < k elements) determines the threshold which allows an even more aggressive pruning.

In non-conjunctive (aka. “andish”) retrieval, a result document (or subtree) should still satisfy most structural constraints, but we may tolerate that some tag names or path conditions are not matched. This is useful when queries are posed without much information about the underlying schema (which is typical for document-centric XML like articles etc.), so the structural constraints merely provide a hint on how the actual text contents should be connected. In conjunctive query mode on the other hand, i.e., when all content and structure conditions have to be matched, if C in the above example is not a valid descendant of A, we may already safely prune the candidate document from the priority queue. TopX provides different approaches to judiciously schedule random accesses for testing the structural query conditions only for the most promising candidates according to their content-related scores [TSW05b] obtained through inexpensive sequential disk I/O.
5 Querying Structured Data

TopX treats data with a well-defined structure (e.g., data from a relational database, but also data-centric XML) as a special case of semistructured data, conceptually mapping non-XML data to a virtual XML document. As an example, for relational data, each tuple of a table corresponds to a document, and its attribute-value pairs are mapped to child elements of this document’s root. Retrieval then considers only the XML-ified data, where queries are conjunctions of elementary conditions of the form \( A = B \) (A is an attribute name, B is a value) that are automatically mapped to the corresponding XML queries.

6 Dynamic Query Expansion

TopX can utilize automatic query expansion to enhance result quality for difficult queries where good recall and/or precision at the top ranks is a problem. For difficult text queries like the ones in the TREC Robust track [TRE], e.g., queries for “transportation tunnel disasters” or “ship losses”, query expansion needs to integrate external knowledge. State-of-the-art approaches use one or a combination of the following sources to generate additional query terms: thesauri such as WordNet with concept relationships and some form of similarity measures, explicit user feedback or pseudo relevance feedback, query associations derived from query logs, document summaries such as Google top-10 snippets, or other sources of term correlations. In all cases, the additional expansion terms are chosen based on similarity, correlation, or relative entropy measures.

6.1 Ontology Service

The TopX Ontology Service provides unified access to multiple thesauri or ontology sources such as WordNet or OpenCyc in a compact API. Internally, the ontology service maintains a concept graph where the nodes denote the meaning of a set of terms and the edges are form the semantic relationships between concepts (like hypernyms, meronyms, antonyms, etc.) as provided by the different sources. The similarity of two related nodes is estimated by the correlation of the corresponding terms in a large corpus, e.g., using statistical measures such as Dice coefficients. The Ontology service further facilitates methods to disambiguate a term in a given document context and for finding related concepts that are similar to it.

6.2 Static vs. Dynamic Expansion

Static query expansion approaches typically choose a fixed set of expansion terms from an external knowledge base based on a predefined similarity threshold. However, such a static expansion technique faces three major problems [BZ04]: (1) the threshold for selecting expansion terms needs to be carefully hand-tuned for each query, (2) an inappropriate choice of the threshold may result in either not improving recall (if the threshold is set too conservatively) or in topic dilution (if the query is expanded too aggressively), and (3) the expansion may often result in queries with more than 50 or 100 terms, which in turn leads to very high computational costs in evaluating the expanded queries.

TopX addresses these issues and provides a practically viable, novel solution [TSW05a]. Our key techniques for making query expansion efficient, scalable, and self-tuning are to avoid aggregating scores for multiple expansion terms of the same original query term and to avoid scanning the entire index lists for all expansion terms. For example, when the term “disaster” in the query “transportation tunnel disaster” is expanded into “fire”, “earthquake”, “flood”, etc., we do not count occurrences of several of these terms as additional...
evidence of relevance for a document. Rather than that, we use a max-score aggregation function that counts only the best match of a document out of all possible expansions for the same original query concept, optionally weighted by the similarity of the expansion to the original concept. Furthermore and most importantly for efficiency, we open scans on the index lists for expansions as late as possible, namely, only when the best possible candidate document from a yet unseen part of the expanded index lists may still contribute to the top-k results.

The algorithm conceptually merges the index lists of the expansion terms with the list of the original query term in an incremental, on-demand manner during the runtime of the query in descending order of scores. For further speed-up, probabilistic score estimations can be used as well. The great advantage of this method is that we do not have to pre-determine the depth of the scanning and merging process itself. The superordinate top-k operator just incrementally inquires the subordinate Incremental Merge operator for the next document on-the-fly, thus preserving the efficient sorted access paradigm.

7 Nested Top-k Operators

For more sophisticated query sub-conditions such as phrase expansions, local scores for individual conditions cannot be fetched from materialized index lists but need themselves to be computed dynamically. This poses a major problem to any top-k algorithm that wants to primarily use sorted accesses. A possible remedy would be that the global top-k operator “guesses” a value \( k' \) and asks the dynamic source to compute its top-k results upfront, with \( k' \) being sufficiently large so that the global operator never needs any scores of items that are not in the local top-k.

TopX treats such situations by running a nested top-k operator on the dynamic data source(s), which iteratively reports candidates to the caller (i.e., the global top-k operator), and efficiently synchronizes the candidate priority queues of caller and callee. The callee starts computing a top-\( \infty \) result in an incremental manner, using a TA-style method itself without a specified target \( k \), hence top-\( \infty \). It gradually builds a candidate queue with upper and lower score bounds for each candidate. The caller periodically polls the nested top-k operator for its currently best intermediate results with their score bounds and integrates this information into its own bookkeeping. From this point, the caller’s processing simply follows the standard top-k algorithm (but with score intervals instead of scores). Note that this architecture also allows for a special Boolean – but ranked – retrieval mode, e.g., enforcing conjunctions at the top-level top-k operator while allowing disjunctions at the leaf operators for high-dimensional expansion tasks.

8 Evaluation

We focus our experiments with textual data on the TREC Terabyte collection which is the largest benchmark corpus currently being available with relevance assessments, consisting of about 25 million documents with a size of roughly 425 GB, with 50 reference queries from the 2005 Terabyte Ad-Hoc task. For XML, we chose the new 6 GB INEX Wikipedia collection with about 660,000 XML-ified Wikipedia articles and the respective batch of the 125 INEX 2006 Ad-Hoc queries. On a mainstream server machine with a dual XEON-3000 CPU, 4GB of RAM, and a SCSI RAID-5, indexing these collections took between 280 minutes for INEX and 14 hours for Terabyte, including stemming, stopword removal, and computing the BM25-based scores. The materialization of the B+ -indexes required roughly the same amount of time as it included sorting a huge intermediate table.

As for efficiency, we consider abstract query execution costs defined as \( \text{cost} = \#SA + \)
\( c_R / c_S \# RA \), i.e., a weighted sum of the number of tuples read through sorted and random accesses from our disk-resident index structures, as our primary metric analogously to [FLN01]. The cost ratio \( c_R / c_S \) between a single sorted vs. a single random access has been determined to optimize our runtime figures at a value of 150, which nicely reflects our setup using Oracle as backend and JDBC as connector, with a relatively low sequential throughput but good random access performance because of the caching capabilities of the DBMS. Wallclock runtimes were generally good but much more sustainable to these very caching effects, with average CPU runtimes being in the order of 0.3 seconds for Wikipedia and 1.2 for Terabyte, and wallclock runtimes being 3.4 and 6.2 seconds, respectively. All the reported cost figures are sums for the whole batch of benchmark queries, whereas the precision figures are macro-averaged.

### 8.1 Terabyte Runs – Efficiency vs. Effectiveness

Figure 2 compares the cost figures for TopX on the Terabyte setup with a DBMS-style merge join that first joins all documents in the query-relevant index lists by their id and then sorts the joined tuples for reporting the final top-\( k \) results (eventually using a partial sort). For \( k = 10 \), the non-approximate TopX run with the conservative pruning already outperforms the full-merge by a factor 5.5, while incurring query costs of about \( 9,323,012 \) compared to \( 54,698,963 \) for the full-merge. Furthermore, we are able to maintain this good performance over for a very broad range of \( k \); only queries of considerably more than 1,000 requested results would let our algorithm degenerate over the full-merge approach.

The approximate TopX with a relatively low probabilistic pruning threshold of \( \epsilon = 0.1 \) generally performs at about 10 percent lower execution costs than the exact TopX setup which confirms exactly to the pruning behavior we would expect and the probabilistic guarantees for the result quality we provide in [TWS04].

![Figure 2: Query execution costs for Terabyte as functions of \( k \).](image1)

![Figure 3: Relative vs. absolute precision for Terabyte as functions of \( \epsilon \), for \( k = 10 \).](image2)

Figure 3 investigates the probabilistic pruning behavior of TopX for the full range of \( 0 \leq \epsilon \leq 1 \) for a fixed value of \( k = 10 \), with \( \epsilon = 1.0 \) (i.e., the extreme case) meaning that we immediately stop query processing after the first batch of \( b \) sorted accesses. Since Terabyte is shipped with official relevance judgments, we are able to study the result quality for both the relative precision (i.e., the overlap between the approximate and the exact top-\( k \)) and the absolute precision (i.e., the fraction of results officially marked as relevant by a human user for a particular topic). We see that the relative precision drops much faster than the absolute precision which means that, although different documents are returned at the top-\( k \) ranks, they are mostly equally relevant from a user perspective. Particularly remarkable is the fact that the ratio of the execution cost between the approximate and the exact top-\( k \) generally drops at a much faster rate than both the absolute and relative precision values. This observation holds for all collection and query setups we considered so far.
8.2 Wikipedia Runs – Efficiency vs. Effectiveness

As for Wikipedia, we provide a detailed comparison of the CO and CAS interpretations of the queries. Figure 4 shows that we generally observe similarly good performance trends as for Terabyte, with cost-savings of a factor of 7 for CAS and 2.5 for CO, and the performance advantage remains extremely good even for large values of $k$, because we never need to scan those long (i.e., lowly selective) element lists for the navigational query tags which would have to be performed by any non-top-$k$-style algorithm.

Figure 4: Query execution costs for Wikipedia as functions of $k$. Figure 5: Relative precision for Wikipedia as functions of $\varepsilon$, for $k = 10$.

Figure 6 depicts a detailed comparison of the query costs being split into individual sorted (#SA) and random (#RA) disk I/Os for the CO and CAS flavors of the Wikipedia queries. It shows that we successfully limit the amount of RA to less than about 2 percent of the #SA according to our cost model. This ratio is maintained also in the case of structured data and -queries which is a unique property among current XML-top-$k$ engines (see also [TSW05b] for a comparison to state-of-the-art competitors). Figure 7 shows an impressive runtime advantage for the dynamic query expansion approach compared to both full-merge and TopX when performing static expansions (measured for the CAS case). The reported numbers reflect large, automatic thesaurus expansions of the original Wikipedia queries based on WordNet, with up to $m = 292$ distinct query dimensions (keywords). Figure 6 finally demonstrates a similarly good pruning behavior of TopX for both the CO and CAS queries in Wikipedia, again showing a very good quality vs. runtime ratio for the probabilistic candidate pruning component.

Figure 6: #SA and #RA for full-merge vs. TopX, for $k = 10$ and $\varepsilon = 0$. Figure 7: #SA and #RA for full-merge vs. TopX, for $k = 10$ and $\varepsilon = 0$.

8.2.1 Further Experiments

TopX has been extensively evaluated throughout two years of active participations in the two major benchmark series in IR, using the largest available collections of the Text RE-
trieval Conference (TREC) [TRE] on text IR and the Initiative for the Evaluation of XML Retrieval (INEX) [M+06] focusing on XML IR. For INEX, the 2005 benchmark included a set of 40 keyword-only and 47 structural queries with relevance assessments that were evaluated on the INEX IEEE corpus. TopX performed very well for CAS queries, ranking among the top-5 of 25, with a peak position 1 for two of the five official evaluation methods. See [TSW06, BMT+06a, TSW07] for more detailed discussions of the results on various TREC and INEX tasks.

9 Conclusions and Future Work

TopX is an efficient and effective search engine for unstructured, semistructured, and structured data. Our future work will concentrate on (1) including linkage information in the retrieval process, especially for XML, (2) extending our top-k algorithms to support non-monotonous scores like proximity, and (3) implementing an efficient inverted file structure. TopX has been the official host used for the INEX 2006 topic development phase, and its Web Service interface will be used by the INEX 2006 Interactive Track. During the topic development phase, more than 20,000 CO and CAS queries from roughly 70 different participants world-wide were conducted partly in parallel sessions over the new Wikipedia XML index.

*Bibliography


484
Industrieprogramm
The Information Integrator: using Semantic Technology to provide a single view to distributed data

Jürgen Angele  Michael Gesmann
ontoprise® GmbH  crossvision Server Technologies
Amalienbadstr. 36  Software AG
76227 Karlsruhe  Uhlandstrasse 12
angele@ontoprise.de  michael.gesmann@softwareag.com

Abstract: For the integration of data that resides in autonomous data sources Software AG uses ontologies. Data source ontologies describe the data sources themselves. Business ontologies provide an integrated view of the data. FLogic rules are used to describe mappings between data objects in data source or business ontologies. Furthermore, FLogic is used as the query language. FLogic rules are perfectly suited to describe the mappings between objects and their properties. Some of these mapping rules can be generated automatically from the data sources metadata. Some patterns do frequently reoccur in user-defined mapping rules, for instance rules which establish inverse object relations or rules which create new object relations based on the objects’ property values.

Within our first project access to information is still typical data retrieval and not so much knowledge inference. Therefore, a lot of effort in this project concentrated on query functionality and even more on performance. But these are only first steps. To strengthen this development and to get more experience in this field Software AG recently joined several EU research projects which all have a focus on exploitation of semantic technology with concrete business cases.

1 Introduction

Data that is essential for a company’s successful businesses often resides in a variety of data sources. The reasons for this are manifold, e.g. load distribution or independent development of business processes. But data distribution can lead to inconsistent data which is a problem in the development of new businesses. Thus the consolidation of the spread data as well as giving applications a shared picture of all existing data is an important challenge. The integration of such distributed data is the task of Software AG’s “crossvision Information Integrator” one of the components in the crossvision SOA suite [crossvision].

486
Information Integrator is based on ontologies. The term ontology is used in a technical sense. That is, ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents in some field of knowledge [Gruber 1993]. Ontologies include the objects and all of the properties, relations, and functions needed to define the objects and specify their actions. Like other model languages, e.g. entity relationship models or SQL, ontologies can be used to describe structural constraints like existence of entity types and attributes or tables and columns now called classes and properties. They can also be used to express semantic constraints like referential integrity or uniqueness of property values. In addition rules as an integral part of the ontology definition can be used to describe more relationships between entities. These rules can be used by inference engines to generically derive information that does not persist in the existing data.

Within the Information Integrator we distinguish between data source ontologies on one hand and business oriented ontologies on the other hand. The first ones can be generated from metadata of underlying data sources. Currently, SQL databases, Software AG’s Adabas databases, and web services are supported types of data sources. The latter ones describe business domain while making use of other business ontologies or data source ontologies. FLogic rules describe the information how objects in different ontologies are related to each other.

Using ontologies Information Integrator solves three major problems. First of all it provides all means to integrate different information systems. This means that comfortable tools are available to bring data from different systems together. This is partially already solved by systems like virtual or federated databases [Batini et al. 1986]. Information Integrator is more powerful compared to most of these systems as it not only supports databases but additional sources like web services, applications etc. The second problem which is solved is that Information Integrator allows reinterpretation of the contents of the information sources in business terms and thus makes these contents understandable by ordinary end users and not only by database administrators. Finally this semantic description of the business domain and the powerful mapping means from the data sources to the business ontology solves the semantic integration problem which is seen as the major problem in information integration. It maps the different semantics within the information sources to the shared conceptualization in the business ontology.

Within Software AG Information Integrator was used for a first project Customer Information Gateway (CIG) whose mission was to integrate data that on one side resides in a support information system and on the other side is stored in a customer information system.

2 Conceptual Layering

Conceptually Information Integrator arranges information and the access to information on four different layers (cf. Figure 1):
• The bottom layer represents different data sources which contain or deliver the raw data which will be semantically reinterpreted on upper layers viz. ontologies. Currently relational databases, Adabas databases and web services are supported.

• The second layer assigns a so called “data-source ontology” to each of the data sources. These “data-source ontologies” can be created automatically from database or WSDL schemas of the data sources. The generation process translates schemas into ontology terminology. The generated ontologies contain rules which specify how a rule inference engine can access these data sources. For this rules make use of so-called connectors which do implement these access operations. From an ontological point of view these “data-source ontologies” are not real ontologies as they do not yet represent a shared conceptualization of a domain.

• The third layer represents the business ontology using terminology relevant to business users. This ontology is a real ontology, i.e. it describes the shared conceptualization of the domain at hand. It is a reinterpretation of the data described in the data-source ontologies and thus gives these data a shared semantics. As a consequence a mental effort is necessary for this reengineering of the data source contents which cannot be done automatically.

• On a fourth layer views to the business ontologies are defined. Basically these views query the integration ontology for the needed information. Exposed as Web services they can be consumed by portals, composite applications, business processes or other SOA components.

Figure 1: Conceptual Layering of Ontologies
The elements of the different layers are connected by so called mappings. The mappings between the data-sources and the source ontologies are created automatically, the mappings between the ontologies are manually engineered and the views are manually defined queries. Mappings define how source structures are mapped to destination structures. Thus mappings provide ways to restructure information, to rename information or to transform values. Up to now, we do not consider and do not plan to consider approaches which try to automatically derive such mappings [Rahm and Bernstein 2001].

This arrangement of information on different layers and the conceptual representation in ontologies and the mediation between the different models by mappings provide various advantages:

- The reengineered information in the business ontology is a value on its own. It represents a documentation of the contents of the data sources. The representation as an ontology is a medium to be discussed easily by non-IT experts. Thus aggregating data from multiple systems this business ontology provides a single view on relevant information in the user’s terminology. More than one business ontology enables different perspectives into the same information.

- It is easy to integrate a new data source with a new data schema into the system. It is sufficient to create a mapping between the corresponding source ontology and the integration ontology and thus does not require any programming know-how; pure modelling is sufficient. Already existing ontologies are not affected.

- The mediation of information between data sources and applications via ontologies clearly separate both. Thus changes in the data source schemas do not affect changes in the applications, but only affect changes in the mediation layer, i.e. in the mappings.

- This conceptual structure strongly increases business agility. It makes it very easy to restructure information and thus to react on changing requirements. Neither the data sources nor the applications have to be changed. Only the business ontology and the mappings have to be modified. Again no programming skills are required, all is done on a model level. Thus it minimizes the impact of change, eases maintenance and allows for rapid implementation of new strategies.

Ontologies have powerful means to represent additional knowledge on an abstract level. So for instance by rules the business ontology may be extended by additional knowledge about the domain. Thus the business ontology is a reinterpretation of the data as well as a way to represent complex knowledge interrelating these data. So business rules are directly captured in the information model, they determine the optimal system access and bring every user to the same level of effectiveness and productivity.
3 Tool Support / Architecture

The crossvision Information Integrator provides a full fledged tool environment for defining models, for mappings between these models and for running queries (cf. Figure 2). IntegratorStudio is an ontology engineering environment based on OntoStudio™.

It allows for defining classes with properties, instances of these classes and rules. Import capabilities generate “source ontologies” from underlying data sources. A powerful mapping tool allows users to interactively define mappings between ontologies by graphical and form based means (cf. Figure 3). Besides defining correspondencies between the data source ontologies and the business ontology it allows to describe functional transformations like value transformations. Rules may be defined with graphical diagrams (cf. Figure 4). IntegratorStudio supports F-Logic, RDF(S), OWL for import and export. In practice the different information sources contain redundant or even inconsistent information. We either use mentioned value transformations and/or additional rules to solve such problems. Queries which define the mentioned views can be generated and may be exported as web services.

Figure 2: Architecture of the crossvision Information Integrator
Figure 3: Mapping Tool in crossvision Information Integrator. The lower part defines the mappings, while the upper part allows to define functional transformations.

Figure 4: Graphical representation of a rule stating that if the customer number of an organization equals the contract number then this organization is owner of the contract.
SemanticServer, the reasoning system, provides means for efficient reasoning in F-Logic [Kifer, Lausen, Wu 1995]. SemanticServer performs a mixture of forward and backward chaining based on the dynamic filtering algorithm [Kifer, Lozinskii 1986] to compute (the smallest possible) subset of the model for answering the query. During forward chaining not only single tuples of variable instantiations but sets of such tuples are processed. It is well-known that set-oriented evaluation strategies are much more efficient than tuple oriented ones. The semantics for a set of F-Logic statements is then defined by a transformation process of F-Logic into normal logic (Horn logic with negation) and the well-founded semantics [Van Gelder, Ross, Schlipf 1991] for the resulting set of facts and rules and axioms in normal logic.

It is clear that our approach strongly relies on rules and reasoning about instance data. Compared to an approach using OWL + SWRL rules, F-Logic is the appropriate choice as reasoning over disjunctions and equalities as it is necessary in OWL + SWRL breaks down performance. Additionally OWL + SWRL relies on open world semantics while databases imply a closed world semantics. On the other hand OWL has its strengths in modelling ontologies. It provides more abstract modelling primitives, subsumption and constraint checking. So in future we additionally might provide a mixed approach, where OWL is mainly used for modelling ontologies and F-Logic for rules. It is clear that both semantics do not fit together and there must be some compromise to mix both approaches. For instance subsumption could be applied to reorganize the ontology and then this reorganized ontology is used in an F-Logic way, or cardinality restrictions could be reinterpreted as constraints to avoid equality reasoning.

Meta data like ontologies, their mappings, web service descriptions and meta information about data sources are stored in the CentraSite repository. Also, IntegratorStudio stores information about exported web services in CentraSite. During startup the inference engine SemanticServer which is based on OntoBroker™ loads the ontologies from the repository and then waits for queries from the exported web services. These queries are evaluated by SemanticServer and are online translated into calls to access connected data sources.

Thus SemanticServer represents the run-time engine, IntegratorStudio the modelling environment and CentraSite the meta data repository. SemanticServer is also integrated into IntegratorStudio thus enables immediate execution of queries to the ontologies.
4 Use Case: Customer Information Gateway

Within Software AG the Information Integrator was used for a first project whose mission was to integrate data that on one side resides in a support information system and on the other side is stored in a customer information system. The support information system maintains for example information about customers, their contact information and active or closed support requests. The customer information system stores information about clients, contracts etc. While the customer system stores its data in an Adabas database, the support system uses an SQL server. The integrated data view is exposed in a browser based application to various parties inside the company. For instance, instead of only seeing reported support requests support engineers shall get a more complete picture of a customer while talking with them.

For illustration purposes we first sketch a very simplified excerpt of imported data and the business ontology. Throughout the following examples we use FLogic syntax.

First of all there are two classes which have been generated by the mentioned automatic mapping from Adabas files:

```flogic
FILE151_CONTRACT[FILE151_AA=>string;
                FILE151_AE=>date]@Source151.

FILE152_CLIENT[FILE152_AA=>number;
               FILE152_AB=>string;
               FILE152_AC=>string]@Source152.
```

The rules define some schematic information. Within these rules the @-operator specifies a context or also called module where the rule applies. The =>-operator is a property definition specifying the name and the type of a property. Multiple property definitions can be separated by semicolons. The square brackets are boundaries for all the definitions of the named object that is given just before the opening bracket. So, the first rule says: within module “Source151” there is a class named “FILE151_CONTRACT” with two properties. One property is named “FILE151_AA” with type string, the other one is “FILE151_AE” with type date.

The cryptic names reflect the internal structure of Adabas files. The names “CONTRACT” and “CLIENT” have been specified by the user during the mapping process. Currently, it is application knowledge that FILE151_AA represents a contract number, FILE151_AE contract end dates, FILE152_AA is a customer id, FILE152_AB a customer name and FILE152_AC the customer address.

In our example we also consider two tables from the SQL database. Table “CUSTOMER” contains columns “id”, “name” and “addr”. Table “CASE” stores support requests with columns “caseId”, “status”, “customerId” and a foreign key “forCustomer”. The resulting class definitions are
CUSTOMER[id=>number;  
    name=>string;  
    addr=>string]@SourceSQL.

CASE[caseId=>number;  
    status=>string;  
    customerId=>string;  
    forCustomer=>CUSTOMER]@SourceSQL.

For the module “SourceSQL” reflecting schema information from the SQL database we have generated a data-source ontology consisting of two classes „CUSTOMER“ and „CASE“. These two classes are mapped from the SQL tables with the same names. In addition there are some property definitions derived from the tables’ columns.

On top of the data-source business ontologies we define a business ontology containing three classes:

Customer[name=>string;  
    address=>string;  
    supportRequests=>SupportRequest]@Business.

SupportRequest[id=>number;  
    status=>string;  
    issuedBy=>Customer]@Business.

Contract[contractId=>string;  
    endOfContract=>date;  
    endOfContractFormatted=>string]@Business.

In the sequel we present some examples on how we used rules within our ontologies and derive some requirements and use cases for rule languages to be used in such a project.

4.1 Data source import

The system needs to be open in a sense that it allows for extensions which provide access to external data sources. In Information Integrator built-in predicates which are implemented by the already mentioned connectors provide this functionality. In the sequel we abstract from a concrete syntax of these built-in predicates. Instead we illustrate this by a generic predicate “accessToSource”:

    accessToSource(connectionInfo, “tablename”, “rowid1”, X, “rowid2”, Y, …)

where connectionInfo describes all parameters that are needed to call the data source, tablename, rowid1, rowid2 are names of some database tables or table columns, respectively. X, Y are the name of a variables which are to be bound by the built-in predicate.

In our example there are four rules which import data from external data sources:
FORALL X, Y
c("FILE151", X) : FILE151_CONTRACT [ FILE151_AA -> X ; FILE151_AE -> Y ]
<- accessToSource(connectionInfo, "FILE151", "AA", X, "AE", Y) @Source151.

To explain the first one (the others are similar). The FORALL part lists a set of variables used within the rule. The ":"-operator is the instance-of operator. X:C says that X is an instance of class C. A P->V statement sets value V for property P. The combination X:C[P->V] says X is an instance of C with property P having value V. Finally, the rule consists of a body and a head meaning that if all the statements in the body are true, then also the information in the head is true. The other way around: if for instance an inference engine is interested on information specified in the head it can derive this information from the body.

The rule given above says: If reading information from an external data source by accessToSource with the given parameters “connectionInfo”, “FILE151”, “AA” and “AE” returns with variable bindings for X and Y, then the bindings for X and Y can be used to instantiate c("FILE151",X) as an instance of class “FILE151_CONTRACT” where value of X is bound to property “FILE151_AA” and the “FILE151_AE”-property receives value Y.

Similar rules exist for the other data sources within their respective data-source ontologies:

FORALL X, Y, Z
c("FILE152", X) : FILE152_CLIENT
    [ FILE152_AA -> X ; FILE152_AB -> Y ; FILE152_AC -> Z ]
<- accessToSource(connectionInfo, "FILE152",
    "AA", X, "AB", Y, "AC", Z) @Source152.

FORALL X, Y, Z
c("CUSTOMER", X) : CUSTOMER[ id -> X ; name -> Y ; addr -> Z]
<- accessToSource(connectionInfo, "CUSTOMER",
    "id", X, "name", Y, "addr", Z) @SourceSQL.

FORALL X,Y,Z
c("CASE", X) : CASE[ caseId -> X ; status -> Y ; customerId -> Z]
<- accessToSource(connectionInfo, "CASE",
    "caseId", X, "status", Y, "customerId", Z) @SourceSQL.

Import from data sources is easy, as long as the order of single result objects and the order of property values within a row are not significant. This is valid for imports from SQL and mostly also for import from Adabas. For data sources like web services which expect and return XML documents this is no longer true, because the order in which elements appear within a parent element might contain significant information. For example for chaining of web services, i.e. the result of one or more web services serves as input for another web service, it is necessary to maintain the structure of the original result documents.
A complex type named “Path” defined as a sequence of elements “Endpoint”, “Via” and again “Endpoint” with an arbitrary number of occurrences for “Via” shall serve as a simple example. The XML Schema type definition is

```xml
<complexType name = "Path">
  <sequence>
    <element name = "Endpoint"/>
    <element name = "Via" minOccurs="0" maxOccurs="unbounded"/>
    <element name = "Endpoint"/>
  </sequence>
</complexType>
```

The example instance is

```xml
<Path>
  <Endpoint>Darmstadt</Endpoint>
  <Via>Frankfurt</Via>
  <Via>Köln</Via>
  <Endpoint>Aachen</Endpoint>
</Path>
```

which is completely different from

```xml
<Path>
  <Endpoint>Aachen</Endpoint>
  <Via> Köln </Via>
  <Via> Frankfurt </Via>
  <Endpoint> Darmstadt </Endpoint>
</Path>
```

FLogic does neither natively support all the features of complex type definitions in XML schemas nor the ordered occurrence of property values, therefore these have to be expressed explicitly within the ontologies. Without going into too much detail the feature used for this are so-called parameterised properties. That means a value assignment for a property can be parameterised with another value, e.g. an order value following the Ordpath numbering scheme [O'Neil at al. 2004]. The parameter values are denoted in brackets behind a property name. For our simple Path example the following FLogic statement defines the type:

```fllogic
Path[Endpoint(string) => string ; Via(string) => string].
```

The next statement defines the intended ordering of our example instance:

```fllogic
SomePathName : Path[ Endpoint("1.1") -> "Darmstadt" ;
  Via("1.3") -> "Frankfurt" ;
  Via("1.5") -> "Köln" ;
  Endpoint("1.7") -> "Aachen"].
```
4.2 Object and property mapping

It is very easy to define that an object in one model representing the data source is also an object in another model representing the business model. For example a contract object in the customer information system is also a contract object in the business model:

FORALL X X:Contract@Business <- X:FILE151_CONTRACT@Source151.

In plain text: If a binding for variable X is an instance of class “FILE151_CONTRACT” in the “Source151”-ontology then it is also an instance of class “Contract” in the “Business”-ontology. Similar rules exist for FILE152_CLIENT and CASE. Like for objects it is also easily possible to specify that a property in one ontology maps to a property in another ontology and that all property values in a first ontology are also values of the mapped property in the second ontology.

FORALL X, Y, Z
  X : Contract[contracted -> Y ; endOfContract -> Z]@Business

If the underlying data from the external sources contains such information, it is also easily possible to describe that two objects are the same. For example a client in the customer information system and a customer in the support information system represent the same object, if these have the same name and address. Please note, in both systems clients and customers, respectively, have a surrogate values as unique keys. But typically these values are not a viable object identifier across independent data sources. Therefore, we need to identify new identifiers. The example in rule terms:

FORALL X, Y, Z
  c("Customer", Y,Z) : Customer[name -> Y ; address -> Z]@Business
  <- X : CUSTOMER[name -> Y; addr -> Z]@SourceSQL.

FORALL X, Y, Z
  c("Customer", Y,Z) : Customer[name -> Y ; address -> Z]@Business
  <- X : FILE152_CLIENT[FILE152_AB -> Y; FILE152_AC -> Z]@Source152.

These simple types of mapping are essential for specification of business ontologies on top of data source or other business ontologies. All of these mapping rules can be described in the Information Integrator with graphical means, i.e. developers do not need to see the FLogic syntax.

4.3 Property value mapping

Often similar data that is represented in one way in a first database can be represented in a different way in another database. For example:
• data that is encoded in a single column or field might be scattered across multiple attributes in another database (comma-separated name versus firstname and lastname; encoding of some numeric or boolean bits into a single bit array)
• data with different representation (time and date values as a number, as XML types, as SQL values)

In all these cases it is very helpful to have an extensibility which allows for adding functions that implement necessary transformations.

An example in rule terms:

FORALL X, Y Y[endOfContractFormatted -> X]@Business
    <- EXISTS Z (Y : Contract[endOfContract -> Z]@Business and date2string(Z, X)).

where date2string() is a predicate that transforms a date from one presentation into another one.

4.4 Object references and more metadata

Every functional model needs to describe relations between objects. Object properties are used to express these relationships in a model. Object identifiers are object property values which reference the object with the identifier.

These properties and property values are similar to foreign keys in relational databases. The foreign key information that is provided with the schema description can be used during generation of the data source model.

An example with a foreign key between table “CUSTOMER” (see above) and table “CASE” in rule terms:

FORALL X, Y X[forCustomer -> c("CUSTOMER", Y)]@SourceSQL
    <- X : CASE[customerId -> Y]@SourceSQL.

This has to be mapped to the business ontology:

FORALL X, Y, Z1, Z2
    X : SupportRequest[issuedBy -> c("Customer", Z1, Z2)]@Business
    <- X : CASE[forCustomer -> Y]@SourceSQL
       and c("CUSTOMER",Y)[name -> Z1 ; addr -> Z2]@SourceSQL.

Also, the inverse reference could be generated. But, while there is a name for the foreign key constraint in SQL databases which can be used for the generation, there is no inverse name in schemas. Therefore the creation of the inverse relation is currently postponed to application development:

FORALL X, Y X[supportRequests -> Y]@Business
    <- Y : SupportRequest[issuedBy -> X]@Business.
Even N:M relationships, which need to be implemented in two 1:N foreign key relations, can be expressed directly.

4.5 Queries

The learning of new languages is always a substantial investment, in particular if this involves the learning of new programming paradigms. Having different languages for the modelling and for the querying of ontologies bears the potential for impedance mismatches and causes additional costs. Therefore, rule language and query language should at best be the same. The Information Integrator uses FLogic not only as a language for ontology definitions but also as a query language. Because most application developers in the database area are not very familiar with FLogic and other logic languages the Information Integrator provides a graphical tool to define basic queries.

Like queries in database applications, the queries in our project shall provide some result information. It was not the goal to find all explanations, why the returned results are valid results, nor was it a goal to find all variable bindings that lead to a result.

Using FLogic as a query language leads to some typical database query language requirements, for example:

- User-defined projections on the query result should be possible. Object relationships should be contained in the result. E.g. for one customer having multiple contracts each having contract items, then the query result should contain the information which contract item belongs to which contract within a single result per customer.
- Aggregations over data should be possible (although not yet used within the project).

4.6 Performance

Because the integrated view is used an application where e.g. support engineers expect immediate or at least fast answers for even complex requests while talking to a customer, the performance of the rule and query processing is a very important requirement. If responsiveness of the system is not sufficient (e.g. response in less than 2 seconds for simple retrieval, which actually is not acceptable in some cases), the whole functionality will not be accepted by its users. This means, systems like the described one can only accept rule languages that allow for efficient processing.

Not surprisingly, experience with the system has shown that efficient processing also to a great extent depends on an optimized rule execution order and caching of intermediate results. Problems that showed up here are very similar to many query optimization issues in federated database systems.
For example, the data source mapping rules we had shown above always addressed only a single table or file in a database. However, a system that implements access to external data sources only via such single-table access rules will definitely not achieve sufficient performance. The reason for this is that resulting access operations do not make use of the data source’s query capabilities like join-operations. Therefore, during query processing it is important to consider which data is stored in which system and to make use of existing indexes, uniqueness of values, or of join capabilities etc. Some of these techniques are already implemented but some are still open tasks.

The current implementation of Information Integrator answers queries all at once. Like in other data intensive applications, it would sometimes be more convenient to have a streaming or cursor result which delivers first results quick and further results on demand.

5 Summary

A data model in Information Integrator consists of ontologies. Data source models describe the structure of data that resides in external data sources. Business ontologies provide a conceptualization of business entities. FLogic rules are used to specify mappings which assemble higher-value business ontologies from other ontologies. Rules are the first choice to express semantics that is not immediately available within the data. Rules within the ontologies allow to express semantics that otherwise had to be evaluated in queries or applications.

Within our first project, the access to information in these data models is still typical data retrieval and not so much knowledge inference. Therefore, many requirements expressed here are typical requirements for querying in data intensive applications (cursor, performance, query functionality).

With an increasing number of web services where quite some of them simply expose data, we also expect the need to support data integration for such web services. Because web services expect and expose structured data, a rule language should directly support this.

The crossvision Information Integrator based on ontoprise OntoStudio™ and Ontobroker™ is the first step for Software AG in the field of semantic technologies. Recently we joined various EU research projects like NeOn (Lifecycle Support for Networked Ontologies) [NEON], “Business Register Interoperability Throughout Europe” and “SemanticGov: Services for Public Administration” [SemanticGov]. All these projects address concrete business cases. With our participation in these projects we intend to achieve deeper understanding of needs for adequate tooling and runtime systems when using semantics technologies for data integration. On the other hand we will contribute our knowledge about data-intensive processing.
Literaturverzeichnis


[NEON] http://www.neon-project.org


Der Einfluss der Datenverteilung auf die Performanz eines Data Warehouse

Thomas Legler  Wolfgang Lehner  Andrew Ross
TU Dresden  PTU NetWeaver AS TREX  SAP AG
Database Technology Group  Technische Universität Dresden  Dietmar-Hopp-Allee 16
Technische Universität Dresden  01307 Dresden, Germany  69190 Walldorf, Germany
t.legler@sap.com  lehner@inf.tu-dresden.de  a.ross@sap.com

Abstract: Dieses Papier befasst sich mit einer Studie über die Optimierungsmöglichkeiten von Anfragen auf verteilten Data Warehouse Architekturen mittels verschiedenartiger Verteilungsstrategien der beteiligten Tabellen am Beispiel SAP NetWeaver BI.

1 Einleitung

Dank wachsender Vernetzung und durch automatisierende Methoden sammeln Unternehmen mehr Daten als je zuvor. Damit haben sie die Möglichkeit, interne Abläufe und Wirtschaftsdaten genau zu analysieren. Die Daten sind jedoch im Regelfall uneinheitlich und zu granulär um persistiert zu werden, wodurch solche Informationen oft in vorverarbeiteter Form in einem Data Warehouse abgelegt werden. Eine weit verbreitete Darstellungsform eines Data Warehouse ist das Sternschema [CCS, KRT+98], eine zentral angeordnete Faktentabelle mit Belegeinträgen und Verweisen auf Dimensionstabellen, welche weiterführende Informationen enthalten. Derart komplexer Beziehungsmuster ermöglichen zahlreiche Optimierungen bei der Verarbeitung. Neben speziellen Optimierungsstrukturen und Berechnungsalgorithmen streben beispielsweise verteilte Architekturen eine gleichmäßige Auslastung ihrer Einzelkomponenten an, um die entstehende Arbeit bestmöglich zu verteilen.

Dieses Papier präsentiert eine Studie über die Auswirkungen verschiedener Datenverteilungen auf die Performanz eines Data Warehouse am Beispiel der Business Intelligence Lösung der Firma SAP - SAP NetWeaver BI. Besonders beachtet wurde dabei, wie vorhandene Metadaten neben klassischen Verteilungsstrategien für eine sinnvolle Optimierung nutzbar sind.
1.1 Sternschemas und materialisierte Sichten

Die SAP-Implementierung eines Sternschemas unterscheidet sich im Detail von der klassischen Variante. Wie in Abbildung 1 zu sehen ist, wird die zentrale Faktentabelle nicht nur von ihren Dimensionstabellen, sondern auch von X, Y und S Tabellen umgeben. Dabei speichern die X Tabellen zeitabhängige und die Y Tabellen zeitunabhängige Daten. Die Dimensionstabellen (D) speichern die Dimensionsinformationen, z.B. über Material oder Kunde, und verbinden die X/Y und S Tabellen mit der Faktentabelle. S Tabellen halten Abbildungen zwischen IDs vor.


1.2 SAP NetWeaver BI Accelerator

Als Lösung dieser Problematik hat SAP die Software SAP NetWeaver BI Accelerator entwickelt, welcher materialisierte Sichten und die sich daraus ergebenden Probleme ablöst
und die laufzeitkritischen Berechnungen ohne etwaige Vorberechnungen auf den Belegdaten ermittelt [LLR06]. Im Voraus überträgt das SAP NetWeaver BI System die zu optimierenden Sternschemata zum BI Accelerator, welcher seinerseits diese Daten in einer internen Struktur namens BIA Index ablegt.


Im September 2006 veröffentlichte die SAP-unabhängige Winter Corporation einen Bericht über den SAP NetWeaver BI Accelerator:

The BI Accelerator was effective in reducing response times for a broad range of queries, and is far more practical than the process of building tuned aggregates. The BI Accelerator scales as the problem size grows and as the blade chassis configuration is expanded with additional blades. Data loading and indexing onto the BI Accelerator occurs at impressive rates, outperforming the construction of optimized aggregates that would be required for comparable query response. [BD]

Eine derartige Konstruktion bietet einige Vorteile. Da sämtliche Daten des zu optimierenden Sternschemas nur in den BI Accelerator geladen werden, entfällt damit das, im Falle der materialisierten Sichten nötige, mehrfache Einbringen von Änderungen in eine Vielzahl von Replikaten.

Ein wichtiges Merkmal des BI Accelerators ist die Partitionierung großer Tabellen. Bisherige Ansätze zur verteilten Berechnung in einem Data Warehouse partitionieren die Faktentabelle auf die zu einem Rechnervерband gehörenden Knoten und replizieren jeweils alle weiteren Tabellen, um auch verteilt Verbundoperationen effizient berechnen zu können [BM00]. Mit diesem Verfahren können globale Anfragen auf den Einzelteilen der Faktentabelle berechnet und zum Endergebnis vereinigt werden. Für die SAP-Architektur ist diese Lösung in vielen Fällen nicht tragbar, da insbesondere Dimensionstabellen von
mehreren Sternschemas benutzt werden können und daher ein solches Replikat auf einer Vielzahl von Systemen erstellt werden müsste. Im BI Accelerator wird Redundanz vermieden und durch geschickte Verteilung der einzelnen Tabellen und eine Nutzung verteilter Verbundoperationen optimiert um effizient derartige Berechnungen ohne Replikate zu realisieren. Im BI Accelerator kann eine solche Umsortierung der Einzeltabellen automatisiert in regelmäßigen Abständen, bei Erreichen eines Schwellwertes für eine theoretisch zu erwartende Verbesserung oder manuell ausgelöst werden. Für den Nutzer oder das übergeordnete SAP-System erfolgt diese Umverteilung transparent.

2 Strategien

Im folgenden Abschnitt werden sechs Ansätze untersucht, nach denen eine Datenverteilung möglich ist. Unterschieden wird hierbei, ob die Distributionsstrategien Wissen über das zu verteilende Schema einbeziehen oder dies ohne Schemainformationen ermitteln.

Zufallsverteilung (ZV)


Round-Robin (RR)

Speicherverbrauch (SV)

CPU-Auslastung (CA)

Verbundpfadoptimiert (VO)
In manchen Anwendungsszenarien ist neben den rein technischen Informationen wie Speichergöße und CPU-Last auch Wissen über die Struktur und die Bedeutung der zu verteilenden Daten vorhanden. Eine klassische OLAP-Anfrage auf einem oder mehreren Sternschemas lässt sich laut [CM04] auf folgende Arbeitsschritte reduzieren:

1. Selektieren von Datensätzen auf der Faktentabelle
2. Gruppieren nach mindestens einer Dimension bzw. deren Attributen
3. Aggregieren der Kennzahlen innerhalb der Einzelgruppen

Selektiert werden dabei im Regelfall nur Ausschnitte der Faktentabelle, meist nur wenige Prozente der Gesamtabelle. Um diese Selektion zu erreichen, wird im allgemeinen Sternschema zuerst eine Selektion auf den gegenüber der Faktentabelle deutlich kleineren Dimensionstabellen durchgeführt und deren Ergebnis zur meist drastischen Einschränkung der betrachteten Einträge der Faktentabelle mittels Verbundoperationen genutzt. Das SAP-Sternschema hat die Dimensionstabellen noch weiter normalisiert und semantisch partitio-
niert und damit um die Dimensionstabellen noch weitere Satellitentabellen platziert. Für eine OLAP-Anfrage muss dadurch erst eine Reihe von Verbundoperationen und Selektionen ausgeführt werden. Diese betreffen jeweils nur kleine Datenmengen und können effizient bearbeitet werden.


Mit diesen Metadaten wird dem BI Accelerator ein Möglichkeit zur Verfügung gestellt, eine Gruppierung der zu verteilenden Tabellen nach den Verbundpfaden, also von der Faktentabelle über die Dimensionstabellen zu den X, Y und S Tabellen, durchzuführen. Eine Optimierung erfolgt auf eine möglichst große Lokalität für Verbundoperationen. Im Regelfall ist die Laufzeit für diese Operationen ohne Verwendung der Faktentabelle klein im Vergleich zur Gesamtlaufzeit einer Anfrage. Eine verteilte Berechnung einer solchen Operation ist meist teurer als eine lokale unverteilte Verarbeitung und sollte vermieden werden. Deutlich zeitaufwändiger ist die Selektion, Gruppierung und die danach folgende Aggregation auf den Daten der Faktentabelle. Der damit verbundene Rechenaufwand sollte auf mehrere Rechner verteilt werden, damit eine Parallelverarbeitung der Daten stattfinden kann. Die Vereinigung der Teilergebnisse im Falle einer verteilten, parallelen Aggregation ist wegen den relativ kleinen zu erwartenden Ergebnismengen meist sehr schnell und effizient, wodurch eine deutliche Steigerung gegenüber einer unpartitionierten Berechnung zu erwarten ist.

Für diese Szenarien schlagen wir folgenden Algorithmus vor:

1. Faktentabelle partitionieren und auf mehrere Rechner verteilen
2. Verbundpfade (mit Ausnahme der Faktentabelle) extrahieren
3. Mehrfach benutzte Tabellen zwischen den Verbundpfaden ermitteln
4. Alle so verbundenen Tabellen auf einem Rechner platzieren
5. Ungleichge Verteilungen der Datenmengen ausgleichen

**Dezentrale Ablage (DA)**
Um eine möglichst hohe parallele Verarbeitung zu erreichen, ist auch eine maximale Verteilung aller wahrscheinlich gleichzeitig an einer Anfrage beteiligter Tabellen denkbar. Unter den gleichen Gegebenheiten wie bei einer verbundpfadoptimierten Verteilung können die an einem Pfad beteiligten Tabellen auf möglichst viele Rechner verteilt werden, um die Selektionen auf den Einzeltabellen möglichst parallel zu verarbeiten. Nachteilig entwickelt sich dann hierbei der Aufwand zur Berechnung einer Verbundoperation, da dieser dann grundsätzlich über Rechnergrenzen hinweg erfolgen muss. Im Szenario eines SAP-Sternschemas ist das jedoch tragbar, da die an dem Verbundpfad beteiligten Tabellen oft sehr klein sind und damit wenig Daten einer Tabelle über ein Netzwerk an den jeweiligen Verbundpartner geschickt werden.

<table>
<thead>
<tr>
<th>Verfahren</th>
<th>Schemawissen</th>
<th>Umplanungen</th>
<th>Aufwand</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZV</td>
<td>nein</td>
<td>keine</td>
<td>minimal</td>
</tr>
<tr>
<td>RR</td>
<td>nein</td>
<td>bei neuen Tabellen</td>
<td>gering</td>
</tr>
<tr>
<td>SV</td>
<td>nein</td>
<td>bei neuen Daten</td>
<td>mittel</td>
</tr>
<tr>
<td>CA</td>
<td>nein</td>
<td>regelmässig</td>
<td>hoch</td>
</tr>
<tr>
<td>VO</td>
<td>ja</td>
<td>bei Schemaveränderung</td>
<td>sehr hoch</td>
</tr>
<tr>
<td>DA</td>
<td>ja</td>
<td>bei Schemaveränderung</td>
<td>hoch</td>
</tr>
</tbody>
</table>

Tabelle 1: Übersicht über Verteilungsstrategien

**3 Evaluation**

Im folgenden Abschnitt werden die vorgestellten Verteilungsverfahren ausgemessen und ihr Einfluss auf die Ausführungszeiten sowohl für sequentielle als auch für Massenanfragen dargestellt.

**3.1 Daten und Anfragen**

Da die zugrunde liegende Architektur, der BI Accelerator, derzeit nur in der Lage ist, mit SAP-Szenarios umzugehen, ist noch keine Messung an standardisierten Benchmarks wie

Das Szenario umfasst insgesamt neun Sternschemas und entsprechend neun Faktentabellen. Die Sternschemas besitzen jeweils 14-16 Dimensionstabellen unterschiedlicher Größen. Die X, Y und S Tabellen sind jeweils von mehreren Schemas und mehrfach innerhalb eines Schemas benutzt und umfassen je zwischen 500.000 und 5.000.000 Einträge für die X/Y Tabellen. Die S Tabellen sind zu ca. 85% mit unter 10.000 Einträgen belegt, 10% erreichen bis zu 1.000.000 Einträge und 5% bis zu 6.000.000 Einträge. Tabelle 2 zeigt eine Übersicht über die Größe der Faktentabellen der verwendeten Sternschemas. Der Aufbau aller Schemas ist identisch. Jede Faktentabelle besteht aus 14-16 Fremdschlüsseln auf die jeweiligen Dimensionstabellen, sowie 22 weiteren Attributen. Insgesamt sind in dem kompletten Szenario 327 verschiedene Tabellen involviert.


3.2 Messungen

Die Messung zur Bestimmung der Auswirkungen unterschiedlicher Verteilungsstrategien erfolgt jeweils für sequentiell sowie parallel ausgeführte Anfragen. Die Laufzeiten stellen die Summe der einzelnen durchschnittlichen Anfragelaufzeiten von 20 Messungen dar. Betrachtet wurden in dem Versuch zwei Versionen einer zufälligen Verteilung (ZV1, ZV2), eine Optimierung nach Speicherverbrauch (SV), eine Round-Robin Implementierung mit nach Namen sortierten Tabellen (RR), eine maximal dezentralisierte Verteilung von Verbundpartnern (DA), eine Gruppierung nach dem Verbundpfad mit und ohne Optimierung bei der Nutzung einer mehrfach benutzten Tabelle (VO1, VO2), sowie sechs Durchläufe mit wachsenden Zeitaufwand zur Bestimmung von Laufzeitstatistiken einer Tabelle (CA1-CA6). Sämtliche Daten befinden sich zur Zeit der Messung bereits im Speicher, was den
Abbildung 2: Summe aller Laufzeiten, sequentiell

Regulären Funktionsprinzipien des BI Accelerators entspricht.

Abbildung 3: Summe aller Laufzeiten, parallel

Eine einfache Optimierung nach Verbundpfaden (VO1) deutlich an Effizienz verliert, da Pfade mit mehrfach genutzten Tabellen geteilt wurden. Wenn diese Gruppenlokality berücksichtigt wird (VO2), so ergibt sich ein günstigeres Laufzeitverhalten. Die CA5-Optimierung zeigt aber, dass deutlich bessere Laufzeiten möglich sind.

Abbildung 4 repräsentiert die einzelnen Anfragelaufzeiten für die Verteilungsstrategien im sequentiellen Betrieb. Hierbei ist gut ersichtlich, dass die für eine Anfrage benötigte Zeit je nach Verfahren sehr starken Schwankungen unterliegt.

3.3 Auswertung und Ausblick


Literatur


Ruminations on Multi-Tenant Databases

Dean Jacobs, Stefan Aulbach
Technische Universität München
Institut für Informatik - Lehrstuhl III (I3)
Boltzmannstr. 3
D-85748 Garching bei München
{dean.jacobs, stefan.aulbach}@in.tum.de

Abstract: This is a position paper on multi-tenant databases. As motivation, it first describes the emerging marketplace of hosted enterprise services and the importance of using multi-tenancy to handle high traffic volumes at low cost. It then outlines the main requirements on multi-tenant databases: scale up by consolidating multiple tenants onto the same server and scale out by providing an administrative framework that manages a farm of such servers. Finally it describes three approaches to implementing multi-tenant databases and compares them based on some simple experiments. The main conclusion is that existing database vendors need to enhance their products to better support multi-tenancy.

1 Hosted Services and Multi-Tenancy

In the hosted service model [GM02a, GM02b, Wa03], a service provider develops an application and operates the system that hosts it. Customers access the application over the Internet using industry-standard web browsers or Web Services clients. As the Internet has matured, hosted services have appeared for an increasingly wide variety of enterprise applications, including ones that manage sales, marketing, support, human resources, planning, manufacturing, inventory, financials, purchasing, and compliance [Th06]. While hosted services are attractive to all segments of the market, they are particularly appealing to small- to medium-sized businesses, which often lack the resources to maintain a complex data center. Hosted services are exploiting such “greenfield” opportunities to expand the overall size of the market.

In comparison to traditional on-premises solutions, hosted services can reduce the total cost of ownership of an application by aggregating customers together and leveraging economy of scale. This principle applies to both capital expenditures, e.g., for hardware and software, and operational expenditures, e.g., for bandwidth and personnel. Because a hosted service is focused on one application, the infrastructure and the procedures for managing it can be highly optimized: well-known examples here include the Google File System [GGL03] and Hotmail [Smn05]. Such optimizations are essential to support large numbers of small- to medium-sized businesses, which would otherwise be prohibitively expensive.
Multi-tenancy is an optimization for hosted services in which multiple customers are consolidated onto the same operational system, a technique pioneered by salesforce.com [Sa06, Co06]. Multi-tenancy allows pooling of resources, which improves utilization by eliminating the need to provision each customer for their maximum load. Multi-tenancy can also improve management efficiencies by providing a uniform framework for administering the system. A multi-tenant system should support both scale up and scale out [De99]: scale up by consolidating multiple customers onto the same server and scale out by having the administrative framework span a farm of such servers. Scale out is required because it is not cost effective to scale up a single server indefinitely.

The administrative framework of a multi-tenant system should support the ability to migrate a customer from one server to another within the farm. For example, a customer might start out on a server that manages trial accounts, be moved to a server that manages small production accounts, and grow until it is moved to a dedicated server that manages only one account. Migration should also be possible between farms, for example, to allow customers to be moved from one data center to another. The administrative framework should also support rolling upgrade, where the servers in a farm are upgraded to a new version of the application one at a time. Since it is difficult to generate realistic Internet-scale workloads for testing, rolling upgrades are needed to gain experience with new versions before they are put widely into production.

2 Requirements on Multi-Tenant Databases

Multi-tenancy can be applied at the database layer of a hosted service, where it can be very effective due to the high cost of provisioning and operating databases. This section argues that, in addition to managing customers’ private data, a multi-tenant database should manage customer metadata and shared public data.

The administrative framework of a multi-tenant database should maintain metadata about customers, such as their contact information, their location in the farm, and the features they are allowed to access. Certain administrative operations will need to access this metadata alongside of customer data. For example, a service provider might want to scan the metadata to find all customers in a given region and then determine which of those customers have more than a certain amount of data. To support such operations, the administrative framework should offer a unified query language that obviates the need for a general-purpose programming language with embedded queries. In addition to being easier to use, this approach makes it simpler to execute administrative operations in bulk on individual databases in the farm.
The target application for a hosted service will generally have a base schema that specifies all of its standard data. A multi-tenant database should maintain an instance of this base schema for each customer. The unified query language should ensure that DDL statements for modifying the base schema and DML statements for transforming existing data within it are applied to all customers in the farm, within the context of a rolling upgrade. The ability to perform such operations in bulk on the individual databases is essential to minimize downtime during an upgrade. Many business applications, such as CRM or ERP, allow customers to extend the base schema, e.g., by adding new columns to existing tables and adding new tables. The administrative framework should maintain the specification of each customer’s schema extensions as part of their metadata.

A service provider may want to maintain public information, such as area code data, census data, streaming stock prices, or industry best practices, that is accessible to all customers. Such shared data will generally be read-only, although there are cases where space-saving technologies like copy-on-write [PB03] should be used to allow customers to privately update the data. A multi-tenant database should allow customers to seamlessly access shared data alongside of their private data. This will entail either replicating the shared data on each server in the farm or providing a mechanism to access it remotely.

3 Implementing Multi-Tenant Databases

This section describes and compares three approaches to implementing multi-tenant databases: shared machine, shared process, and shared table. These approaches are increasingly better at pooling resources and executing administrative operations in bulk. However they increasingly break down the isolation between customers, weakening security and increasing contention for resources.

The discussion presents the results of some preliminary experiments on the memory and disk usage of five databases: PostgreSQL [Po06], MaxDB [Ma06], and three commercial databases cleverly code-named Commercial1, Commercial2, and Commercial3. Unless otherwise indicated, no specialized tuning was performed on these databases. The experiments used a simplified version of the CRM schema offered by salesforce.com containing eleven tables, each with about twelve fields.

The experiments were focused on the shared process approach: they created different numbers of schema instances, each with eleven empty tables, and loaded them into memory in the same database process. The experiments are preliminary in that the tables were not populated and no queries were executed against them. The results are shown in Table 1. In all cases, the storage requirements were close to being linear in the number of schema instances, hence only the end points at 1 and 10,000 instances are presented. PostgreSQL and Commercial1 produced various errors that prevented the experiments from completing and the final numbers are extrapolated. These errors were generally fixed limits on internal structures, such as the number of objects in a table space. All numbers are in megabytes.
Table 1. Storage Requirements for Schemas Instances (in megabytes)

<table>
<thead>
<tr>
<th></th>
<th>Memory 1 instance</th>
<th>Memory 10,000 instances</th>
<th>Disk 1 instance</th>
<th>Disk 10,000 instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>55</td>
<td>79</td>
<td>4</td>
<td>4,488</td>
</tr>
<tr>
<td>MaxDB</td>
<td>80</td>
<td>80</td>
<td>3</td>
<td>1,168</td>
</tr>
<tr>
<td>Commercial1</td>
<td>171</td>
<td>616</td>
<td>200</td>
<td>414,210</td>
</tr>
<tr>
<td>Commercial2</td>
<td>74</td>
<td>2,061</td>
<td>3</td>
<td>693</td>
</tr>
<tr>
<td>Commercial3</td>
<td>273</td>
<td>359</td>
<td>1</td>
<td>13,630</td>
</tr>
</tbody>
</table>

3.1 Shared Machine

In the shared machine approach, each customer gets their own database process and multiple customers share the same machine. An example of this approach is CasJobs, which supports analysis of data in the Sloan Digital Sky Survey database [Mu05]. This approach is popular in practice because it does not require modifying the implementation of the database. In addition, it does not substantially reduce customer isolation, particularly if each process is placed in its own virtual machine.

The main limitation of this approach is that it does not pool memory. The first column of Table 1 shows that the memory requirements to handle one schema instance vary from 55 MB to 273 MB across the five databases. This result indicates that this approach cannot scale beyond tens of active customers per server. Another limitation of this approach is that each database requires its own connection pool on each application server, so sockets will not be shared among customers. To improve this situation, the implementation could be modified to share memory and sockets among co-located database processes, possibly at a level below those processes.

Because isolation between customers is strong in this approach, executing administrative operations in bulk is not feasible: each database will execute queries on its own. In addition, providing seamless access to shared data requires a mechanism to access it remotely, otherwise it will end up being replicated multiple times on the same server. Customer migration is straightforward however, in that it entails simply moving files from one server to another.

An interesting issue is brought to light by the third column of Table 1, which shows the disk requirements to handle one schema instance. Commercial1 pre-allocates 200 MB for the database, although this could be reduced by additional tuning. Under this approach, this practice could result in terabytes of wasted disk space when multiplied across thousands of small businesses.
3.2 Shared Process

In the shared process approach, each customer gets their own tables and multiple customers share the same database process. For most databases, it does not matter whether each customer gets their own schema, since schemas are implemented using a lightweight prefixing mechanism. However, it is useful if each customer gets their own physical table space so that customer migration entails simply moving files from one server to another. In addition, it allows the system administrator to balance the I/O load by distributing customers across different backing disks.

This approach is considerably better at pooling memory. The second column of Table 1 shows that the memory requirements to handle 10,000 schema instances vary from 79 MB to 2,061 MB across the five databases. MaxDB stands out in that the memory requirements were the same regardless of the number of schema instances, suggesting that the data dictionary is being paged. Further experiments are required to determine the performance implications of this technique. In any case, this result indicates that this approach should easily scale up to thousands of active customers per server, a two orders of magnitude improvement over the shared machine approach. Further improvements could be provided by modifying the implementation to keep only one copy of the schema and having each instance refer to it. This technique could also be applied to the disk, although the third and fourth columns of Table 1 indicate that it would save only gigabytes of storage.

Since there is only one database in this approach, customers can share connection pools. There is a well-known downside to connection pooling however: all connections must be associated with a fixed principal who can access everything. This means that both security and the management of resource contention have to be handled at the application layer. Thus errors in the application code could allow one customer to access another customer’s tables or prevent them from getting their fair share of resources. This problem could be mitigated by allowing the principal associated with a database connection to be picked up from the application server. Support for this feature is beginning to appear in databases today.

To execute administrative operations in bulk in this approach, the metadata embodied in the schemas has to be made available as data and it must be possible to create operations that are parameterized over the domain of table names. Some support for these features is available in databases today. These capabilities are also required in the administrative framework to handle queries that go across the individual databases in the farm.

In this approach, administrative operations such as adding customers, removing customers, and extending the base schema entail executing DDL statements. This requirement can be problematic for some databases, which behave poorly if schemas are modified while the system is in operation.
3.3 Shared Table

In the shared table approach, data from many customers is stored in the same tables, as illustrated in Figure 1. A TenantId column is added to each table to identify the owner of each row. Every application query is expected to specify a single value for this column. To allow customers to extend the base schema, each table is given a fixed set of additional generic columns. These columns might be of type VARCHAR, as shown in Figure 1, or they might have a mix of types. The data for the \( n \)-th new column of a table for each customer is placed in the \( n \)-th generic column of the appropriate type, after performing any necessary type conversions.

<table>
<thead>
<tr>
<th>TenantId</th>
<th>Account</th>
<th>Name</th>
<th>…</th>
<th>Val0</th>
<th>…</th>
<th>Val100</th>
</tr>
</thead>
<tbody>
<tr>
<td>1041</td>
<td>0021</td>
<td>Acme</td>
<td></td>
<td>1/3/95</td>
<td></td>
<td>------</td>
</tr>
<tr>
<td>1041</td>
<td>0029</td>
<td>Ball</td>
<td></td>
<td>3/7/72</td>
<td></td>
<td>------</td>
</tr>
<tr>
<td>1053</td>
<td>0016</td>
<td>Gump</td>
<td></td>
<td>red</td>
<td></td>
<td>------</td>
</tr>
<tr>
<td>1053</td>
<td>0049</td>
<td>Wonk</td>
<td></td>
<td>blue</td>
<td></td>
<td>------</td>
</tr>
</tbody>
</table>

Figure 1. Account Table in the Shared Table Approach

This approach is clearly the best at pooling resources. Its ability to scale up is limited only by the number of rows the database can hold, which should offer several orders of magnitude improvement over the shared process approach. Administrative operations can be executed in bulk simply by executing queries that range over the TenantId column. And since there is only one database, customers can share connection pools.

There are however several significant problems with this approach. First, since files on the disk have intermingled data from multiple customers, migration requires executing queries against the operational system. Second, such intermingling can impact the performance of accessing a customer’s data, since it may be spread out across many pages. Third, security can be pushed down into the database only if different access privileges can be assigned to different rows in the same table. Fourth, the use of generic columns is feasible only if the database has a compact representation for sparse tables. Fifth, if typing of the generic columns has been abandoned, it will be hard to use column-oriented features such as indexes and integrity constraints.

The sixth and biggest problem with this approach is that queries intended for a single customer have to contend with data from all customers, which compromises query optimization. In particular, optimization statistics aggregate across all customers and table scans go across all customers. Moreover, if one customer requires an index on a column, then all customers have to have that index. To improve this situation, the implementation could be modified to treat the TenantId column as if it defined separate tables. Alternatively, this problem could be handled above the database in a query preprocessor that performed its own optimizations.
4 Conclusions

This paper has argued that multi-tenant databases are essential for hosted services to manage high traffic volumes at low cost. Multi-tenant databases may also be useful behind the firewall of a single organization, e.g., to handle multiple branches that have the same schema.

This paper described and compared three approaches to implementing multi-tenant databases. The authors’ opinion is that, among these, shared-process is the most promising approach. Shared-machine will continue to be used in circumstances where security concerns are paramount. Modifications to increase resource pooling could provide moderate improvements in scalability, however they may not be worth the effort of performing delicate inter-process coordination. Shared-table swings too far in the other direction, potentially compromising performance, customer migration, security, and typing. Shared-process can be made more scalable by maintaining only one copy of the base schema, and more secure by having the principal associated with a database connection be picked up from the application server. In any case, it is clear that existing database vendors need to enhance their products to better support multi-tenancy.

It is interesting to note that support for multi-tenancy generally requires blurring the line between data and metadata: shared-process offers bulk execution of administrative queries by allowing them to be parameterized over the domain of table names, while shared-table repairs query processing by having the TenantId column implicitly define tables. Since these features require only limited support for dynamic manipulation of schemas, it should be possible to support them without compromising query optimization.

Acknowledgements This paper benefited greatly from discussions with Alfons Kemper, Mike Carey, Jim Gray, Adam Messinger, Anno Langen, and the people who started it all, the developers at salesforce.com.

References


1 Einleitung

In dieser Arbeit werden grundsätzliche Architekturfragen sowie die Benutzerschnittstelle eines mobilen CAFM-Systems beschrieben, das im Rahmen einer Diplomarbeit als Erweiterung des seit langem im Einsatz befindlichen CAFM-Systems consultware® realisiert wurde.

2. Das CAFM – System consultware®

consultware® ist ein CAFM-System, das vom Lübecker Unternehmen PietschCONSULT GmbH entwickelt wurde. consultware® wird unter anderem am Universitätsklinikum Schleswig-Holstein eingesetzt.

consultware® beinhaltet die Funktionsbereiche Fläche, Reinigung, Schlüssel, Miete, Technik, Gefahrstoff und Brandschutz.


Reinigung stellt eine moderne Umsetzung des Gesamtprozesses Unterhaltsreinigung mit all seinen Facetten dar. Integrierte Prozesse wie die ergebnisorientierte Qualitätssicherung, die taggenaue Abrechnung von Normal- und Sonderleistungen, die Revierplanung oder das Auftrags- und Reklamationswesen bieten in der Praxis erprobte und funktionelle Lösungen.

Schlüssel realisiert die Verwaltung von Zutrittsberechtigungen auf grafischer Basis. Hierzu werden alle benötigten Daten wie Räume, Türen, Personen, Schlüssel, Zylinder und Lesegeräte in einem geschlossenen System geführt. Es werden die an Personen ausgegebenen Schlüssel verwaltet, so dass sich auf einen Blick feststellen lässt, wer welchen Schlüssel besitzt und welche Räume durch welche Personen geöffnet werden können.

Miete behandelt die Kosten für die Nutzung von Räumen. Der Anwender wird unterstützt, den Einsatz vorhandener Flächen zu optimieren und so die Wirtschaftlichkeit zu verbessern. Leerstehende und ungenutzte Räume können abgefragt und bei Bedarf schnell anderen Nutzern zur Verfügung gestellt werden.

Technik ist speziell auf die Anforderungen der Betriebstechnik zugeschnitten. Technische Informationen zu Gebäuden und Räumen, zum Inventar und zu Anlagen werden verwaltet. Wartungen, Reparaturen und Aufträge lassen sich effizient planen und abwickeln und das integrierte Störmanagement hilft bei der Verbesserung der Servicequalität. Die Struktur der zu verwaltenden Daten kann selbst bestimmt werden, so dass ein auf spezielle Anforderungen zugeschnittenes System entsteht.


3 Grundsätzliche Architekturfragen für ein mobiles CAFM-System

In diesem Abschnitt sollen einige grundsätzliche Fragestellungen in Bezug auf mobiles CAFM angesprochen werden. Außerdem wird erläutert, welche Vorteile ein mobiles CAFM prinzipiell bieten kann.

3.1 Warum mobiles CAFM?

Stationäre Systeme sind normalerweise nicht vor Ort direkt im verwalteten Gebäude nutzbar, weil dort in der Regel kein Zugriff auf das CAFM-System besteht. Ein Zugriff über das WWW scheidet meistens aus, da die Kunden, die das Gebäude nutzen, die Verwendung ihrer PCs zu derartigen Zwecken meist nicht gestatten. Um CAFM wirklich mobil im verwalteten Gebäude nutzen zu können, ist daher ein System notwendig, welches den Zugriff zu den Daten über mobile Geräte wie PDAs gestattet.


Die folgenden beiden Unterabschnitte sollen 2 grundsätzliche Alternativen für die Architektur eines mobilen CAFM-Systems betrachten. Diese betreffen die Frage der Aufteilung zwischen mobilem Gerät (Client) und dem zentralen Teil des CAFM-Systems (Server).

3.2 Thin-Client

Das Konzept der „Thin-Client-Architektur“ bedeutet, dass der Client seine Daten möglichst vollständig und bei jedem Einsatz neu von einem Server bezieht. Die funktionale Ausstattung der Client-Anwendung beschränkt sich im Wesentlichen auf die Ein- und Ausgabe, wobei die Einsatzumgebung auf dem Client durch einen konventionellen Browser gegeben ist.

Der große Vorteil einer Thin-Client-Architektur ist, dass eine Veränderung der Daten auf dem Server bei allen Clients direkt sichtbar wird. Insofern ist ein solches System

3.3 Fat-Client

Das Konzept des „Fat Client“, auch „Rich-Client“ genannt, bedeutet, dass die eigentliche Verarbeitung der Daten vor Ort auf dem Client vollzogen wird. Dieser stellt dafür eine grafische Oberfläche zur Verfügung. Die wesentlichen Vorteile sind:


Folgende Tabelle zeigt eine Gegenüberstellung der beiden Ansätze:

<table>
<thead>
<tr>
<th>Vorteile</th>
<th>Nachteile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thin-Client</td>
<td>- Plattformunabhängig (es wird lediglich ein Browser benötigt)</td>
</tr>
<tr>
<td></td>
<td>- keine Update-Probleme</td>
</tr>
<tr>
<td></td>
<td>- wenig Ressourcen auf dem Client notwendig</td>
</tr>
<tr>
<td>Fat-Client</td>
<td>- Gestaltung der Benutzeroberfläche durch HTML eingeschränkt</td>
</tr>
<tr>
<td></td>
<td>- ständige Verbindung erforderlich</td>
</tr>
<tr>
<td></td>
<td>- Umfangreiche Benutzeroberfläche</td>
</tr>
<tr>
<td></td>
<td>- Offline-Verwendung</td>
</tr>
<tr>
<td></td>
<td>- lokale Behandlung der Interaktionen</td>
</tr>
</tbody>
</table>

- evtl. Update-Probleme |
- Synchronisation erforderlich |
- Transaktion erforderlich

525
Das Einsatzgebiet spielt eine entscheidende Rolle bei der Wahl der Architektur. Eine Architektur, die eine ständige Verbindung zum Server erfordert, ist in vielen Einsatzgebieten nicht realisierbar, einerseits wegen der Kosten, andererseits weil kein Mobilfunknetz oder WLAN-Netz zur Verfügung steht.

3.4 Auswahl der Rauminformationen


4 Ein mobiles CAFM-System

In diesem Abschnitt wird ein mobiles CAFM-System beschrieben, welches auf dem System consultware® aufsetzt. Um ein möglichst universell einsetzbares mobiles CAFM-System zu realisieren, wurde für das hier beschriebene System eine Fat-Client-Lösung realisiert, d.h. der Client (z.B. ein PDA) realisiert die mobil benötigten Funktionen selbstständig und kann offline ohne Verbindung zum Server betrieben werden. Zu diesem Zweck ist es möglich, die für eine konkrete Verwendung (ein Gebäude oder eine Gruppe von Gebäuden) relevanten Daten auf ein PDA herunterzuladen und mit diesen Daten zu arbeiten und sie auch im PDA zu verändern. Die veränderten Daten werden später auf den Server heraufgeladen, sobald eine Verbindung existiert. Hierbei ergibt sich das Problem der Datenkonsistenz, wenn mehrere Clients gleichzeitig mit den gleichen Daten arbeiten.

4.1 Benutzerschnittstelle

Zur Realisierung der Benutzerschnittstelle wurde ein handelsübliches PDA verwendet. Die Anschaffungskosten inklusive des Scanaufsatzes betragen derzeit weniger als 300 €.

4.1.1 Arbeitsweise des mobilen Gerätes

Die Arbeitsweise des mobilen Gerätes wird in Abb. 1 demonstriert.

Abbildung 1: Arbeitsweise des mobilen Gerätes


4.1.2 Arbeitsweise des Systemclients zur Datensynchronisation

Zur Sicherstellung der Konsistenz müssen die Daten der mobilen Geräte mit dem Hauptsystem synchronisiert werden. Die Synchronisation der Daten wird über einen separaten Systemdialog gemäß Abb. 2 durchgeführt.
Abbildung 2: Export auf den PDA und Import vom PDA (Synchronisation)
4.1.3 Anwendungsbeispiele


4.2 Implementierung


Für die Darstellung von grafischen Komponenten kommt die Abstract Window Toolkit (AWT)-Bibliothek von Java zum Einsatz. Es gibt zwar vereinzelt Java Virtual Machines für mobile Geräte, die Swing unterstützen, was jedoch eher die Ausnahme als die Regel ist.

Eine erste Prototypversion des mobilen CAFM wurde im Rahmen einer Diplomarbeit realisiert, die bei der Firma PietschCONSULT GmbH erstellt wurde [Ga06].

5. Zusammenfassung und Ausblick


werden kann. Dies ist der wesentliche Unterschied zu der „Java Cryptography Extension“ (JCE), welche seit der Version 1.4 im SDK von Sun enthalten ist [Zi03].


**Literaturverzeichnis**

[Bo] The Legion of the Bouncy Castle, siehe http://www.bouncycastle.org/
Überlegungen zur Entwicklung komplexer Grid-Anwendungen mit Globus Toolkit

Andreas Walter¹, Klemens Böhm², Stephan Schosser²

¹FZI Forschungszentrum Informatik, IPE, Haid-und-Neu-Straße 10-14, 76131 Karlsruhe
awalter@fzi.de

²Universität Karlsruhe (TH), IPD, Am Fasanengarten 5, 76131 Karlsruhe
{boehm, schosser}@ipd.uka.de


1 Einleitung


¹ Globus Toolkit; http://www.globus.org


2 Globus Toolkit


2.1 Anforderungen an Grid-Middleware

Globus Toolkit soll Teilnehmern einer Grid-Anwendung das sichere Verfügbarmachen von Ressourcen wie zum Beispiel Rechenleistung, Datenbanken und Speicherplätzen über das Internet ermöglichen. Die Hauptanforderung der Teilnehmer ist dabei, dass die lokale Kontrolle über die zur Verfügung gestellten Ressourcen nicht verloren geht. Foster et al. [FK02, FK01] haben diese Anforderungen und ihre Umsetzung durch Standardkomponenten in Globus Toolkit wie folgt definiert:
Sicherheitsanforderungen: Die Sicherheitsanforderungen umfassen die Authentifizierung und Autorisierung von zugangsberechtigten Nutzern eines Grids sowie die Verschlüsselung sämtlicher über das Internet übertragenen Daten. Globus Toolkit stellt hierfür ein Sicherheitscenter zur Erstellung von SSL-Zertifikaten für berechtigte Teilnehmer bereit [Th99].


Neben der Sicherheitsinfrastruktur sind Grid-Services „das wichtigste Konzept in Globus Toolkit“ [Fo05].

2.2 Rolle und Chancen von Grid-Middleware in der Industrie

In diesem Abschnitt gehen wir kurz auf die aktuellen und zukünftigen Möglichkeiten von Grid-Middleware in der Industrie ein. Das Ausnutzen von Rechenleistung auf Rechnern freiwilliger Teilnehmer ermöglicht umfangreiche kostengünstige Berechnungen [GS02]. Hierzu können Firmen ihre vorhandenen Systeme als Grid zusammenschließen und gemeinsam umfangreiche Berechnungen durchführen. So genannte „Access Grids“,

3 Verwandte Arbeiten

Hier stellen wir zuerst Erweiterungen der Globus Toolkit Plattform vor, die bei der Entwicklung von komplexen Grid-Anwendungen hilfreich sein können, und anschließend einige interessante Grid-Anwendungen, die auf Globus Toolkit basieren.

3.1 Erweiterungen für Globus Toolkit


Die Standardkomponenten für die Steuerung von Ressourcen in Globus Toolkit sind nicht besonders flexibel. Sie sind nicht geeignet zur Verwaltung von ganzen Clustern von Computersystemen. Ferner ermöglichen sie nicht die Definition von komplexen Zuteilungsregeln für Ressourcen [Fo05].


---

2 http://dmrl.latech.edu/Access-Grid/
3 Sorma-Projekt - www.sorma-project.org
4 Condor – High Throughput Computing; http://www.cs.wise.edu/condor
Die Middleware OGSA DAI\textsuperscript{6} ermöglicht die Integration von heterogenen Datenbanksystemen in ein Grid. Diese kann man, beispielsweise zur Speicherung von Daten, die Web-Crawler liefern, einheitlich in ein Grid integrieren.

3.2 Grid-Anwendungen basierend auf Globus Toolkit


4 Referenzarchitektur eines verteilten Web Crawlers


4.1 Referenz-Architektur für verteilte Web-Crawler

Abbildung 1 zeigt die Referenzarchitektur für einen verteilten Web-Crawler, die Shkapenyuk und Suel vorgeschlagen haben [SS02]. Web-Crawler kontaktieren Web-Server im Internet und laden die dort vorhandenen Web-Seiten. Anschließend übertragen die Web-Crawler diese Web-Seiten zur Speicherung an Speicherplätze.

\textsuperscript{6} The OGSA-DAI Project, http://www.ogsadai.org.uk

Shkapenyuk und Suel haben diese Referenzarchitektur innerhalb eines lokalen Netzwerks installiert. Während der Evaluation sahen sie als Engpässe hauptsächlich die Ressource Bandbreite, die die acht eingesetzten Web-Crawler für den Download von Web-Seiten benötigen, und die Ressource Speicherplatz, die die Speicherplätze für das Ablegen der Web-Seiten benötigen. Das Kontrollsystem benötigte hauptsächlich Rechenleistung zur Extraktion neuer Adressen sowie Hauptspeicher, um über bereits bearbeitete Web-Seiten Buch führen zu können.

4.2 Umsetzung der Referenzarchitektur mit Globus Toolkit

trollsystem immer die aktuell vorhandene Menge an Web-Crawlern und Speicherplätzen im Grid ermitteln.


4.3 Erweiterungen für einen hochgradig verteilt arbeitenden Web-Crawler


Die Knoten eines verteilten Kontrollsystems müssen untereinander kommunizieren, um zu ermitteln, welche Web-Seiten andere Knoten bereits bearbeitet haben. Zusätzlich wird noch ein Mechanismus benötigt, der hinzukommende Crawler und Speicher gleichmäßig auf die Knoten des verteilten Kontrollsystems aufteilt, um Fairness zu gewährleisten.


538

5 Evaluation von Globus Toolkit


Zur Evaluation dieser Annahme haben wir einen Web-Crawler erstellt und innerhalb der Globus Toolkit Plattform installiert. Dieser Web-Crawler bearbeitet parallel acht ver-

\(^7\) http://boinc.berkeley.edu
\(^8\) http://www.unicore.com
schiedene Jobs, indem er die entsprechenden Web-Seiten lädt, neue Webadressen extrahiert und die Web-Seite anschließend auf der Festplatte speichert.


6 Zusammenfassung und Ausblick


7 Literaturverzeichnis


⁹ www.apache.org
Data Staging for OLAP- and OLTP-Applications on RFID Data

Stefan Krompaß, Stefan Aulbach, Alfons Kemper

Technische Universität München – Lehrstuhl für Informatik III
85748 Garching bei München
(krompass|aulbach|kemper)@in.tum.de

Abstract:
The emerging trend towards seamless monitoring of all business processes via comprehensive sensor networks – in particular RFID readers – creates new data management challenges. In addition to handling the huge volume of data generated by these sensor networks the information systems must support the efficient querying and analysis of both recent data and historic sensor readings. In this paper, we devise and evaluate a data staging architecture that consists of a distributed caching layer and a data warehouse. The caches maintain the most recent RFID events (i.e., sensor readings) to facilitate very efficient OLTP processing. Aged RFID events are removed from the caches and propagated into the data warehouse where related events are aggregated to reduce storage space consumption. The data warehouse can be utilized for business intelligence applications that, e.g., analyze the supply chain quality.

1 Introduction

Radio frequency identification (RFID) is expected to become the key technology for monitoring object movement by storing product information on tags which are attached to individual items. RFID sensors (or RFID readers) can read the identification of an item without requiring either contact or line of sight. Thus, with sensors placed at various locations, the movement of items can be tracked. For this purpose the electronic product code (EPC, [EPC06]), an identification scheme for the real-time, automatic identification of objects, has been standardized by EPCGlobal [Hei05].

From a database perspective, the challenge is to manage the data to enable comprehensive control and analysis. There are two types of applications which are important for the analysis: On the one hand, lightweight queries are processed on most recent data, i.e., information associated with the current location and the last time an item was scanned. These queries are submitted frequently, e.g., by an object tracking service where customers can query the current location of their order or by an RFID-based process control ([TDF06]). On the other hand, OLAP queries allow business intelligence applications to analyze, for example, the quality of supply chains. The OLAP queries on the warehouse are less frequently issued in comparison to the lightweight queries, which are run against the caches, but process a much larger data volume.

In this paper, we devise and evaluate a data staging architecture that consists of a distributed caching layer and a data warehouse. The caches constitute main-memory databases which maintain the most recent RFID events to facilitate very efficient query processing.
Aged RFID events are removed from the caches and are propagated into the data warehouse where related events are aggregated to reduce storage space consumption. Currency of data ([CKRS04]) is an integral part of our novel architecture. In contrast to traditional warehouses, where it is acceptable to have updates only infrequently and at predictable times, our data staging approach updates the warehouse on the fly. For building such an RFID infrastructure, the contributions of this paper can be summarized as follows.

- **Database design.** Our proposed database schema efficiently precomputes and stores the path information of the objects in the warehouse.
- **Data cleaning.** In our architecture, a cache maintains the most recent data about an item. But not all events are necessary for maintaining the aggregated data in the warehouse. The cache filters these events and propagates only information relevant for business intelligence applications to the warehouse.
- **Data staging.** We describe two novel data staging mechanisms, *tuple-wise data staging* and *bulk data staging* to update the aggregated data in the warehouse. The data staging process is only triggered when tracked objects move to another location. While the *tuple-wise* approach updates the warehouse on a state change for individual items, the *bulk* approach processes state changes for groups of items.
- **Validation.** We present analytical estimates and benchmarks which show that traditional database schemas are not sufficient for tracking moving objects. Furthermore, we demonstrate the impact of the data cleaning and data staging mechanisms on the performance of RFID event processing.

The rest of the paper is organized as follows. Section 2 gives an overview of our RFID architecture. In Section 3, we explain how RFID events are processed. We give a description of the data staging and explain our two approaches for data staging: *tuple-wise data staging* and *bulk data staging*. Benchmarks for the quantitative evaluation of our architecture are presented in Section 4, followed by an overview of related work in Section 5. In Section 6, we summarize our studies and outline ongoing and future work on this subject.

## 2 Architecture

This section gives an overview of the components and how they are connected to each other before the design details of the cache and the warehouse are presented.

### 2.1 System Design

The Item Tracking System is implemented as a 3-tier-architecture, consisting of data stores at the lowest level, middleware atop of it and finally a set of clients, which can be either sensors or query clients. Figure 1 shows a comprehensive overview.

**Data Stores** The data stores are divided in two main storage areas, depending on the kind of data they store. The different data stores reflect the different types of queries a client can execute. Transactional processing takes place within the cache, while analytic processing is done within the warehouse.
The first data store type is the cache layer, which stores the most recent data inside a main memory relational database system used as cache. After receiving data from a sensor, the cache is updated to reflect the current item state. The middleware keeps track of the changes to the warehouse and ensures that no information necessary for constructing the warehouse data is overwritten.

This paper illustrates the usage of just one cache for the most recent data, but ongoing work is focused on scaling the system to multiple caches. Multiple caches are combined into a cache group, which in fact constitutes a partitioned database. Every cache group contains all items that are known to the system. In order to facilitate different query types, multiple cache groups can be set up, but just one cache group is necessary for data staging. In such a case, the information about item locations is stored redundantly.

The partitioning is based on different strategies. For example, a single RFID sensor with a very high scan rate (e.g., at the entrance of a warehouse) may have its own cache, while the other sensors share the remaining caches. We denote this partitioning strategy as reader-based. The other type of strategy is item-based, where the partitioning is based on product groups, supplier or other item-specific attributes. Combining multiple strategies leads to redundant data caching.

The second data store type is the data warehouse. The warehouse is based on a conventional relational database management system and contains historic data needed to analyze the sensor readings, e.g., for analyzing the supply chain over a longer period of time. Section 3 describes the data staging algorithms for propagating recent cache data into the warehouse.

**Middleware** The middleware is responsible for data cleaning, to compress and optimize the raw RFID data streams generated by the sensors. Furthermore, the middleware triggers the data staging. Placing this functionality into the middleware enables to intercept cache update operations that may lead to overwritten data, and initiate a data staging first.

Different middleware plug-ins support a variety of interfaces to sensors, clients and even data staging algorithms. Currently, two data staging algorithms are implemented that will be compared in the course of this paper. As part of the data staging procedure, the middle-
Sensors and Query Clients Two types of clients use the interfaces provided by the middleware. On the one hand, data sensors acquire data and send it to the middleware via socket connections. Each sensor produces RFID events \((\text{epc}, \text{reader}, t)\), where \text{epc} represents a scanned item, \text{reader} is a unique identifier of the reader and \(t\) is the time when the item passed the sensor area. The interface to the sensors can be extended in future, e.g., to support environmental sensors in refrigerated stores, even integrated in RFID tags ([OTS+06]). The socket-based connection allows the exploitation of modern, high performance network technologies (e.g. the Socket Direct Protocol (SDP) via InfiniBand) in order to decrease the protocol overhead.

On the other hand, there are clients querying the data stores. OLTP queries that rely on the most recent data are processed by the cache, while OLAP queries on historic data are processed by the warehouse. The middleware acts as a proxy for the clients so that incoming queries via JDBC are redirected to the appropriate data store(s).

### 2.2 Cache Design

The first part of the database schema consists of the table \text{RFID_RAW} that is maintained in the cache. This relation, shown in Table 1, contains the most recent RFID entries – cleaned and processed by the middleware. As part of the data cleaning, a new entry is inserted into \text{RFID_RAW} if a yet unknown item has been scanned. Otherwise, the existing entry of that particular scanned item is updated. The detailed description of the procedure of inserting events into the cache is part of Section 3. An example set of such entries is given in Table 2.

### 2.3 Data Warehouse Design

Every item travels along a path, consisting of the RFID readers the item passes by. For each reader on the path, the warehouse stores the timestamps when an item arrives at this reader and leaves the reader, respectively. Since the \text{RFID_RAW} relation inside the cache is independent of the contents of the warehouse, different approaches for representing this path information can be used.

---

**Table 1: Column Description of Cache Table**

<table>
<thead>
<tr>
<th>item</th>
<th>reader</th>
</tr>
</thead>
<tbody>
<tr>
<td>The EPC of the item</td>
<td>The EPC of the reader that scanned the item</td>
</tr>
<tr>
<td>t_in</td>
<td>timestamp</td>
</tr>
<tr>
<td>The time when the item arrived at the current path</td>
<td>The last timestamp when the item was scanned by the reader</td>
</tr>
<tr>
<td>subpath_out</td>
<td>The time when the item was last scanned by the previous reader</td>
</tr>
</tbody>
</table>

**Table 2: Example of Cache Contents After Scanning at Timestamp \(t_6\)**

<table>
<thead>
<tr>
<th>item</th>
<th>reader</th>
<th>t_in</th>
<th>timestamp</th>
<th>subpath_out</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i_1)</td>
<td>(s_3)</td>
<td>(t_6)</td>
<td>(t_6)</td>
<td>null</td>
</tr>
<tr>
<td>(i_2)</td>
<td>(s_3)</td>
<td>(t_6)</td>
<td>(t_6)</td>
<td>null</td>
</tr>
<tr>
<td>(i_3)</td>
<td>(s_4)</td>
<td>(t_5)</td>
<td>(t_6)</td>
<td>null</td>
</tr>
<tr>
<td>(i_4)</td>
<td>(s_4)</td>
<td>(t_5)</td>
<td>(t_6)</td>
<td>null</td>
</tr>
<tr>
<td>(i_5)</td>
<td>(s_5)</td>
<td>(t_5)</td>
<td>(t_6)</td>
<td>null</td>
</tr>
<tr>
<td>(i_6)</td>
<td>(s_5)</td>
<td>(t_5)</td>
<td>(t_6)</td>
<td>null</td>
</tr>
</tbody>
</table>
2.3.1 Single Event Approach: No Path Extensions

With the single event approach, every location change of an item is stored in a single row in the table RFID_READ. An entry in this table contains the times when the item reaches and leaves the reader in columns \( t_{\text{in}} \) and \( t_{\text{out}} \). Figure 2(a) shows the schema of the single event approach. Table 3 contains some example entries 1.

During data staging no additional processing has to be done. The only task is to insert new rows into the table RFID_READ, so there is no need for any space or time consuming transformation, and the data staging run is quite fast. However, the approach incurs some severe drawbacks: since no path information is stored, this information must be computed at query time, which significantly impacts the query performance.

If a client wants to reconstruct the path of a certain item, multiple rows must be fetched. The number of rows corresponds to the path length. As the events are inserted sequentially, the path can be reconstructed by retrieving all events of a single item sorted by the column \( t_{\text{in}} \).

2.3.2 Path Materialization Approach: Read and Path

The query performance analysis of the previous approach leads to an improvement by maintaining an additional attribute \( \text{path\_reader} \) in the RFID_READ relation as depicted in Figure 2(b). The column \( \text{path\_reader} \) contains the path the item has traversed until it has reached the current reader. To obtain the path of a certain item, only the row corresponding to its most recently processed event must be returned.

Expediting the query processing induces a slightly more expensive processing during data staging, but our performance evaluation in Section 4.1 indicates that this data staging overhead is tolerable. However, a major problem remains: Maintaining the path information in every tuple induces an intolerable space consumption. Table 3 shows example entries based on this schema.

---

1This table contains an additional attribute \( \text{path\_reader} \), which is used by the following data schema and is not part of the No Path Extension schema.
2.3.3 Bulk Movements Approach: Read and Bulk

To increase the efficiency of the business analytics process, we extend the schema to incorporate the observation according to [GHLK06] that RFID tagged items generally travel in bulks, i.e., groups along the same path. The main modification compared to the previous approaches consists of the additional table RFID_PATH which maintains the path information without redundancy. The path information is referenced from the RFID_READ table. As we consider groups of multiple items that reference the same path, the space consumption is reduced compared to the previous approach. As depicted in Figure 2(c) an entry in the RFID_READ table consists of the last reader that scanned the item, a reference to the path the item has traversed and the timestamp when the item entered the scan area of the last reader. Table 4 explains the columns in detail and Table 6 shows some sample entries.

The path information is stored in RFID_PATH. For every identified path a set of entries is generated in this table. Since many items travel along the same route, those items can share a path which is composed of sub-paths that model the situation when an item stayed at a certain reader. These sub-paths are connected via the previous_id; if an entry describes the start of a path, the previous_id is undefined. The construction of these paths is covered in the next section. Table 5 describes the columns of this table and Table 7 illustrates a sample configuration.

<table>
<thead>
<tr>
<th>RFID_READ</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>The EPC of the item</td>
</tr>
<tr>
<td>reader</td>
<td>The EPC of the reader that scanned the item</td>
</tr>
<tr>
<td>id</td>
<td>The id of the path on which the respective item currently is</td>
</tr>
<tr>
<td>t_in</td>
<td>The time when the item arrived at the current path</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RFID_READ</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>The EPC of the item</td>
</tr>
<tr>
<td>reader</td>
<td>The EPC of the reader that scanned the item</td>
</tr>
<tr>
<td>id</td>
<td>The id of the path on which the respective item currently is</td>
</tr>
<tr>
<td>t_in</td>
<td>The time when the item arrived at the current path</td>
</tr>
</tbody>
</table>

Table 3: Example Data for Data Schema Read and Path

<table>
<thead>
<tr>
<th>RFID_READ</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>The EPC of the item</td>
</tr>
<tr>
<td>reader</td>
<td>The EPC of the reader that scanned the item</td>
</tr>
<tr>
<td>id</td>
<td>The id of the path on which the respective item currently is</td>
</tr>
<tr>
<td>t_in</td>
<td>The time when the item arrived at the current path</td>
</tr>
</tbody>
</table>

Table 4: Column Description of Item Information Table RFID_READ

<table>
<thead>
<tr>
<th>RFID_PATH</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The identification of the path</td>
</tr>
<tr>
<td>previous_id</td>
<td>The identification of the previous sub-path</td>
</tr>
<tr>
<td>reader</td>
<td>The reader that scanned the item most recently</td>
</tr>
<tr>
<td>path_id</td>
<td>The string representation of the ids of sub-paths of this path</td>
</tr>
<tr>
<td>path_reader</td>
<td>The string representation of the readers which scanned the items</td>
</tr>
<tr>
<td>t_in</td>
<td>The time when the current path was reached (i.e., was first scanned by the current reader)</td>
</tr>
<tr>
<td>t_out</td>
<td>The time when the first item of the group left the range of the respective reader</td>
</tr>
</tbody>
</table>

Table 5: Column Description of Path Information Table RFID_PATH

547
Table 6: Example Item Information in the Warehouse

<table>
<thead>
<tr>
<th>item</th>
<th>reader</th>
<th>id</th>
<th>t_in</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>s3</td>
<td>p8</td>
<td>t6</td>
</tr>
<tr>
<td>i2</td>
<td>s3</td>
<td>p8</td>
<td>t6</td>
</tr>
<tr>
<td>i3</td>
<td>s4</td>
<td>p5</td>
<td>t5</td>
</tr>
<tr>
<td>i4</td>
<td>s4</td>
<td>p5</td>
<td>t5</td>
</tr>
<tr>
<td>i5</td>
<td>s5</td>
<td>p6</td>
<td>t5</td>
</tr>
</tbody>
</table>

Table 7: Example Path Information in the Warehouse

<table>
<thead>
<tr>
<th>id</th>
<th>previous_id</th>
<th>reader</th>
<th>path_id</th>
<th>path_reader</th>
<th>t_in</th>
<th>t_out</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>null</td>
<td>s1</td>
<td>“p1”</td>
<td>“s1”</td>
<td>t1</td>
<td>t2</td>
</tr>
<tr>
<td>p2</td>
<td>p1</td>
<td>s4</td>
<td>“p1;p2”</td>
<td>“s1;s4”</td>
<td>t3</td>
<td>t4</td>
</tr>
<tr>
<td>p3</td>
<td>null</td>
<td>s1</td>
<td>“p3”</td>
<td>“s1”</td>
<td>t1</td>
<td>t3</td>
</tr>
<tr>
<td>p4</td>
<td>p3</td>
<td>s2</td>
<td>“p3;p4”</td>
<td>“s1;s2”</td>
<td>t4</td>
<td>t4</td>
</tr>
<tr>
<td>p5</td>
<td>p4</td>
<td>s4</td>
<td>“p3;p4;p5”</td>
<td>“s1;s2;s4”</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>p6</td>
<td>p2</td>
<td>s5</td>
<td>“p1;p2;p6”</td>
<td>“s1;s4;s5”</td>
<td>t5</td>
<td>null</td>
</tr>
<tr>
<td>p7</td>
<td>p3</td>
<td>s2</td>
<td>“p3;p7”</td>
<td>“s1;s2”</td>
<td>t4</td>
<td>t5</td>
</tr>
<tr>
<td>p8</td>
<td>p7</td>
<td>s3</td>
<td>“p3;p7;p8”</td>
<td>“s1;s2;s3”</td>
<td>t6</td>
<td>null</td>
</tr>
</tbody>
</table>

As we will see in the comparison of the schemas in Section 4, this schema essentially increases the effectiveness of queries and reduces the amount of used disk space, but the user has to cope with a more complex database design, and the data staging runs consume more time.

3 Event Processing

An incoming event \((e, s, t)\) is first processed by the cache to maintain the most recent data. If necessary, RFID events are further processed to update the warehouse.

The event processing described in this section is based on two assumptions. First, we assume reliable RFID data streams, i.e., none of the RFID events generated by the sensors is dropped. Dealing with unreliable RFID streams is not in the scope of this paper, but is addressed in, e.g., [JGF06]. Second, the timestamps of the incoming events are increasing, i.e., “older” events cannot overtake “younger” events with smaller timestamps.

3.1 Event Processing in the Cache

For each item, the cache stores the current location and the corresponding time information in the table RFID_RAW (Table 1). The current location of the item is represented by the sensor \(s\) that most recently scanned the item. The time information comprises times \(t^F\) and \(t^R\), the item was first and most recently scanned by \(s\), as well as time \(t^L\) when the item was last scanned by the previous sensor.

For each incoming event \((e, s, t)\) the information in the cache is updated, as shown in the algorithm in Figure 3. If a new item has been scanned, i.e., EPC \(e\) is not yet stored in the cache, the algorithm inserts a new tuple for the item which stores the current sensor \(s\) of the item. Furthermore, as the item was first scanned by \(s\), \(t^F\) and \(t^R\) are initialized with the incoming timestamp \(t\). As the item has not been scanned by any other sensor, \(t^L\) does not need to be initialized. Thus, the resulting tuple for item \(e\) in table RFID_RAW is:

\[
\langle e, s, t, t, null \rangle
\]

If there is an entry for \(e\) in RFID_RAW, the algorithm checks if the item moved to another sensor. Let \(s_{\text{cur}}\) denote the sensor currently stored for item \(e\) in the cache. If \(s\) equals \(s_{\text{cur}}\), the item did not leave the range of the sensor. In this case, the algorithm only updates
Input: A RFID event \((e, s, t)\), where \(e\) denotes the EPC of the scanned item, \(s\) the sensor that scanned the item, and \(t\) the timestamp when the item was scanned by the sensor.

1. If the item is not yet stored in the cache, insert a new tuple in table RFID_RAW to store the current location of the item and the time when it has been scanned. No further processing is needed.
2. Otherwise, there is an entry for \(e\) in the cache. Let \(s_{cur}\) denote the sensor currently stored for the item.
3. Check if the item moved to a new sensor, i.e., if sensor \(s\) is different from \(s_{cur}\).
   (a) If item \(e\) stayed at the same reader, update the timestamp column for the item by setting it to \(t\).
   (b) If, on the other hand, \(e\) moved to another reader, store the value for \(e\)’s current timestamp value in subpath.out. Also, update the information about the sensor and the time when the item was first scanned.

Figure 3: Update of the Cache

\(e\)’s cache entry so that it contains the time when the item was scanned last by updating timestamp to \(t\).

If the sensor \(s\) of the incoming RFID event is not equal to \(s_{cur}\), \(e\) reached the range of a new sensor. In order to allow data staging (described in Section 3.2), the last scan time at the previous sensor \(s_{cur}\) has to be stored in subpath.out. Let \(t’\) denote the time currently stored in timestamp, i.e., the time when \(e\) was last scanned by \(s_{cur}\). Time \(t’\) is copied to subpath.out. After that, the sensor information including the scan time is updated:

\[\langle e, s, t, t, t’\rangle\]

In order to prevent indefinite cache growth due to continuous event generation by the sensors, we provide a mechanism, called cache cleaning to control the cache size. Due to space limitations, we cannot describe cache cleaning in full detail. The idea of the approach is to keep only those items in the cache that have been read within a certain time window. The information of outdated items is stored in the warehouse and can be restored if the item is scanned again.

3.2 Data Staging

The data staging process creates an aggregated view of the path information in the data warehouse. Items may travel in bulks that are defined by their borders. Each bulk is a group of items in RFID_RAW, that correspond in their current sensor \(s_i\), in their initial appearance timestamp \(t^F_i\) at this sensor, in their most recent scanning timestamp \(t_i\), and in their path which they have traversed. We denote the first item of a bulk as pioneer item. This item serves as an identifier for the whole bulk. Since the number of items and the scan order of the items inside the bulk is not fixed, the pioneer item can vary from location to location, but the varying pioneer items still identify the same bulk. Figure 4 and Table 8 show how bulks can move: Every rectangle denotes a certain bulk at the given location \(s_1\) through \(s_5\).

The path of a single bulk is a singly-linked list of path nodes. Every path node \(p_i\) stores a
pointer to the previous path \(p_{i-1}\), an RFID sensor \(s_j\), and two timestamps \(t_i^F\) and \(t_i^L\), when a bulk was first and last scanned by \(s_i\). Furthermore, we store for each path \(p_i\) a string representation \(ids_i\) of the path ids “\(p_1; \ldots; p_i\)” and the sensors \(sens_i = “s_1; \ldots; s_i”\) on the path.

The backlink of an RFID path’s starting point \(p_1\) is undefined (null), thus, the entry for the first step on a path is \(\langle p_1, \text{null}, s_1, “p_1”, “s_1”, t_1^F, t_1^L \rangle\). The \(n\)-th entry of a path is \(^2\)

\[
\langle p_n, p_{n-1}; s_n, ids_{n-1} \oplus “p_n”, sens_{n-1} \oplus “s_n”, t_n^F, t_n^L \rangle
\]

If a bulk of items \(e_1, \ldots, e_k\) moves from sensor \(s_{n-1}\) to \(s_n\), the path information in table RFID_PATH has to be generated only for the pioneer item \(e_1\). The path information is reused for all other items \(e_2, \ldots, e_k\) of that bulk.

To reduce space consumption of the path information, different bulks can share path nodes if they are identical, i.e., if the bulks initially travel along the same route and then split up. So, the overall structure regarding all bulks can be viewed as a forest.

We present two different data staging algorithms which differ in the time when they are triggered. **Tuple-wise data staging**, which is described in Section 3.2.1, is triggered each time an incoming RFID event at the cache indicates that the corresponding item moved to a new location. In contrast to that, **bulk data staging**, described in Section 3.2.2, updates the warehouse for several items in a batch.

### 3.2.1 Tuple-wise Data Staging

With **tuple-wise data staging**, the information in the tables RFID_PATH in the cache and RFID_READ in the warehouse are asynchronously updated if item \(e\) is scanned for the first time or if it reaches the range of a new reader. The input parameter for the data staging process is a single tuple from the cache table RFID_RAW, containing, amongst others, the time \(t_{n-1}^L\) when \(e\) left the last reader and time \(t_n^F\) when the item reached the range of \(s_n\).

An item that is scanned for the first time cannot have a time information about when it left the previous reader. Thus, if \(t_{n-1}^L\) is not set, we can identify an item that is scanned for the first time. In this case, we generate a new path \(p_1\) that has no predecessor. Additionally, we store the sensor \(s_1\) and the time when the item is scanned by the sensor. Since \(e\) is on the first step of its path, “\(p_1\)” and “\(s_1\)” constitute the string of paths and the string of passed sensors, respectively.

\[
\langle p_1, \text{null}, s_1, “p_1”, “s_1”, t_1^F, \text{null} \rangle
\]

\(^2\oplus\) denotes string concatenation with a delimiter “;”
Input: Single tuple \{item: e, reader: s_n, t_in: t_{n}^E, timestamp: t_n, subpath_out: t_{n-1}^L\}, transferred from RFID_RAW, where e denotes the EPC of the scanned item, s_n the sensor that scanned e, t_{n}^E the time when e reached the range of s_n, t_n the timestamp, when s_n scanned e most recently, and t_{n-1}^L the time when the previous sensor last scanned e.

\[
\text{IF (} t_{n-1}^L = \text{null} \text{) THEN (Item scanned for the first time)}
\]

- Generate new path \( p_1: \{p_1, \text{null, } s_1, \text{null, } s_1, \text{null, } t_1^F, \text{null}\} \)
- Insert new entry for e with a reference to \( p_1 \) in RFID_READ.

\[
\text{ELSE (The item is scanned by a new sensor)}
\]

1. Retrieve e’s path \( p_{n-1} \), currently referenced in RFID_READ’s attribute id.

\[
\text{STEP 2: Terminate paths:}
\]

- Let \( p_{TERM} \) denote the terminated path.
- \[
\text{IF (} p_{n-1}.t_{\text{out}} = \text{null} \text{) THEN}
\]

\[
\begin{align*}
&\text{IF (} p_{n-1}.t_{\text{out}} := t_{n-1}^L; p_{TERM} := p_{n-1} \\
&\text{ELSE IF (} p_{n-1}.t_{\text{out}} = t_{n-1}^L) \text{ THEN}
\end{align*}
\]

\[
\begin{align*}
&\text{p}_{TERM} := p_{n-1} \\
&\text{ELSE}
\end{align*}
\]

- (a) Check, if there is already a path \( p_{TERM} \) with \( \text{previous}_id = p_{n-1}.\text{previous}_id, \text{reader} = p_{n-1}.\text{reader}, \text{t}_{\text{in}} = p_{n-1}.\text{t}_{\text{in}}, \)

\[
\begin{align*}
&\text{and } \text{t}_{\text{out}} = t_{n-1}^L.
\end{align*}
\]

- (b) If such a path exists, reuse this path as \( p_{TERM} \).

- (c) Otherwise, if such a path does not exist, create \( p_{TERM} \) by copying \( p_{n-1} \).

\[
\begin{align*}
&p_{TERM}.t_{\text{out}} := t_{n-1}^L
\end{align*}
\]

3. For the pioneer item, create the path \( \{p_n, p_{TERM}, s_n, ids_n, sens_n, t_n^E, \text{null}\} \), for all other items in the bulk reference the pioneer item’s path.

4. Update e’s entry in RFID_READ, set id = p_n and t_{\text{in}} = t_{n}^E.

\[\text{Figure 5: Tuple-wise Data Staging}\]

If the item moved from a sensor to a new sensor \( s_n \), the information when \( e \) left the path currently stored in RFID_READ has to be maintained. An item that moved from one location to another is identified by a value for \( t_{n-1}^L \) which is not equal to \( \text{null} \). This timestamp is used to terminate the current path of the item, i.e., the timeout information of e’s current path must be updated to contain \( t_{n-1}^L \), before we can actually create \( p_n \). Let \( p_{TERM} \) denote the terminated path. The update of the timeout information of paths depends on how bulks are split. Consider a bulk \( B \) at sensor \( s_{n-1} \) which splits into two bulks \( B_1 \) and \( B_2 \). There are three cases for the termination of a path.

First, \( e \) is the first item of bulk \( B \) to be scanned by its new sensor \( s_n \). In this case, the timeout information \( t_{\text{out}} \) in \( p_{n-1} \) is not yet set because \( e \) is the first item that leaves the range of \( s \). For this purpose, we update \( t_{\text{out}} \) to \( t_{n-1}^L \).

Second, consider a bulk \( B \) that splits into two bulks \( B_1 \) and \( B_2 \). All items of \( B \) leave the range of \( s_{n-1} \) in parallel at time \( t_{n-1}^L \). Furthermore, assume that at least one item in \( B_1 \) has already been processed as described above, i.e., \( p_{n-1} \)’s timeout value is \( t_{n-1}^L \). Let \( e \) denote the first item of bulk \( B_2 \) to be processed. In this case, the timeout information in
path $p_{n-1}$ is already correct for $e$, there is nothing more to do for path termination.

Third, in contrast to case two, the items in bulk $B_2$ stay longer at sensor $s_{n-1}$ than the items in $B_1$ which left the range of $s_{n-1}$ at time $t_{n-1} < \hat{t}_{n-1}$, i.e., $p_{n-1}$’s timeout information is $\hat{t}_{n-1}$. Let $e$ denote the first item of $B_2$ to reach a new sensor. In this case, we look for a path $p_{\text{TERM}}$ which stores the same information as $p_{n-1}$, but with correct timeout information. If no such path exists, we create $p_{\text{TERM}}$ by copying $p_{n-1}$ and setting the timeout information of $p_{\text{TERM}}$ to $\hat{t}_{n-1}$.

After terminating the path for $e$, the movement of $e$ to the new sensor $s_n$ has to be stored. For this, a new path $p_n$ with $p_{\text{TERM}}$ as predecessor is created. In order to build the string of path identifiers $\text{ids}_n$, we concatenate $\text{ids}_{\text{TERM}}$ (string containing all paths from $p_1$ to $p_{\text{TERM}}$) and “$p_n$”. Similarly, the string of sensors $\text{sens}_n$ passed by the item is the concatenation of the $\text{sens}_{\text{TERM}}$ (string of sensors from $s_1$ to $s_{\text{TERM}}$) and “$s_n$”. Furthermore, $p_n$ stores the sensor $s_n$ and the respective scan time $t_n^\text{F}$:

$$\langle p_n, p_{\text{TERM}}, \text{ids}_{\text{TERM}} \oplus "p_n", \text{sens}_{\text{TERM}} \oplus "s_n", t_n^\text{F}, \text{null} \rangle$$

In order to link the newly generated path to the item, the algorithm updates the item information for $e$ in $\text{RFID}_{\text{READ}}$ to reference the newly generated path $p_n$:

$$\langle e, s_n, p_n, t_n^\text{F} \rangle$$

The algorithm for the tuple-wise data staging is shown in Figure 5.

**Example** An initial bulk of six items is successively split resulting in three bulks of two items that move from sensor to sensor. The path of the items is depicted in Figure 4 and the corresponding timestamps, when the sensors scan the items, are listed in Table 8. At the beginning, the tables in the cache and in the warehouse are empty. The schemas of the warehouse tables are depicted in Table 4 for $\text{RFID}_{\text{READ}}$ and in Table 5 for $\text{RFID}_{\text{PATH}}$. For brevity, we omit the description for items that do not reach a new sensor, where only the information in the cache is updated.

**Time $t_1$:** Data staging is triggered after the first incoming event $\langle i_1, s_1, t_1 \rangle$ (item $i_1$ is scanned by sensor $s_1$ at time $t_1$) is inserted into the cache. The warehouse does not yet contain any information about items, so that we insert path $\langle p_1, \text{null}, s_1, "p_1", "s_1", t_1, \text{null} \rangle$ into $\text{RFID}_{\text{PATH}}$. Additionally, we insert a new entry for $e_1$ which references path $p_1$ into $\text{RFID}_{\text{READ}}$. All other items scanned at $t_1$ are on the same path as $i_1$. When the corresponding events for $i_2, \ldots, i_6$ arrive, the cache is updated and data staging is triggered. As the suitable path already exists, we only insert the corresponding entries in $\text{RFID}_{\text{READ}}$ with references to $p_1$.

**Time $t_2$:** (no data staging)

**Time $t_3$:** Data staging is performed for $i_5$ which moves along with $i_6$ to sensor $s_4$. Since $i_5$ is the first item scanned by $s_4$, it is identified as pioneer item. The current path of $i_5$, $p_1$, does not store a value for timeout information. Thus, we update $t_{\text{out}}$ to contain $t_2$. After creating a new path $p_2$ with predecessor $p_1$ to reflect $i_5$’s movement to a new sensor, $\text{RFID}_{\text{READ}}$ is updated to reference $p_2$. For $i_6$, data staging only updates the corresponding entry in $\text{RFID}_{\text{READ}}$ to reference $p_2$. 

552
Time $t_4$: The cache is updated when $i_1$ moves to sensor $s_2$. During data staging, $i_1$ is identified as pioneer item because it is the first item scanned by $s_2$. The timeout information for $i_1$’s current path $p_4$ contains time $t_2$ (data staging for $i_5$ at time $t_3$). In order to store the correct timeout information, we create a duplicate path $p_3$ of $p_4$ with correct timeout information. The newly generated path is predecessor for $p_4$, which stores $i_1$’s movement to $s_2$. The item information of $i_1$ in RFID_READ is updated to reference path $p_4$. Data staging for items $i_2, \ldots, i_4$ updates the respective entries in RFID_READ to reference $p_4$.

Time $t_5$: The arrival of $i_3$ at sensor $s_3$ triggers data staging. The timeout information of $i_3$’s current path is set to $t_4$ (the last time when $i_3$ was scanned by its previous reader). We use the terminated path as predecessor for path $p_5$ which is used to update RFID_READ for $i_3$ and, finally, $i_4$. When $i_5$ moves to sensor $s_5$ at the same time, path $p_2$ is terminated. We create a new path $p_8$ which references $p_2$ as predecessor. The entry for $i_5$ in RFID_READ is updated to reference $p_8$. For item $i_6$, the path information can be reused.

Time $t_6$: When $i_1$ and $i_2$ reach sensor $s_3$, $i_1$’s current path $p_4$ already has a timeout value $t_4$, which is different from the timestamp $t_5$ when $i_1$ was last scanned by $s_2$. Thus, we duplicate $p_4$ by creating $p_7$ which contains $t_5$, i.e., the time when $i_1$ was last scanned by the previous reader, as timeout information. Since $i_1$ moved to a new sensor, a new path $p_8$ is generated. Finally, we update RFID_READ for $i_4$. When $i_2$ is scanned by $s_3$, only the information for that item is updated to reference path $p_8$. For all other incoming events at that time, only the cache has to be updated.

The resulting tables in the warehouse are depicted as Tables 6 and 7.

3.2.2 Bulk Data Staging

In contrast to the tuple-wise data staging algorithm where the warehouse is updated on a per-tuple basis, the bulk data staging algorithm updates the warehouse for several items that changed their location in a batch.

To allow for bulk data staging, the location change of an individual item is not immediately propagated to the warehouse. For this purpose, we store the time when the item was scanned for the last time by the previous reader. Only if this information is already set, i.e., the item moves to yet another sensor, data staging is enforced. In such a case the cache update for the item is delayed to avoid overwriting the old item state. The old item state is necessary for a correct data staging. After the data staging the cache is updated.

The first step of bulk data staging is to select the set of candidate events for data staging. This set contains the following items: Let $t_{last}$ denote the time of the last data staging run and $t_{now}$ the current point in time. The algorithm selects all entries from the cache table RFID_RAW whose values for column timestamp are in $[t_{last}, t_{now}]$. The selected events are all events that may have had a location change in the past. Since scanning individual items is not an atomic operation, we must exclude the timestamp $t_{now}$, i.e., probably not all events of $t_{now}$ are already in the cache. Events with timestamp $t_{now}$ are processed in the following data staging run.

This set constitutes the input for the bulk data staging algorithm shown in Figure 6. It is first joined with RFID_READ to get the current path $p_{n-1}$ for each item in the set. The set contains one or more bulks – identified by their pioneer items – that must be extracted for
Input: Set of tuples \( \{ \text{item}: e, \text{reader}: s_n, \text{timestamp}: t_n, \text{subpath out}: t_{n-1} \} \)

transferred from RFID_RAW, where \( e \) denotes the EPC of the scanned item, \( s_n \) the sensor that scanned \( e \), \( t_{n}^{F} \) the time when \( e \) reached the range of \( s_n \), \( t_n \) the timestamp, when \( s_n \) scanned \( e \) most recently, and \( t_{n-1}^{L} \) the time when the previous sensor last scanned \( e \).

1. Left outer join input with RFID_READ on item. Resulting tuples \( \{ e, s_n, t_{n}^{F}, t_n, t_{n-1}^{L}, p_{n-1} \} \), where \( p_{n-1} \) denotes the path id of the path the item \( e \) had traversed so far, if it exists.

2. Group this resulting set by reader, path, incoming and current timestamp to obtain a set of bulks.

3. For all bulks \( B \) of this set:
   (a) Select the first item of the bulk as pioneer item \( e_p \).
   (b) Generate path information for pioneer item \( e_p \) only:
      - IF \( (p_{n-1} = \text{null}) \) THEN (The item has been scanned for the first time)
        - Create new path \( p_1: (p_1, \text{null}, s_1, "p_1", "s_1", t_{1}^{F}, \text{null}) \) for \( e_p \) in RFID_PATH
      - ELSE (The item is scanned by a new sensor.)
        i. Retrieve \( e_p \)'s path \( p_{n-1} \), currently referenced in RFID_READ's attribute id.
        ii. Terminate paths (see (⋆) in Figure 5)
        iii. Create path \( (p_n, p_{\text{TERM}}, s_n, \text{ids}_n, \text{sens}_n, t_{n}^{F}, \text{null}) \) for \( e_p \).
   (c) Update RFID_READ for all items in \( e_p \)'s bulk, set \( \text{id} = p_n \) and \( \text{t.in} = t_{n}^{F} \) for \( e_p \) and all items \( e \) within the bulk.

Figure 6: Bulk Data Staging

further processing. The path information created with the pioneer items is then applied to all other items inside the bulk.

If the pioneer item was never scanned before, there are no matching entries in the tables RFID_READ and RFID_PATH. In this case a path is created. For a new pioneer event \( \{e, s_1, t_1^{F}, \text{null} \} \) the tuple \( \{p_1, \text{null}, s_1, "p_1", "s_1", t_{1}^{F}, \text{null} \} \) is created in RFID_PATH. Instead of creating new paths, no longer referenced paths can be reused.

If a sensor already scanned an item, there are existing entries in tables RFID_READ and RFID_PATH. As in tuple-wise data staging, the algorithm retrieves this information, and then determines whether the item has moved to a new sensor or it stayed at its old position.

If an item has moved, the currently stored path \( p_{n-1} \) must be terminated accordingly. The procedure is similar to the tuple-wise data staging. At last, a new path is created with the terminated path as predecessor and with \( \text{t.in} \) set to \( t_{n}^{F} \). This new path is then referenced from the items inside the bulk.

The path information is only computed for the pioneer item of a bulk. The resulting path ID is cached, so it can be applied to all other events inside the bulk without a need for calculating their path information. A new calculation is only necessary if the algorithm identifies a new bulk inside the set of candidates.

Example  This example is based on the same movements and scan times of the items as for the previous example. The path of the items can be found in Figure 4 and the scan times in Table 8. As before, the tables in the cache and in the warehouse are empty at the
beginning.

**Times t₁ and t₂:** (Only cache operation, no data staging.)

**Time t₃:** When item i₅ reaches the range of s₄ at t₃, data staging is triggered. The set of data staging candidates contains all items with \( t₁ < \text{timestamp} < t₄ \) stored in the cache. This selects only items i₅ and i₆, because the cache has not been updated for these items. During data staging, a new path \( p₁ \), which represents that the items were scanned by s₁ for the first time at \( t₁ \), is generated. After that, the cache is updated for items i₅ and i₆ to reflect the movement of the items from s₁ to s₄.

**Time t₄:** When item i₁ reaches the range of sensor s₂ for the first time, data staging is triggered again. The set of candidates contains all items with \( t₃ < \text{timestamp} < t₄ \), i.e., items i₁ to i₄. The items belong to two bulks, represented by the pioneer items i₁ (the bulk that has been at s₁) and i₅ (the items that moved to s₄). For i₁, the path \( p₁ \) can be used because the \( \text{t}_{\text{out}} \) information is not set for this path.

Item i₅ is currently located on the not yet terminated path \( p₁ \). Path \( p₁ \) is terminated with \( t₂ \), and a new path \( p₂ \) with predecessor \( p₁ \) is created. For i₅ and i₆ RFID_READ is updated with the newly generated path.

**Time t₅:** Data staging is triggered when item i₃ arrives at s₄. The set of candidates contains all events with \( t₃ < \text{timestamp} < t₅ \), including pioneer items, i₃ and i₅. Currently, item i₃ is on path \( p₁ \), so a path similar to \( p₁ \), but with termination time \( t₃ \) is needed. There is no path with these characteristics, so the algorithm duplicates \( p₁ \) and creates path \( p₃ \) with the correct \( \text{t}_{\text{out}} \) information. Path \( p₃ \) is then used as predecessor path for \( p₄ \), which represents the movement of items i₃ and i₄ to s₄. The path \( p₄ \) is linked to items i₃ and i₄. Finally the pending cache update is done and the remaining events are processed.

**Time t₆:** The event for item i₁ triggers data staging. The data staging candidates are all items with \( t₄ < \text{timestamp} < t₆ \). The set of candidates contains three bulks represented by three pioneer items. For the bulk that moves along with pioneer item i₁, the algorithm finds the non-terminated path \( p₄ \) as a suitable path to be further used. Pioneer item i₃ finds the path \( p₄ \), too, but terminates it with timestamp \( t₄ \). Then the new path \( p₅ \) is created with \( p₄ \) as predecessor and linked to the bulk. At last, item i₆ takes the not terminated path \( p₂ \), terminates it with \( t₄ \) and creates a new path \( p₆ \). After this data staging no further data staging is needed, and only cache updates remain for the following incoming events.

For the last data staging run all events are selected, whose \( \text{timestamp} \) represents a time after the previous data staging run. If we apply this to our example, the last pioneer item is item i₆. This item is currently on path \( p₄ \). Searching for a suitable path, the algorithm does not find any, so \( p₄ \) must be duplicated. The resulting path \( p₇ \) is then used as predecessor path for path \( p₈ \). The last path is then used for items i₄ and i₂. All other bulks did not move, so no further processing action is necessary.

### 4 Evaluation

In this section, we will compare the different database schemas, described in Section 2.3, by their space requirements and their query performance. Furthermore, based on estima-
tions about the event rate of the German retailer Quelle and the car manufacturer BMW, we will assess the performance of the data staging algorithms.

4.1 Comparison of the Different Data Warehouse Schemas

As stated in Section 3.2, most items tend to move in bulks. For a comparison of the different database schemas, we adopted the tree model of [GHLK06]: Each node in the tree represents a location where a set of items has been scanned, while an edge represents the movement of objects between two locations. This model assumes that items at locations near the root move in larger groups, and with increasing tree depth the groups become smaller. The bulkiness \( B = (s_1; s_2; \ldots; s_n) \) defines the maximum size \( s_i \) of a group at level \( i \), with \( n \) being the path length. Our assumption implies \( s_i \geq s_j \) for \( i < j \). For example, a bulkiness of \((10; 4; 2)\) denotes that a bulk at level 1 contains ten items, at level 2 there are two bulks with four items each, and one with two items, and at level 3 each previous bulk is split into bulks containing two items.

The data set used for this comparison consists of 5 million entries, generated by 1 million items passing by 5 RFID sensors. The query *What is the average time for product X to go through locations \( L_1, \ldots, L_k \) entering \( L_k \) between times \( t_1 \) and \( t_2 \)?* was chosen, since it has to walk through all these entries. In order to evaluate the impact of the size of the travel groups, we defined three different degrees of bulkiness \( B_1 = (500; 230; 125; 63; 31) \), \( B_2 = (500; 150; 40; 8; 1) \), and \( B_3 = (100; 10; 1; 1; 1) \).

As shown in Figures 7(a) and 7(b) the space requirements and the query processing time using the No Path Extension and the Read and Path data schema are independent of the bulkiness. These data schemas store the movement of an item on a per-item level. In contrast to that, the Read and Path schema stores the movement of items on a per-bulk level, resulting in less entries to be stored per item. Thus, the higher the size of the bulks the higher is the reduction of disk space usage with the Read and Bulk schema. Using schemas No Path Extension and the Read and Path, the path information for the example query must explicitly be reconstructed by \( n \) self-joins where \( n \) is the path length. With the Read and Bulk approach, this information can be easily obtained with a single join between tables RFID_READ and RFID_PATH. For large bulks, this results in a significant speed-up in query processing performance, as shown in Figure 7(b).
If the bulk sizes along the path for all items are small, the space requirements and the query processing time using the Read and Bulk schema are higher than using the other approaches. But in practice, we even expect bigger bulks than the ones we used for the presented scenario with 1 million items passing 5 sensors. Large containers or pallets can carry many more items. 5 RFID sensors constitute a rather low estimate considering that items are tracked throughout a supply chain. Thus, the Read and Bulk schema benefits even more compared to the other data schemas.

4.2 Estimation of Event Rates

We analyze the requirements of two representative companies: Quelle, a retailer and BMW, a manufacturer.

According to their website, the German mail order company Quelle ships 771,000 items per day from their most modern facility at Leipzig. Let us assume that each item passes by five RFID sensors, and that the facility is on duty eight hours a day. To satisfy these needs, the RFID architecture must handle at least 133 RFID events per second.

The values above model just the outward movement of goods. The event rate increases if automated stock keeping, where stock items are scanned frequently, is also modeled. In 2005, BMW manufactured about 1.2 million cars, each consisting of about 20,000 parts. We assume that BMW tags about 1,000 of these 20,000 compounds with RFID tags, and 20 events per part are generated before the part is built into the car. Furthermore, we assume that BMW has 16 working hours a day and 220 working days per year. Together, all BMW facilities produce about 1,890 events per second. If we look at the facility at Leipzig, where currently 650 cars per day are produced, the event rate is about 226 events per second.

4.3 Evaluation of the RFID Data Staging Algorithms

The focus of the last part of this section is the performance analysis of the data staging algorithms.

Benchmark Configuration For the benchmarks, the middleware and the cache were running on the same host, equipped with an Intel Xeon 2.80 GHz CPU and 1 GB main memory. Since the cache and the middleware were placed on the same host, the JDBC driver used DMA for data transfer. The warehouse was placed on a host with a dual Intel Xeon 3.20 GHz EM64T CPU and 8 GB main memory. The simulated RFID sensors and the query clients were distributed over 8 hosts whose performance characteristics were similar to the middleware host. For every sensor or client, a dedicated socket connection to the middleware is opened.

Three different parameters – as explained in Figure 8 – affect the performance of the RFID architecture.

Since the number of items has no influence on the performance, as long as the dataset fits into main memory, this value is constantly held at 10,240. The path length has only an impact on the storage consumption inside the data warehouse, so this parameter was set to 5. The impact of storage consumption has been discussed above.
Bulkiness  The bulkiness specifies the hierarchical splittings of the individual bulks. This models a typical supply chain where items initially moved in larger containers.

Duplicate reads  This parameter specifies the number of scans at a certain location. At a given location, the event is at least scanned once, but not more often than the value of this parameter. The intention behind this parameter can be found in stock keeping. If an item stays at a certain location for a longer period of time, an RFID reader may scan the item more than once.

OLTP/OLAP query ratio  Every query client issues OLTP or OLAP queries. The ratio between these two types is specified by this parameter.

The evaluation system creates a tree of bulk movements, similar to the one used previously in the examples. To model some fuzziness, some of the leaves are cut off randomly, such that less than 1% of the items get lost.

RFID Queries  The evaluation system generates some background load using different lightweight OLTP queries on the cache data and heavy OLAP queries executed on the historic data in the warehouse. The ratio between these types of queries is defined as described in Figure 8. An example lightweight query is

Where is item i now?

whereas a heavy OLAP query is

What items need more time to get from l_i to l_j than the average of items of the same type?

We implemented several different query types for OLTP and OLAP but omit the description due to space limitations.

Results  For each of the parameters in Figure 8, we performed different benchmarks. To evaluate the impact of a parameter on the throughput for data staging, we always varied one parameter while holding the remaining two parameters constant.

The basis for all benchmarks are 10,240 items which pass 5 RFID sensors with a bulkiness of $B_2 = (10,240; 3,520; 320; 80; 5)$. Every item is scanned twice by the same reader before moving to another location. 30% of the user queries are OLAP queries. This standard case is represented by the middle bars in Figures 9(a), 9(b), and 9(c).

Figure 9(a) shows the impact of the bulkiness on the performance of the data staging. The parameter bulkiness varies the number of items that travel together in one group. Bulkiness $B_1 = (10,240; 860; 75; 30; 5)$ denotes a scenario where the groups of items traveling together are smaller than in the standard case. In contrast to that, bulkiness $B_3 = (10,240; 6,500; 3,000; 1,250; 320)$ models bigger groups of items along the path. While the throughput with tuple-wise data staging is nearly constant (variation of less than 7%), the impact of bulk data staging is quite high. There is an 25% performance improvement with big bulks and less splits because a higher number of items are processed in one batch in bulk data staging.

The number of duplicates, i.e., how often the same sensor scans an item, has a significant impact on the performance of data staging, as shown in Figure 9(b). While the throughput
for tuple-wise data staging is doubled if all events are scanned eight times instead of two times, the bulk data staging has a performance boost of one third. The more duplicate reads there are, the more transactions can be handled entirely by the cache, i.e., there are no updates of the warehouse.

As depicted in Figure 9(c), the variation of the ratio between lightweight (OLTP) and heavyweight (OLAP) queries has no impact on the throughput of event processing. This shows that the limiting factor for the performance of the data staging is not the warehouse, but the cache and the middleware. We use this as an indicator that our architecture has a scalability potential by replicating and distributing the main-memory caches. Handling RFID events and queries with distributed caches is ongoing work and is beyond the focus of this paper.

The average event processing throughput of our system is about 1500 events per second for bulk data staging, and about 850 events per second for tuple-wise data staging. As expected, bulk data staging performs better for two reasons: First, events are processed in efficient batch operations within the warehouse and second, the data staging process is triggered less often.

The data staging algorithms does not have significant impact on the query execution time of lightweight and heavyweight queries. While the data staging algorithms are running, there was no noteworthy performance degradation (between 1% and 3%) for the individual queries, compared to the data stores not loaded with any data staging algorithm.

5 Related Work

Many application service vendors, including IBM [IBM], HP [HP], Microsoft [Mic05], Oracle [Ora], SAP [BLHS04], Siemens [WL05, WLLB06], and Sun [GS04] are working on their own RFID middleware infrastructures. These frameworks transform the proprietary events stemming from the RFID sensors into events that can be processed by high-level applications. The generic design of such RFID middleware solutions does not explicitly store paths of tracked items, although RFID events are automatically stored in the database. Our approach could thus be viewed as an extension which exists on top of existing RFID middleware.
There are different approaches to reduce the storage space for RFID data in databases. Hu et al. [HSCS05] describe an Oracle bitmap datatype which exploits the structure of the EPC to reduce the amount of data to be stored. The bitmap datatype represents a collection of EPCs which share a common prefix, e.g., items stemming from the same manufacturer. For all items, the prefix is only stored once. To further reduce the space requirements, a compressed representation of the EPC suffixes (the serial numbers of the items) is maintained in the data type. Wang and Liu [WL05] propose a temporal data model for RFID data for object tracking and monitoring. In order to reduce the storage space for the RFID data, they aggregate information by exploiting (hierarchical) containment relationships among objects, e.g., a pallet loaded with cases. In contrast to our approach, the data model does not store the path of a tracked object.

An approach similar to our work is presented by Gonzales et al. [GHLK06]. They use RFID Cuboids to store aggregated path information at different abstraction levels. Although they identify incremental updates as crucial point for RFID applications, their work does not describe incremental updates of the path information in the warehouse.

6 Conclusion and Future Work

In this work we devised an information system for supporting OLTP and OLAP queries on RFID data at the same time. It proposes algorithms for generating the path information at data staging time, to facilitate OLAP queries for RFID tracking and supply chain analysis. Different data warehouse schemas for storing the path information are evaluated and implemented in an RFID tracking infrastructure, consisting of a main memory DBMS used as cache and an RDBMS used as a data warehouse.

Our evaluation shows the impact of different factors on the data staging algorithms. Bulk data staging leverages the efficient bulk operations of DBMSs and demonstrates its advantage over propagating events to the data warehouse tuple by tuple.

Supporting multiple caches is part of our ongoing work. We are implementing an RFID routing system that passes the incoming events to distributed caches, whereby a specific item is always in the same cache. Further development consists of dynamic routing and load balancing strategies. In the future, we will investigate how much of the routing functionality can be placed inside intelligent networks, e.g., Cisco SONA, or inside a publish-subscribe-architecture, where the RFID sensors publish their data streams and the caches subscribe those events that are interesting for their clients. In this context we will also investigate RFID application-specific data stream mining techniques, that, e.g., raise alarms, if certain supply chain abnormalities (correlated events) occur.

Acknowledgments

We would like to thank Wolfgang Becker, Tobias Brandl, and Dr. Ulrich Marquard of SAP’s Research and Breakthrough Innovation Technology and Foundation group for their cooperation. Also, we would like to thank Holger Pawlita for helping in the implementation of the Evaluation System, and Christian Sosnowski for the detailed comparison of the database schemas.
References


[GHLK06] Hector Gonzalez, Jiawei Han, Xiaolei Li and Diego Klabjan. Warehousing and Analyzing Massive RFID Data Sets. In Proceedings of the 22nd International Conference on Data Engineering, pages 83–94, 2006.


Ein Nachrichtentransformationsmodell für komplexe
Transformationsprozesse in datenzentrischen
Anwendungsszenarien

Matthias Böhm 1
Systemberater

Uwe Wloka 2
Lehrgebiet Datenbanken

Dirk Habich 3
Lehrstuhl Datenbanken

Jürgen Bittner 1
Geschäftsführer

Wolfgang Lehner 3
Lehrstuhl Datenbanken

1 SQL GmbH Dresden 2 HTW Dresden (FH) 3 TU Dresden
Franklinstraße 25a Friedrich-List-Platz 1 Nöthnitzer Str. 46
D-01069 Dresden D-01069 Dresden D-01187 Dresden
transconnect@sql-gmbh.de wloka@htw-dresden.de dbgroup@mail.inf.tu-dresden.de

Abstract: Die horizontale Integration von Systemen durch eine nachrichtenbasierte
Kommunikation über Middleware-Produkte stellt eine, sich immer weiter verbrei-
tende, Art der Anwendungsintegration dar, um eine hinreichend lose Kopplung der
partizipierenden Systeme und Anwendungen zu gewährleisten. Für die Beschreibung
derartiger Integrationsprozesse kommen zunehmend funktional-orientierte Prozessbe-
schreibungssprachen wie beispielsweise WSBPEL zum Einsatz, welche allerdings De-
fizite bei der Beschreibung von datenzentrischen Anwendungsszenarien offenbaren.
Dieses Papier leistet einen Beitrag zur systematischen Modellbildung für komplexe
Nachrichtentransformationen in datenzentrischen Prozessen. Die Realisierbarkeit der
Ergebnisse wird dabei an der Integrationsplattform TransConnect©, der Firma SQL
GmbH, verdeutlicht.

1 Einleitung

Mittlerweile hat sich auf der Ebene der Prozessintegration mit WSBPEL eine Prozessbe-
schreibungssprache zur Orchestrierung von Diensten in einer Service Oriented Archite-
cture (SOA) weitestgehend durchgesetzt. Ferner kommen aber zunehmend auch im Rahmen
datenzentrischer IntegrationszENARIOEN von MessageBroker-Systemen über EAI-Servers
bis zu ETL-Tools, ähnliche Beschreibungsmittel zum Einsatz. Dabei existieren allerdings
derzeit keine allgemeingültig, anerkannten Modelle respektive Standards, welche eine ein-
heitliche externe Sicht auf datenzentrische Integrationsprozesse gewährleisten.
In Anlehnung an die Arbeit [MMLW05] kann prinzipiell eingeschätzt werden, dass so-
wohl in Workflow- als auch in ETL-Beschreibungen Aspekte des Kontroll- und Datenflus-
ses abzubilden sind. Dabei weisen die tupelorientierten Workflow-Systeme umfangreiche
Möglichkeiten der Spezifikation des Kontrollflusses, allerdings Defizite hinsichtlich der

Diese Motivation und Problemstellung hat auch die Anforderungen an die Entwicklung von TransConnect® 2.0, als nachrichtenbasierte Integrationsplattform, maßgeblich beeinflusst. So bestand die Notwendigkeit der Definition eines konzeptuellen Modells und dessen Abbildung mit Prozessbeschreibungssprachen, um einerseits die höchstmögliche Flexibilität bei der Modellierung von Integrationsprozessen und andererseits die Datennahängigkeit und die Unabhängigkeit von den konkreten Prozessbeschreibungssprachen zu gewährleisten.

Entsprechend der Kategorisierung von Integrationsformen nach [DMMW03] und [Her03], liegt der Fokus dieser Arbeit also auf der Definition eines generischen, konzeptuell Modells zur Beschreibung von Prozessen der Informations- und Anwendungsintegration mit Methoden und Standards der Prozessintegration.


2. Datentypen: Syntaxische Abbildung der Datenfelder einer Nachricht samt Datentypen

1. Datenrepräsentation: verlustfreie Formatkonvertierungen, wie die Überführung von CSV in XML, aber auch Kompression und Verschlüsselung

0. Transport: verlustfreie Transformation zum Zwecke der Übertragung mit bestimmten Transportprotokollen

So hat die elementare Nachrichtentransformation, im engeren Sinne der Transformation, die Ebenen 0 bis 2 zum Gegenstand. Darauf aufbauend betrifft die komplexe Nachrichtentransformation, im weiteren Sinne, alle vier Ebenen der Transformation. Auf Grund der Komplexität solcher Transformationen können diese nur als Folge von kontrollfluss- und datenfluss-orientierten Teilschritten abgebildet werden.

Die Abbildung 1 zeigt ein Beispielszenario im Rahmen des ETL-Prozesses, welches mit dem hier vorgestellten Nachrichtentransformationsmodell abzubilden ist. Die Problemstellung umfasst dabei die Übernahme von Stamm- und Bewegungsdaten aus Quellsystemen von zwei eigenständigen Vertriebs- und Einkaufsorganisationen in ein zentrales Data

Abbildung 1: Untergliederung des Beispielszarnarios in Teilprozesse


2 Verwandte Arbeiten

Die Prozessbeschreibungssprache WSBPEL 2.0 [OAS06] und deren Defizite hinsichtlich der Modellierung des Datenflusses ist als Ausgangspunkt dieser Arbeit anzusehen. Zwar existiert mit II4BPEL [IBM05] ein Erweiterungsvorschlag, der ähnliche Ziele wie diese Arbeit adressiert, jedoch beschränken sich die darin formulierten Erweiterungen ausschließlich auf die Interaktion mit DBMS. Neben WSBPEL existieren alternative Sprachen, welche eine explizite Trennung des Kontroll- und Datenflusses vorsehen und so-

3 Anforderungen an ein Nachrichtentransformationsmodell


- Interaktion mit beliebig vielen Quell- und Zielsystemen
- Abbildbarkeit des synchronen und des asynchronen Verarbeitungsmodells
- Ermöglichung des Content Based Routings durch eine geeignete Anfragesprache
- Umgang mit unstrukturierten, semistrukturierten und strukturierten Daten
- Umgang mit unterschiedlichen Datenmengen bis hin zur "Nachrichtenmenge"
- Transformation der Semantik von Nachrichten durch Hinzufügen, Ändern und Entfernen von Attributen einer Nachricht
- Unterstützung spezifischer Methoden der elementaren Nachrichtentransformation
- Implizite oder explizite Validierung von Nachrichten

Im Gegensatz zu den funktionalen Anforderungen beschreiben die nicht-funktionalen Anforderungen Rahmenbedingungen und Charakteristika der Nachrichtentransformation.

- Effiziente und skalierbare Verarbeitung (Datenmenge und Parallelität)
- Zuverlässige Verarbeitung durch ein Transaktionskonzept
- Hohe Flexibilität durch die Abstrahierung von konkreten Systemtypen

565
4 Message Transformation Model (MTM)

Aufbauend auf den Anforderungen soll nun das Message Transformation Model (MTM), als ein konzeptuelles Modell für komplexe Nachrichtentransformationen, definiert werden.

4.1 Einordnung des Modells


Abbildung 2: Adaptierte Drei-Schichten-Architektur

1. Die **externe Ebene** umfasst auf der einen Seite die unterschiedlichen Abbildungen von Nachrichten und damit auch von Daten. Auf der anderen Seite schließt sie aber ebenfalls die verschiedenen Prozessbeschreibungssprachen ein. Somit stellt diese Ebene die Sicht eines Nutzers auf Nachrichten und Transformationsprozesse dar.
2. Im Gegensatz zu der externen Ebene welche als standard- und sprachorientiert einzuschätzen ist, bildet die konzeptionelle Ebene die Anforderungen komplexer Nachrichtentransformationen in Bezug auf deren statischen und dynamischen Aspekte, ab. Somit ist sie eine Generalisierung von Daten- und Prozessbeschreibungen, weshalb die Datenunabhängigkeit und die Unabhängigkeit von speziellen Prozessbeschreibungen gewährleistet ist.

3. Die interne Ebene stellt die physische Realisierung des konzeptionellen Modells dar, wobei sich wiederum unterschiedliche Realisierungsalternativen anbieten.

Das konzeptionelle Nachrichtenmodell orientiert sich dabei zunächst grundlegend an dem relationalen Datenmodell [Cod70] und dem evolutionären Molekül-Atom-Datenmodell (MAD) [HMWMS87]. Da diese Modelle jedoch ausschließlich strukturierte Daten abbilden, wurden diese um Konzepte von Modellen zur Abbildung von semistrukturierten Daten, wie beispielsweise des Object Exchange Model (OEM) [PGMW95] und des “Yet Another Tree-based Data Model” (YAT) [CDS98], sowie des XML-Datenmodells angereichert. In Analogie zum Nachrichtenmodell lehnt sich auch das konzeptionelle Prozessmodell an das Relationenmodell an, und bedient sich der Mengenoperationen und relationalen Operationen. Da in einem Transformationsprozess jedoch auch die Beschreibbarkeit von Kontrollflüssen gewährleistet sein muss, reichen die Mittel des Relationenmodells nicht aus. So werden diese um Semantiken des ebenfalls mathematisch fundierten Petri-Netz-Modells, sowie aktueller Prozessbeschreibungssprachen wie WS-BPEL [OAS06] erweitert.

4.2 Definition des konzeptionellen Nachrichtenmodells


Der logische Aufbau des Kopfsegments und des Datensegmentes, welcher in Abbildung 3.b) dargestellt ist, lehnt sich stark an das relationale Modell an, weist jedoch erweiterte Aspekte auf. So setzt sich das Kopfsegment aus k Name-Wert-Paaren zusammen.
Das Datensegment hingegen besteht aus einer logischen Tabelle mit m Attributen und n Tupeln. Hierbei können die Attribute sowohl atomare Typen aufweisen, als auch wiederum selbst ein Datensegment und damit eine "Nested Table" darstellen.

Abbildung 3: Aufbau des konzeptuellen Nachrichtenmodells


Definition 4.1: Message
Sei \( M \) ein Nachrichtentyp der sich mit \( M = (H, D) \) aus einem Kopfsegmenttyp \( H \) und einem Datensegmenttyp \( D \) beschreibt so ist eine Nachricht \( m \) mit \( m \subseteq M \) definiert. Weiterhin sei ein Kopfsegment mit \( h = \{a_1, \ldots, a_k\} \) als Menge von elementaren Attributen mit \( k > 0 \) und ein Datensegmenttyp mit \( D = A_1 \times \ldots \times A_l \) als Menge von Attributtypen mit \( l > 0 \) definiert, wobei der Attributtyp \( A_i \) entweder atomar ist oder mit \( A_i \subseteq D \) ein Datensegment abbilden kann. So wird definiert, dass ein Attribut \( a_i : D \rightarrow A_i \) die Abbildung eines Datensegmenttyps auf einen Attributtyp ist. Außerdem wird für alle Tupel eines Datensegmentes \( \forall t \in d : t[a_i] \equiv t[i] \) definiert.

Ein weiterer wesentlicher Aspekt des konzeptuellen Nachrichtenmodells ist die Überführung der externen Datenrepräsentationen in dieses Modell. Während die Abbildung von relationalen Daten direkt auf das Modell angewandt werden kann, ist dies bei der Überführung von XML ungleich komplexer. Prinzipiell wird hierbei, in Anlehnung an SQL:2003 Part 14: SQL und XML [Mel03], eine Element-orientierte, genauer eine strukturorientierte, Zerlegung, als ein spezieller Ansatz des so genannten XML-Schreddings, avisiert. Dabei erfolgt die Zerlegung generisch auf Grundlage selbstdefinierter, strukturspezifischer Regeln und damit unabhängig vom konkreten Inhalt. Intern wird somit ein attributorientierter, feingranularer Ansatz verfolgt, bei dem Nachrichten bis auf die atomaren Attribute zu zerlegen sind, was wiederum die Voraussetzung für den Direktzugriff auf einzelne Attribute ist. Neben dem vorgestellten Ansatz bilden

4.3 Definition des konzeptuellen Prozessmodells


Abbildung 4: Entwurfsdimensionen des konzeptuellen Prozessmodells

1. In Bezug auf die strukturelle Art des Prozessmodells wird das graphenorientierte Prozessmodell verwendet. Allerdings wird mit dem "Prozess" ebenfalls ein hierarchisches Element integriert. Dieser Ansatz, geht davon aus, dass es ausschließlich elementare Prozessschritte gibt, welche auch als Knoten bezeichnet werden. Diese Knoten sind dann mit beliebig gerichteten Kanten verbunden, um den Prozessab-


Abbildung 5: Grundmodell “Gerichteter Graph“ (in Anlehnung an [JBo06])

**Definition 4.2: Prozess**

Ein Prozesstyp $P^x$ definiert sich mit $P^x = (N^x, S^x, F^x)$ als eine 3-Tupel-Darstellung eines gerichteten Graphen, wobei $N^x$ mit $N^x = \{n_1^x, \ldots, n_k^x\}$ und $k > 0$ eine Menge von Knoten, $S^x$ mit $S^x = \{s_1^x, \ldots, s_l^x\}$, $l > 0$ und $s_i^x = \{o_1^x, \ldots, o_m^x\}$ mit $m > 0$ eine Menge von Diensten, samt deren jeweiligen Operationen, und $F^x$ mit $F^x \subseteq (N^x \times S^x)$ eine Menge von Flussrelationen darstellt. $P^x$ ist dabei mit $P^x \subseteq N^y$ gleichzeitig ein Knotentyp. Ein Prozess $p^x$ mit $p^x \subseteq P^x$ weist hierbei einen bestimmten Zustand $z(p^x)$ mit $z(p^x) = \{z(n_1^x), \ldots, z(n_k^x)\}$ auf. Der Prozesszustand ergibt sich also aus der Gesamtheit der einzelnen Knotenzustände $z(n_i^x)$ mit $z(n_i^x) = \{M[i]\}$ und $(M[i] = \neg \emptyset) \lor (M[i] = \emptyset))$.


Aufbauend auf dem Grundmodell “Directed Graph” wird nun das anforderungsorientierte Prozessmodell definiert. Hierbei werden spezifische Operatoren als spezialisierte Prozessschritte und damit als Knotentypen definiert, welche so angelegt sind, dass sie gleichermaßen den Kontrollfluss (b), Datenfluss (c), sowie die Interaktion mit externen Systemen (a) abbilden und dabei einen möglichst redundanzfreien Charakter aufweisen.

**a) Interaktionsorientierte Operatoren**


<table>
<thead>
<tr>
<th>Name</th>
<th>Allgemeine Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke</td>
<td>Senden/Empfangen einer Nachricht an eine/von einer Operation eines hinreichend abstrahierten Dienstes</td>
</tr>
<tr>
<td>Receive</td>
<td>Empfangen einer Nachricht von der aufrufenden Stelle</td>
</tr>
<tr>
<td>Reply</td>
<td>Senden einer Ergebnisnachricht an die aufrufende Stelle</td>
</tr>
</tbody>
</table>

Tabelle 1: Interaktionsorientierte Operatoren des konzeptuellen Prozessmodells
b) Kontrollflussorientierte Operatoren
Mit der Spezifikation dieser Operatoren werden Spezialfälle der Abbildung des Kontrollflusses adressiert, da einfache Abfolgen und Muster bereits mit dem Grundmodell “Directed Graph” spezifiziert wurden. Da eine kompilierte Repräsentation avisiert wurde, werden diese Operatoren intern als einfache Kontrollstrukturen einer prozeduralen Programmiersprache verwendet.

<table>
<thead>
<tr>
<th>Name</th>
<th>Allgemeine Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch</td>
<td>Auswahl von ausgehenden Knoten entsprechend inhaltsbasierter Bedingungen (Alternative)</td>
</tr>
<tr>
<td>Fork</td>
<td>Start einer parallelen Verarbeitung und Weitergabe der eingehenden Nachrichten</td>
</tr>
<tr>
<td>Delay</td>
<td>Verzögerung der Verarbeitung bis zu einem Zeitpunkt beziehungsweise für eine Zeitspanne</td>
</tr>
<tr>
<td>Signal</td>
<td>Initiierung eines Signals, worauf kontrolliert mit einem SignalHandler reagiert werden kann</td>
</tr>
</tbody>
</table>

Tabelle 2: Kontrollflussorientierte Operatoren des konzeptuellen Prozessmodells

c) Datenflussorientierte Operatoren

<table>
<thead>
<tr>
<th>Name</th>
<th>Allgemeine Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign</td>
<td>Einfache Wertzuweisungen in Form von elementaren oder komplexen Objekten (XPath als Anfragesprache)</td>
</tr>
<tr>
<td>Translation</td>
<td>Durchführung elementarer Transformationen mittels XML-Transformationssprachen (XSLT, STX, ...)</td>
</tr>
<tr>
<td>Selection</td>
<td>Auswahl von Tupeln entsprechend einer Selektionsbedingung</td>
</tr>
<tr>
<td>Projection</td>
<td>Auswahl von Attributen entsprechend einer Attributliste</td>
</tr>
<tr>
<td>Join</td>
<td>Verbund von Daten mehrerer Nachrichten entsprechend einer Verbindungsbefüllung und eines Verbindungstyps</td>
</tr>
<tr>
<td>Setoperation</td>
<td>Anwendung der Mengenoperationen Vereinigung, Durchschnitt und Differenz auf eine Nachrichtenmenge</td>
</tr>
<tr>
<td>Split</td>
<td>Zerlegung einer großen Nachricht in mehrere kleine Nachrichten</td>
</tr>
<tr>
<td>OrderBy</td>
<td>Sortierung einer Nachrichtenmenge entsprechend eines Attributes</td>
</tr>
<tr>
<td>Validate</td>
<td>Validierung von Nachrichten entsprechend einer Prüfbedingung</td>
</tr>
<tr>
<td>Action</td>
<td>Ausführen einer beliebigen Java-Klasse (Erweiterbarkeit)</td>
</tr>
</tbody>
</table>

Tabelle 3: Datenflussorientierte Operatoren des konzeptuellen Prozessmodells

An dieser Stelle wird auf eine detaillierte Erläuterung der einzelnen Operatoren verzichtet und lediglich auf die Arbeit [Böh06] und die formale Beschreibung im Anhang A verwiesen.
5 Beispielszenario “ETL-Prozess“

In diesem Abschnitt wird das, in der Einleitung eingeführte, Beispielszenario konkretisiert und daran exemplarisch die Modellierung mit dem MTM auf konzeptueller Ebene veranschaulicht.

Abbildung 6: MTM-Prozesse bs_process2 und bs_process3

Initiiert durch Geschäftsvorfälle im SAP R/3-System, werden Daten der X GmbH, mittels des IDoc-Formats [SAP06], unmittelbar in die konsolidierte Datenbank eingebracht. Dabei werden die speziellen IDoc-Typen, DEBMAS05, CREAMS03, MATMAS03 und ORDERS05 aus den Modulen SD (Sales and Distribution) und MM (Materials Management) verwendet. Die Bewegungsdaten von Einkaufs- und Vertriebsprozessen der Y GmbH werden aus einem proprietären System exportiert und im Filesystem als XML-Dateien hinterlegt. Da die Y GmbH eine Menge von heterogenen Systemen unterhält, sind vor dem Einbringen der Daten in die konsolidierte Datenbank spezielle Daten aus einer CRM-Datenbank nachzuladen, welche physisch in einem MS SQL Server verwaltet wird. Das Schema der konsolidierten Datenbank (KDB) entspricht dem TPCH-Schema [Tra05], welches um diverse Flag-Attribute und Zeitstempel erweitert wurde und keinerlei Constraints enthält. Somit werden gegebenenfalls nicht-konsistente Daten und Duplikate zunächst in die KDB eingebracht. Während zwischen den Quellsystemen und der KDB sehr viele tupelorientierte Transaktionen ausgeführt werden, sind die Daten der KDB nur einmal täglich als Datenmenge in das Data Warehouse (DWH) zu übernehmen. Vor einer derartigen Übernahme ist jedoch der Prozess des DataCleansings [RD00], [MF03]

Die MTM-Transformationsprozesse, welche exemplarisch mit der Abbildung 6 dargestellt wurden, können auf externer Ebene, beispielsweise mit WSBPEL, beschrieben und dann durch eine Middleware-Plattform in die konzeptuelle respektive die interne Ebene überführt und verarbeitet werden.

6 Ausgewählte Realisierungsaspekte am Beispiel von TransConnect®

Die Realisierbarkeit des MTM wurde im Rahmen der Entwicklung von TransConnect® 2.0 an einem Prototypen nachgewiesen. TransConnect® ist eine Integrationsplattform, welche unter anderem als EAI-Server und ETL-Tool zur Anwendung kommt. Prinzipiell ist TransConnect® in die folgenden drei Teilkomponenten zu differenzieren:

- TransConnect® Manager (Präsentation)
- TransConnect® Server (Geschäftslogik)
- TransConnect® DataStore (Daten)

Im Rahmen dieses Abschnitts werden wesentliche Teilaspekte der Realisierung des MTM herausgegriffen und näher erläutert.

6.1 Entwurf des TransConnect® Servers


6.2 Überführung von Prozessbeschreibungen in das Prozessmodell


2. Im Rahmen der Ebene der internen Analyse und Optimierung wird die interne XML-Prozessbeschreibung zunächst regelbasiert analysiert und anschließend sowohl einer regelbasierten als auch einer kostenbasierten Optimierung unterzogen. Für die kostenbasierte Analyse wurden zwar bereits erste theoretische Ansätze auf Basis eines definierten Kostenmodells erarbeitet, bedürfen jedoch einer weiteren Verfeinerung, so dass diese Gegenstand zukünftiger Arbeiten sein werden.


6.3 Überführung von externen Repräsentationen in das Nachrichtenmodell


Abbildung 9: Überführung von externen Daten in das interne Format
In einem zweiten Schritt kann nun die interne, transiente Repräsentation in eine interne, persistente Repräsentation überführt werden, indem die Nachricht in den zu Grunde liegenden TransConnect\textsuperscript{\textregistered} DataStore eingebracht wird. Hierbei ergibt sich die Schwierigkeit, dass zur Entwicklungszzeit die Struktur der Nachrichten nicht bekannt ist und DDL-Transaktionen das System unnötig belasten würden, so dass eine generische Relationenstruktur zur Speicherung dieser Repräsentation notwendig ist.

Abbildung 10: Überführung des internen Formats in eine persistente Repräsentation

Da die Überführung der transienten in die persistente Repräsentation und vice versa relativ oft im Lebenszyklus eines Workflows ausgeführt wird, kommt hierbei natürlich dem effizienten Zugriff eine große Bedeutung zu. So wird das Konzept verfolgt, rekursiv für jede transiente Nachrichtenrepräsentation, ein Insert-Statement auf die Relation DataTable, sowie je einen Batch von Insert-Statements für die Relationen DataAttribute und DataValue anzulegen und diese dann zusammenhängend auszuführen, somit kann die Anzahl der physischen Datenbankanfragen in dem obigem Beispiel auf neun Anfragen reduziert werden. An dieser Stelle sei erwähnt, dass dies jedoch bei sehr großen und tief geschachtelten XML-Dokumenten ineffizient ist. Hier wäre es kostengünstiger die Nachricht "Bestellung" in der Applikation wieder in einen Zustand "unparsed" zu versetzen und diesen dann als CLOB persistent zu machen, was in lediglich drei physischen Datenbankanfragen resultieren würde. Bei einem abermaligen Zugriff müsste man dann die transiente DataTable wieder in den Zustand "parsed" versetzen, um innerhalb des Workflows gezielt lesend und schreibend auf Einzelwerte zugreifen zu können.

7 Zusammenfassung und Schlussfolgerungen


Literatur


<table>
<thead>
<tr>
<th>Name</th>
<th>Formale Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invoke</td>
<td>( \iota_{s,o}(m_1) = M { m_1, m_2 } ) mit ( i \subseteq N ), ( S_o \leftarrow m_1 ) und ( m_2 = \langle h_{new}, d \leftarrow m_{s,o} \rangle )</td>
</tr>
<tr>
<td>Receive</td>
<td>( \rho() = { m_1 } ) mit ( \rho \subseteq N ), ( p = \rho \leftarrow m_{in} ) und ( m_1 = \langle h_{new}, d \leftarrow m_{in} \rangle )</td>
</tr>
<tr>
<td>Reply</td>
<td>( \varphi(m_i) = { } ) mit ( \varphi \subseteq N ) und ( m_{out} \leftarrow m_1 )</td>
</tr>
<tr>
<td>Switch</td>
<td>( \text{SWITCH}_{\text{expr}(n)}(m_1) = { \text{expr} \wedge (m_1 \rightarrow n_i) } ) mit ( \text{SWITCH} \subseteq N ) und ( \text{expr} = { \text{expr}_1, \ldots, \text{expr}_n } ) mit ( n &gt; 0 )</td>
</tr>
<tr>
<td>Fork</td>
<td>( \text{FORK}_{n}(m_1) = \forall i(m_1 \rightarrow n_i) ) mit ( \text{FORK} \subseteq N ) und ( n = { n_1, \ldots, n_m } ) mit ( n &gt; 1 )</td>
</tr>
<tr>
<td>Delay</td>
<td>( \text{DELAY}_t(m_1) = m_1 ) mit ( \text{DELAY} \subseteq N ) und ( t = \text{timestamp} )</td>
</tr>
<tr>
<td>Signal</td>
<td>( \text{SIGNAL}<em>{n,\text{type}}(m_1) ) = ( m_2 ) mit ( \text{SIGNAL} \subseteq N ), ( m_2 = m</em>{in} ) und ( m_2.h = \text{signaltype} \rightarrow m_1.h )</td>
</tr>
<tr>
<td>Assign</td>
<td>( \iota_{\text{part}(m_1), \text{part}(m_2)}(m_1, m_2) = m_3 ) mit ( \iota \subseteq N ), ( m_3.h \equiv m_{2.h} ) und ( m_2 \rightarrow m_3 \wedge m_1.\text{part}(m_1) \rightarrow m_3.\text{part}(m_2) )</td>
</tr>
<tr>
<td>Translation</td>
<td>( \tau_{\text{file},\text{type}}(m_1) = m_2 ) mit ( \tau \subseteq N ), ( m_1 \subseteq m_2 ), ( m_2.h \equiv m_{2.h} ) und ( \text{type} = \text{XSLT} )</td>
</tr>
<tr>
<td>Selection</td>
<td>( \sigma_{\text{expr}}(m_1) = m_2 ) mit ( \sigma \subseteq N ), ( m_1.h \equiv m_{2.h} ) und ( m_2.d = { t</td>
</tr>
<tr>
<td>Projection</td>
<td>( \pi_{\beta}(m_1) = m_2 ) mit ( \pi \subseteq N ), ( m_1.h \equiv m_{2.h} ), ( \beta \subseteq D ) und ( m_2.d = { t</td>
</tr>
<tr>
<td>Join</td>
<td>( \iota_{\text{file}[i].d}(m_1[i]) = m_2 { \text{t}</td>
</tr>
<tr>
<td>Setoperation</td>
<td>( \mu_{m_1[i]} = m_2 ) mit ( \mu \subseteq N ), ( m_2.h = m_{1[0], h} ) und ( (\cup(m_1[i])) = m_2 { \text{t} \in m_1[0] } ) ( \forall (m_1[i]) = m_2 { \text{t} \in m_1[0] \wedge \text{t} \in m_{1[i]} } ) ( \forall (\neg(m_1[i])) = m_2 { \text{t} \in m_1[0] \wedge \neg \text{t} \in m_{1[i]} } )</td>
</tr>
<tr>
<td>Split</td>
<td>( \psi(m_1) = m_2 ) mit ( \psi \subseteq N ), ( \forall i(m_2[i].h \equiv m_{1.h}) ) und ( m_2[i].d = d = \emptyset )</td>
</tr>
<tr>
<td>OrderBy</td>
<td>( \omega_{\beta,\text{type}}(m_1[i]) = m_2 { m[i] \rightarrow m[i] } ) mit ( \omega \subseteq N ), ( \beta \subseteq D ), ( m_1[i] ) ( m_2[i] = \emptyset \wedge (m_2[i] \rightarrow m_1[i] = \emptyset) ) und ( \text{type} = \text{asc} ) ( \wedge \text{desc} )</td>
</tr>
<tr>
<td>Validate</td>
<td>( \nu_{\text{expr}}(m_1) = m_2 { { \text{expr} \wedge m_1 } \vee (\neg \text{expr} \wedge h_{\text{error}} \wedge d_{\text{new}}) } ) mit ( \nu \subseteq N )</td>
</tr>
<tr>
<td>Action</td>
<td>( \text{ACTION}_{e,o}(m_1) = m_2 ) mit ( \text{ACTION} \subseteq N )</td>
</tr>
</tbody>
</table>

Tabelle 4: Formale Beschreibung der Operatoren des MTM
Integrating Query-Feedback Based Statistics into Informix Dynamic Server

Alexander Behm 1, Volker Markl 2, Peter Haas 3, Keshava Murthy 4

1 Berufsakademie Stuttgart / IBM Germany
alexbehm@gmx.de

2 IBM Almaden Research Center
marklv@us.ibm.com

3 IBM Almaden Research Center
peterh@almaden.ibm.com

4 IBM Menlo Park
rkeshav@us.ibm.com

Abstract: Statistics that accurately describe the distribution of data values in the columns of relational tables are essential for effective query optimization in a database management system. Manually maintaining such statistics in the face of changing data is difficult and can lead to suboptimal query performance and high administration costs. In this paper, we describe a method and prototype implementation for automatically maintaining high quality single-column statistics, as used by the optimizer in IBM Informix Dynamic Server (IDS). Our method both refines and extends the ISOMER algorithm of Srivastava et al. for maintaining a multidimensional histogram based on query feedback (QF). Like ISOMER, our new method is based on the maximum entropy (ME) principle, and therefore incorporates information about the data distribution in a principled and consistent manner. However, because IDS only needs to maintain one-dimensional histograms, we can simplify the ISOMER algorithm in several ways, significantly speeding up performance. First, we replace the expensive STHoles data structure used by ISOMER with a simple binning scheme, using a sweep-line algorithm to determine bin boundaries. Next, we use an efficient method for incorporating new QF into the histogram; the idea is to aggregate, prior to the ME computation, those bins that do not overlap with the new feedback records. Finally, we introduce a fast pruning method to ensure that the number of bins in the frequency distribution stays below a specified upper bound. Besides refining ISOMER to deal efficiently with one-dimensional histograms, we extend previous work by combining the reactive QF approach with a proactive sampling approach. Sampling is triggered whenever (as determined from QF records) actual and estimated selectivities diverge to an unacceptably large degree. Our combined proactive/reactive approach greatly improves the robustness of the estimation mechanism, ensuring very high quality selectivity estimates for queries falling inside the range of available feedback while guaranteeing reasonably good estimates for queries outside of the range. By automatically updating statistics, query execution is improved due to better selectivity estimates, and the total cost of ownership (TCO) is reduced since the database administrator need not update statistics manually for monitored columns.
1 Introduction

In today’s database market performance and the total cost of ownership (TCO) are two important factors that provide a competitive advantage. Performance is dependent on the query optimizer’s choices of physical execution plans, which heavily rely on the accuracy of available statistics [IC91, Lyn88]. The total cost of ownership is influenced by the administration cost, among others. A superior performance and TCO will provide a significant competitive advantage. Traditionally, statistics are updated manually by the database administrator (DBA) at intervals according to vendor-recommendations, based on experience or in response to bad performance. On the one hand, this approach requires expertise and time of the DBA which may increase the total cost of administration. On the other hand, manually updating statistics at certain intervals may result in either too frequent gathering of statistics (which is expensive and may impact the processing of user queries) or too infrequent gathering of statistics, resulting in suboptimal query plans due to inaccurate selectivity estimates. So far, there exists no optimal solution to determine when to update the statistics for which tables and columns. For tackling this problem we propose a method for automatically and autonomously gathering statistics. It ensures accurate statistics because they automatically evolve with changes in the database and it minimizes the need for manual intervention by the DBA which reduces costs. The general idea of feedback based statistics as described in [SLMK01, CR94] suggests exploiting results of actual queries that run on the database, creating a feedback-loop to improve future selectivity estimates by “learning” from previous queries. Embodiments of this principle may include suggesting tables and columns for which to update statistics (e.g. based on a significant discrepancy between estimated and actual cardinalities), suggesting statistical parameters such as the number of frequent values and quantiles to gather [Beh05], and creating multi-dimensional histograms (see chapter 2), and others.

In this work we present a query-feedback based method for creating and maintaining single-column statistics in IBM Informix Dynamic Server. Our work is a specialisation of ISOMER [SHM+06] to the one dimensional-case which allows for performance improvements (see 2). We gather a sampling-based distribution for a given column and continuously refine it using QF. The process of refinement consists of eliminating inconsistencies based on a timestamp (recent information is preferred) and creating a maximum entropy (ME) cumulative frequency distribution. This distribution is consistent with all available information (QF, previous sampling information) and uniform in the absence of information (see chapter 6). In contrast to the work described in [Beh05], we do not use the ME distribution for recommending statistical parameters but make this distribution available to the optimizer directly for selectivity estimation. Moreover, this approach offers flexibility when thinking of cases in which feedback only delivers information on small regions in the column domain. During the course of time the accuracy of statistics will degrade as the data in the column changes. Hence, there is a good chance that queries not covered by feedback will yield a large discrepancy between estimated and actual cardinalities. If a significant discrepancy were to be detected one might schedule an updating of statistics via sampling. So, in between sampling procedures the QF is used to continuously refine the distribution until a said discrepancy is detected. In case feedback continuously delivers information on the whole column domain it may not be necessary to sample the column...
ever again. Apart from this flexibility, our hybrid approach is reasonable in terms of computational costs. Sampling a column is very Input/Output (I/O) intensive since the actual data is tapped. Refining an existing distribution with the said method is very CPU intensive, but very low on I/O costs because only a very low number of records have to be read (the feedback records). As a rule of thumb, in terms of total costs I/O is nine times more expensive than CPU cycles. Therefore, the refinement process is less expensive by orders of magnitude and at the same time a lot is to be gained due to better execution plans.

2 Related Work

Single and multi-dimensional statistics are traditionally created proactively using techniques involving data-scans as described in [PI97, GKVD00, DGR01, MD88, TGIK02] or sampling techniques as in [CMN98]. On the one hand these approaches are costly and scale poorly with growing table sizes or require large sample sizes, respectively. On the other hand these approaches disregard the fact that a user’s workload may not require high accuracy in all parts of the domain. So, a reactive method may prove to be more feasible because learning from previous queries ensures high accuracy in often queried regions of the domain and scanning the actual data becomes obsolete or less frequently required. The idea of using QF for creating statistics needed for selectivity estimation is known since [CR94]. Various methods have been proposed to reactively improve selectivity estimates [CR94, SLMK01] but we will focus on the construction of histograms. This work specializes the method described in ISOMER [SHM+06], which consistently creates multi-dimensional histograms based on QF, to the case of one-dimensional histograms. Previous work [BCG01, PI97, AC99, LWV03] in the area of feedback based multi-dimensional histograms lacked the feature of consistently incorporating new feedback, thus requiring heuristics to determine which feedback to keep. This reduces the overall accuracy of the histogram. ISOMER is a full-blown solution for feedback based statistics that uses the STHoles datastructure [BCG01] which is superior to the one in MHist [PI97] but even so, imposes an unnecessary overhead for the case of one-dimensional histograms. The one-dimensional case allows for optimization. We use a sweep-line algorithm to determine the atoms (bins) for the ME histogram instead of iterating though all levels in the STHoles structure. In order to meet storage requirements we use a one-pass pruning algorithm which does not need to incrementally recompute the ME solution as is the case in ISOMER. Furthermore, for adding new feedback to the histogram we use an efficient refinement algorithm based on the principle of maximum entropy. In essence, we propose a QF based method for consistently creating and refining one-dimensional histograms based on the ME principle. This method is similar to ISOMER but focuses on the one dimensional case which allows for improvements (compared to ISOMER). This effort is novel in the sense that we are not aware of any previous work specializing in QF based one-dimensional histograms.
3 Background

3.1 Informix Dynamic Server (IDS)

IBM Informix Dynamic Server (IDS) is a light-weight relational database management system that focuses on online transaction processing. It uses a cost-based optimizer for choosing an efficient physical execution plan. Currently, only single-column statistics are used for selectivity estimation. These statistics are gathered proactively by sampling. So, we wish to enhance selectivity estimates by the use of QF. Since only single-column statistics are currently used we expect a big impact by improving their accuracy.

3.2 Distributions in Informix Dynamic Server

![Figure 1: Single-column, sampling-based distribution in IDS]

3.3 Updating Statistics in Informix Dynamic Server

Apart from data distributions IDS gathers other statistical information on tables, columns and indexes which are useful to the IDS optimizer when choosing an execution plan. The “UPDATE STATISTICS” command is used to update statistics. This command offers three basic modes of operation: UPDATE STATISTICS [LOW—MEDIUM—HIGH]. We add a new mode of UPDATE STATISTICS for enabling the feedback based updating of statistics described in this work:
FBS (feedback-based statistics)

1. monitor queries on specified column gathering estimated and actual cardinalities
2. create/refine maximum-entropy distribution at given condition (e.g. update-delete-insert counter or the number of feedback records reach a certain threshold)
3. use maximum-entropy distribution for selectivity estimation in optimizer

3.4 Representing Feedback/Distribution Information

In order to ultimately achieve a ME frequency distribution we need to define a common way of representing available information on the respective column. The information retrieved from the Query-Feedback Warehouse (QFW), from the sampling-based distribution and from an existing ME distribution is represented as a set of triples: \( I = \{(l_1, u_1, f_1), (l_2, u_2, f_2), \ldots, (l_n, u_n, f_n)\}\) where for \( n \) pieces of information \( i \in \{1, 2, 3, \ldots, n\} \)
\( l_i \) is the lower boundary of the interval, \( u_i \) is the upper boundary of the interval and \( f_i \) is the relative frequency of occurrence of values in \( (l_i, u_i) \).

For example consider the following SQL-Query:
\[
\text{SELECT COUNT(*) FROM TAB WHERE COL > A and COL \leq B}
\]
With \( |\text{TAB}| \) as the table cardinality, \( C \) as the returned count and \( F = \frac{C}{|\text{TAB}|} \) we represent the result of this query with the triple \((A, B, F)\).

Now that all information is in a standard format we can start preparing constraints for the ME distribution. Since some intervals may overlap or we may lack information on some parts of the column domain we need to find a partitioning of the column domain such that no bins in the partitioning overlap. We refer to “bins” as mutually disjoint intervals that partition the whole column domain.

Consider the following example:
Let there be a set of triples \( I = \{(l_1, u_1, f_1), (l_2, u_2, f_2), (l_3, u_3, f_3)\}\) which we can represent graphically by the following diagram:

![Graphical representation of a set of triples and resulting bins](image)

Figure 2: Graphical representation of a set of triples and resulting bins
The set of non-overlapping triples \( \{r_1, r_2, r_3, r_4\} \) is the refinement of the set of overlapping triples. It is the smallest set of mutually disjoint intervals, which we call “bins”, such that each interval in \( I \) can be expressed as a finite union of bins. We distinguish non-overlapping triples from overlapping triples by representing each non-overlapping one as triple of the form \((a, b, m)\) equivalent to \((l, u, f)\) for intervals. We denote the set of bins by \( R = \{r_1 = (a_1, b_1, m_1), r_2 = (a_2, b_2, m_2), ..., r_k = (a_k, b_k, m_k)\} \), where for \( k \) number of bins \( i \in \{1, 2, 3, ..., k\} \) \( a_i \) is the lower boundary of the bin, \( b_i \) is the upper boundary of the bin and \( m_i \) is the relative frequency of occurrence of column-values in \((a_i, b_i]\).

Looking at Figure 2 one can easily derive a system of linear equations describing the frequencies of our intervals as a sum of frequencies of bins. In this example the system of linear equations looks like:

\[
\begin{align*}
    f_1 &= m_1 + m_2 + m_3 + m_4 \\
    f_2 &= m_2 + m_3 \\
    f_3 &= m_3 + m_4
\end{align*}
\]

We call the set of linear equations “constraints” and denote the set of constraints by \( T = \{t_1, t_2, t_3, ..., t_n\} \), each element in \( T \) corresponding to a linear equation of the form \( f_n = \delta_1 m_1 + \delta_2 m_2 + \delta_2 m_3 + ... + \delta_k m_k \) with \( \delta_1, ..., \delta_k \in \{0, 1\} \) Note that for a relative frequency distribution all frequencies must add up to one. This is why we always add the interval \((l_1, u_1, 1.0)\) to \( I \), where \( l_1 \) is the minimum value in the column domain and \( u_1 \) is the maximum value in the column domain. The resulting constraint we add to \( T \) looks like this: \( f_1 = m_1 + m_2 + m_3 + ... + m_k = 1.0 \).

### 3.5 The Maximum Entropy Principle

Our knowledge on a given column as presented in the previous sections may, on the one hand, be incomplete due to lack of information in some parts of the column domain and, on the other hand, some intervals may overlap. We seek to find a set of \( k \) bins with frequencies \( \{m_0, m_1, ..., m_k\} \) that are consistent with all available information. There exist many sets of bins satisfying all constraints because frequencies of bins on which no information is present can be arbitrarily chosen (as long as the sum of all frequencies equals one). However, we seek exactly the set of bins which is uniform in the absence of information. The principle of maximum entropy delivers a solution to this problem:

“Model everything that is known and assume nothing about the unknown.”
[GS85]

Previous work using this principle includes [MMK+05] in which consistent estimates for selectivities of conjunts of predicates are created in the presence of incomplete, overlapping knowledge. ISOMER [SHM+06] uses the principle to consolidate information retrieved from QF. The latter applies to this work except that we focus on the one-dimensional case. Hence, the entropy we need to maximize is similar to the one in ISOMER.
ISOMER defines the following entropy:

\[ H(R) = - \sum_{i=1}^{k} m_i \ln \left( \frac{m_i}{h_i} \right) \]  

(1)

where \( h_i \) denotes the volume of bin (bucket) \( i \)

Since we focus on the one-dimensional case \( h_i \) is simply the length of bin \( i \) \((u_i - l_i\) for integers).

3.6 Inconsistency and Implied Zero Removal

**Inconsistency Removal:** Query feedback is gathered over time and hence is a time series of snapshots giving insight into the actual distribution of data within a given column at a given time. Changes to the respective column occur all the time due to normal database usage. This is the reason why inconsistencies between feedback records or between feedback records and an existing distribution occur. The ME principle cannot be applied to an inconsistent system because an optimal solution does not exist. The problem of inconsistency elimination has been addressed in [KHM+06, MHK+06] and we follow the linear programming solution approach described therein.

**Implied Zero Removal:** As recognized in [KHM+06, MHK+06] bins having an implied frequency of zero pose a threat to solving the ME problem efficiently. The iterative scaling algorithm will try to push the frequency of said bins to zero requiring a high number of expensive iterations to converge, assigning a frequency unequal to zero to these bins (the logarithm of zero is undefined). The latter introduces an estimation error which may exceed acceptable limits if many implied zeros exist. In order to ensure that the iterative scaling algorithm converges fast and to eliminate the said error we need to detect and remove implied zero bins. For removing implied zero bins we pursue the approximate zero detection approach described in [KHM+06, MHK+06].

3.7 The Iterative Scaling Algorithm

The constrained optimization problem induced by the ME principle can be numerically approximated efficiently using the iterative scaling algorithm [DR72] or variations thereof [MMK+05]. The algorithm determines the bin frequencies without violating the constraints in \( T \) while adhering to the ME principle.
4 Feedback-Based Statistics Maintenance in IDS

4.1 Overview: Maintaining Statistics

Figure 3: Informix Dynamic Server architecture for automatically maintaining statistics

<table>
<thead>
<tr>
<th>Glossary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>act</td>
<td>actual cardinality</td>
</tr>
<tr>
<td>est</td>
<td>estimated cardinality</td>
</tr>
<tr>
<td>num_qf</td>
<td>number of query feedback records for a given column</td>
</tr>
<tr>
<td>UDI</td>
<td>Update, Delete, Insert counter for a given column</td>
</tr>
<tr>
<td>DBA</td>
<td>database administrator</td>
</tr>
<tr>
<td>QFW</td>
<td>query feedback warehouse, stores estimated and actual cardinalities of intermediate results of queries</td>
</tr>
</tbody>
</table>

4.2 Determining the Bin Boundaries

A sweep-line algorithm as described in [JP82] is an efficient method for intersecting geometric figures. We use our own, specific implementation for determining the boundaries and lengths of each bin constructing the constraint matrix on-the-fly. The constraint matrix stores the system of linear equations column-wise and is an internal representation for the constraints. A detailed description of the sweep-line algorithm used in this work can be found in [Beh05]. In summary this is how the sweep-line algorithm works: First we need
the set of intervals sorted by lower boundary. The sweep-line traverses from lower boundary to lower boundary. In every iteration (or “sweep-event”) the current interval is added to the “sweep-event structure”. In our case the sweep-event structure is a heap having the interval with the lowest upper boundary as head. Now we distinguish several cases for finding the bins. Each time a bin is found the constraint matrix is updated accordingly. An interval is deleted from the sweep-event structure only if its upper boundary has been used to add a bin.

Keep in mind that we are ultimately interested in the frequencies of all bins. The sweep-line algorithm performs the first step to achieving this, namely finding the boundaries and length of each bin. The frequencies are still to be determined up to this point.

Figure 4: Example of sweep line algorithm showing iterations

Figure 4 shows a set of intervals \( \{(l_1, u_1, f_1), (l_2, u_2, f_2), (l_2, u_2, f_2)\} \) and a set of bins \( \{r_1, r_2, r_3, r_4, r_5\} \) whose boundaries and lengths are to be determined by the sweep-line algorithm. I, II and III denote the iterations (or sweep-events) of the algorithm. During these iterations bins \( \{r_1, r_2, r_3\} \) are recognized. Afterwards, all intervals are on the sweep-event structure (heap). The heap is “cleaned” and in this process bins \( \{r_4, r_5\} \) are found.

The difference between the “cleaning” and the normal iterations is that the “cleaning” recognizes bins that have an upper boundary of an interval as lower boundary. Normal iterations recognize bins having a lower boundary that is a lower boundary of an interval. Additionally, the sweep-line algorithm constructs the following matrix that represents our constraints (the system of linear equations):

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m_1 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( m_2 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( m_3 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( m_4 )</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( m_5 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5: Constraint matrix resulting from intervals in figure 4
4.3 Determining the Frequencies with Maximum Entropy

Resolving Conflicts: Consider the example given in figure 4. Assume that the length of interval 1 and 2 are the same. Further, let $f_2 = 0.4$ and $f_3 = 0.2$. We can try to determine the frequencies $\{m_2, m_3, m_4\}$ algebraically:

$$m_2 + m_3 = 0.4 \quad (2)$$

$$m_3 + m_4 = 0.2 \quad (3)$$

replacing $m_3$ in equation 2 with $0.2 - m_4$ yields:

$$m_2 - m_4 = 0.2 \quad (4)$$

We see there are many possibilities to assign frequencies to the bins without violating any known information. However, we do not want to imply any knowledge and hence we seek the set of frequencies which does not violate any information, but is as uniform as possible. In other words, we seek the frequencies which maximize the entropy, namely $m_2 = 0.3, m_3 = 0.1, m_4 = 0.1$.

Filling Gaps: On some parts of the column domain we may not have any information because there is no QF covering them. As seen in figure 4 we have no QF giving insight into the frequencies $\{m_1, m_5\}$. Again, we use the principle of maximum entropy to assign frequencies to these “gaps”. Assuming that the lengths of bins $\{r_1, r_5\}$ are identical, we can easily compute the frequencies having the maximum entropy:

$$m_1 + 0.3 + 0.1 + 0.1 + m_5 = 1 \quad (5)$$

using the principle of maximum entropy we conclude $m_1 = m_5 = 0.25$.

The iterative scaling algorithm will deliver exactly the desired solution.

4.4 Meeting a Space Budget - Pruning

The number of bins in a distribution grows as the distribution continuously is refined with new feedback information. Since we cannot spend an infinite amount of memory for a distribution and the costs for refining a distribution grow exponentially with an increasing number of intervals (see section 5.1) we seek a method for reducing the number of bins. In general, the pruning mechanism could be based either on a tolerable error or based on a predefined maximum number of bins. We choose to pursue the latter approach since it requires less implementation effort and allows for precise control over storage requirements.
Let $\text{max}$ be the maximum number of bins for a column and $\text{err}(r_x, r_{x+1})$ be the resulting error when merging bins $x$ and $x + 1$. Consider the following example:

![Figure 6: Set of bins before pruning](image)

$h_1 = 4 \quad m_1 = 0.2$
$h_2 = 2 \quad m_2 = 0.1$
$h_3 = 3 \quad m_3 = 0.1$
$h_4 = 2 \quad m_4 = 0.2$
$h_5 = 3 \quad m_5 = 0.3$
$h_6 = 4 \quad m_6 = 0.05$
$h_7 = 3 \quad m_7 = 0.05$

$max = 5 \quad k = 7$

We must choose $k - \text{max} = 2$ bin pairs to merge. Of course, we wish to take the pairs which minimize the resulting error which we estimate by:

$$\text{err}(r_x, r_{x+1}) = h_x \left| \frac{m_x}{h_x} - \left( \frac{m_x + m_{x+1}}{h_x + h_{x+1}} \right) \right| + h_{x+1} \left| \frac{m_{x+1}}{h_{x+1}} - \left( \frac{m_x + m_{x+1}}{h_x + h_{x+1}} \right) \right|$$  \hspace{1cm} (6)

The error equals zero if and only if two bins imply uniformity.

In this example merging bins 1,2 and 4,5 minimizes the error because:

- $\text{err}(r_1, r_2) = 4(\left| \frac{0.2}{4} - \frac{0.2+0.1}{4+2} \right|) + 2(\left| \frac{0.2}{4} - \frac{0.2+0.1}{4+2} \right|) = 0$
- $\text{err}(r_4, r_5) = 2(\left| \frac{0.3}{3} - \frac{0.2+0.3}{2+3} \right|) + 3(\left| \frac{0.3}{3} - \frac{0.2+0.3}{2+3} \right|) = 0$
4.5 Adding Feedback to the ME Distribution Efficiently

Assume there is an existing ME distribution with \( k \) bins that we wish to refine with \( n \) QF records. The straightforward way to achieve this is to start the process “from scratch” with \( n + k \) intervals (see figure 7). However, since the cost of the process grows exponentially with an increasing number of intervals (see 5.1) a more efficient way for refinement is desirable. An important observation is that in real-life scenarios feedback covers only some parts of the column domain. Thus, some parts of the existing maximum entropy distribution are “independent” of the new feedback (they are not really independent but we can apply the principle of maximum entropy in a simpler fashion, see below). We exploit this to reduce the total number of intervals, saving costs.

![Figure 7: Differences between creating a ME distribution from scratch and refining an existing one](image)

![Figure 8: Legend for efficient refinement illustrations](image)

![Figure 9: Affected parts of ME distribution during refinement](image)

**Aggregating Bins:** In figure 9 we see that some parts of the existing ME distribution do not overlap with any new QF interval. We say these bins are “not directly affected” by the new QF. To reduce the total number of intervals we can aggregate not directly affected bins as follows:
Instead of the original bins we use the aggregated ones in the refinement process (sweep-line, inconsistency+zero removal, iterative scaling). So, in total we have the QF intervals, the directly affected bins and the aggregated bins as intervals in the refinement process.

**Restoring the Original Bins:** After having run the refinement process we need to replace the aggregated bins with the original ones. Note that this step takes place before the pruning. The new maximum entropy distribution looks as follows:

We now replace the aggregated bins with the corresponding original bins. If the frequencies of the aggregated bins are not changed during the inconsistency removal we are done at this point. Otherwise, we must compensate for the changes made to the aggregated bins by modifying the frequencies of the original bins, respectively. Note that the linear program (section 3.6) will “randomly” adjust the frequencies to achieve consistency. Since we have no further information from QF we follow the principle of maximum entropy and distribute the change in frequency (equation 7) proportionately among the original, unaffected bins.

Let $G^o$ denote the set of all aggregated bins with unmodified frequencies (after the step “aggregate bins”), $G^n$ denote the set of all aggregated bins with modified frequencies (after the step “iterative scaling”), $O(Y)$ denote the set of all original bins that aggregated bin $Y$ consists of, $F^n(X)$ denote the frequency of aggregated bin $X \in G^n$, $F^n(X)$ denote the frequency of aggregated bin $X \in G^n$ and $F(X)$ denote the frequency of original bin (not aggregated) $X \in O(Y)$. Then the total change in frequency we need to distribute
proportionately among the original, unaffected bins is given by:

\[ \Delta F = \sum_{i \in G^n} F^n(i) - \sum_{j \in G^o} F^o(j) \]  \hspace{2cm} (7)

Now, for each aggregated bin \( Y \in G^o \), we must adjust the frequencies of each original bin \( X \) that aggregated bin \( Y \) consists of, according to the following equation:

\[ F_{\text{new}}(X) = F(X) + \frac{F(X)}{\sum_{j \in G^o} F^o(j)} \Delta F \]  \hspace{2cm} (8)

for all \( X \in O(Y) \) and all \( Y \in G^o \).

5 Measurements

5.1 Performance

In this experiment we want to examine the performance of creating a maximum entropy distribution using the already introduced method with an increasing number of intervals. In order to identify bottlenecks the performance of each step is measured separately. Since the size of the two linear programs and the complexity of the iterative scaling algorithm are directly related to the number of intervals we expect an exponential increase in time needed.

Test settings:
- maximum number of bins 200
- column domain 20000 rows, [0, 10000], skewed
- feedback range [0, 10000] randomly generated intervals

![Stacked bar chart, performance with an increase in intervals](image)

We see the costs for the sweep-line and pruning are negligible. As expected the increase
in needed time is exponential. Having a more detailed look reveals that the bottleneck is the linear program for detecting implied zeros being outperformed even by the iterative scaling algorithm. This leaves room for future improvement because we use the generic method of solving a linear program, namely the simplex algorithm.

5.2 Quality

In this chapter we wish to analyse how the maximum entropy distribution compares to the actual data distribution with an increasing number of feedback records. We use the Kolmogorov Distance\(^1\) and the average distance\(^2\) over the cumulated distributions to evaluate the accuracy of a given ME distribution. With an increasing number of intervals we expect both distances to converge but not actually reach zero because of fixing the maximum number of bins.

<table>
<thead>
<tr>
<th>Test settings:</th>
<th>maximum number of bins</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>column domain</td>
<td>20000 rows, [0, 10000], skewed</td>
<td></td>
</tr>
<tr>
<td>feedback range</td>
<td>[0, 10000] randomly generated intervals</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram](image.png)

Figure 13: Accuracy of maximum entropy distribution with increasing intervals

Both curves converge towards zero quickly with an increasing number of intervals. Considering that the performance is acceptable for up to 500 intervals and the maximum error is below 2% at approx. 100 intervals we see that the feasible region in terms of cost/benefit lies within 100 and 500 intervals. However, keep in mind that the generated feedback delivers information on the whole column domain. This is the best-case scenario for feedback based statistics and unlikely to happen in a real-life environment. The next experiment shall give insight into how the Kolmogorov-Distance and the average distance

\(^1\)Maximum absolute distance of two cumulative frequency distributions
\(^2\)Average distance of two cumulative frequency distributions

596
behave when feedback allows for only limited insight into the column domain.

<table>
<thead>
<tr>
<th>maximum number of bins</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of intervals</td>
<td>100</td>
</tr>
</tbody>
</table>

Test settings:
- column domain: 20000 rows, [0, 10000], skewed
- feedback range: varying, randomly generated intervals covering percentage of column domain

Figure 14: Distances with QF covering increasing percentages of column domain

Here we can observe a clear trend. The distances improve continuously with feedback covering more and more of the column domain. The improvement is almost proportional to the percentage covered. The accuracy of the maximum entropy distribution is regulated by the QF, especially how much of the column domain is covered by QF. Do keep in mind, though, that this distribution is completely feedback based. The bad accuracy with low percentages of the column domain covered by QF is the main reason why we combine a sampling-based distribution with the feedback.

6 Conclusion

6.1 Simplifying ISOMER

ISOMER [SHM+06] is a full-blown solution for creating and maintaining multi-dimensional histograms with the use of QF. In principle, our work is a simplification of ISOMER to the one-dimensional case. This allows for optimization. Here is an overview of the differences of this work to the solution in ISOMER:
**Combining proactive and reactive methods:** In contrast to ISOMER our method does not solely rely on feedback for histogram construction. We start with a sampling based distribution which we continuously refine using feedback. If at any given time the histogram delivers poor estimates (e.g. when queries tend not to query the same portion of the data) we can update statistics the conventional way - by sampling. So, we combine the proactive and the reactive approach in a consistent manner.

**Datastructure:** ISOMER uses the STHoles datastructure [BCG01]. For a one-dimensional histogram this structure imposes an unnecessary overhead when estimating selectivities. For a given predicate the STHoles structure requires summing up partial selectivities in possibly all levels of the tree. In contrast to STHoles we store the histogram as an array with cumulated selectivities, so selectivity estimation requires less computation.

**Determining the atoms:** In order to cope with overlapping knowledge the STHoles datastructure “drills” holes into the histogram creating a tree-structure. In the one-dimensional case this “drilling” becomes unnecessarily expensive especially when several feedback records are to be added at the same time. This is because the STHoles datastructure would require each feedback record to be added to the histogram consecutively, requiring the splitting of bins and the drilling of new holes each time. Since we assume that several records are added to the histogram at the same time we need to overcome this drawback (the refinement of the histogram occurs periodically). We use an efficient sweep-line algorithm to determine the bins (atoms) in one pass (see section 4.2).

**Efficient refinement with feedback:** For efficiently incorporating new feedback into a histogram ISOMER incrementally computes the new ME distribution. However, the method in ISOMER only speeds up the iterative scaling. In this work we introduce an efficient algorithm that speeds up all parts of the process, including the inconsistency removal, zero elimination and iterative scaling. This is specific to the one-dimensional case.

**Meeting a space budget:** ISOMER exploits the fact that the ME computation delivers an “importance measure” for each bucket. So, when the number of buckets exceeds an acceptable limit ISOMER removes the bucket with the least importance, then incrementally computes then new ME solution and repeats this process until the number of buckets is acceptable. In this work we propose a fast pruning algorithm that merges bins in order to fulfill storage constraints and minimizes the impact on accuracy in one pass. Here, no recomputation of the ME solution is required.

### 6.2 Summary

Feedback based statistics are a low-cost alternative to statistics created proactively using either data scans or sampling. We focus the ideas in ISOMER to single-column statistics combining the feedback-based approach with sampling-based statistics. This is specific to the one-dimensional case because creating multidimensional statistics via sampling is very
expensive and hence not desirable in ISOMER. Further, we have shown, for the single-
column case, that purely feedback based histograms can yield acceptable accuracies at
relatively low costs even if the whole column domain is not covered by QF. This is mainly
due to the features of consistently adding new feedback information while making no as-
sumptions about the unknown. We employ well founded concepts such as an efficient
sweep-line algorithm for information preparation, linear programming for inconsistency
removal and the principle of maximum entropy for ensuring uniformity in the absence of
information. Furthermore, in order to be compliant with storage constraints a fast pruning
algorithm is used that minimizes the impact on accuracy. New feedback is added quickly
with an efficient refinement method that reduces the total number of intervals. Further
improvements can be made on the implied zero detection because it is the performance bottlenecks,
especially with a large number of intervals. Having a predefined rule set (when
exactly to trigger refinement) the database management system can autonomously refine
feedback based histograms when new feedback is generated by normal database activity.
The optimal parameters for these rules are still to be determined, e.g. referring to figure 3
when do we consider “act << est” or “act >> est”. In essence, we have specialized the
ideas in ISOMER to the one-dimensional case and introducing an industrial-strength, auto-
matic and autonomous method for feedback based single-column statistics that is founded
on well known concepts and delivers accurate selectivity estimates for either previously
seen or unseen queries at reasonable costs.

References

[AC99] A. Aboulnaga and S. Chaudhuri. Self-Tuning Histograms: Building Histograms With-
out Looking at Data. In Proceedings of the 1999 ACM SIGMOD International Confer-
ence on Management of Data, 1999.

[BCG01] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: A Multi-Dimensional Workload-
Aware Histogram. In Proceedings of the 2001 ACM SIGMOD International Confer-

[Beh05] Alexander Behm. DB2 Learning Optimizer, Query Feedback, Frequent Values and


[CR94] C. M. Chen and N. Roussopoulos. Adaptive Selectivity Estimation Using Query Feed-
back. In Proceedings of the 1994 ACM SIGMOD International Conference on Man-
agement of Data, 1994.

[DGR01] A. Deshpande, M. Garofalakis, and R. Rastogi. Independence Is Good: Dependency-
Based Histogram Synopses for High-Dimensional Data. In Proceedings of the 2001


Demonstrationen
HiSbase: Informationsfusion in P2P Netzwerken

Tobias Scholl  Bernhard Bauer  Richard Kuntschke  Daniel Weber  
Angelika Reiser  Alfons Kemper 

Technische Universität München  
{scholl, bauerb, kuntschk, weberd, reiser, kemper}@in.tum.de

1 Einleitung und Motivation


Diese Arbeiten sind Teil des AstroGrid-D Projekts in der D-Grid Initiative und werden durch das Bundesministerium für Bildung und Forschung (BMBF) unter Vertrag 01AK804F und durch Microsoft Research Cambridge (MSRC) unter Vertrag 2005-041 gefördert.

602
2 HiSbase-Architektur

Die DHT-Struktur *Pastry* [RD01] verwaltet die HiSbase-Stationen und wickelt den Nachrichtenaustausch im Overlay-Netzwerk ab. Wie in Chord [SMK01] verteilt Pastry die Daten gleichmäßig auf einen eindimensionalen, ringförmigen Schlüsselraum. Im Vergleich zu anderen DHT-Systemen ([SMK01, RFH01]) optimiert Pastry das Routing. In den ersten Phasen des Routings senden Peers bevorzugt Nachrichten zunächst an physische Nachbarn, was die Kommunikation über das Overlay-Netzwerk beschleunigt.

2.1 Einspeisung der Daten


2.2 Mehrdimensionale Daten und Raumfüllende Kurven

In vielen Bereichen der Wissenschaft werden mehrdimensionale Daten verwendet, z.B. in der Klimaforschung, in der Medizin und besonders in der Astronomie. Objekte aus Beobachtungsdaten, die eine benachbarte Position im astronomischen Koordinatensystem haben, werden hier als logisch benachbart betrachtet. Gängig ist die Verwendung des sphärischen Koordinatensystems, das die Koordinaten in Rektaszension und Deklination bezogen zum Erdmittelpunkt angibt. Abbildung 1 zeigt das Vorgehen in HiSbase an einem kleinen vereinfachten Beispiel. In Abbildung 1(i) sind die Datenobjekte des zweidimensionalen Datenraums als gelbe Punkte dargestellt.

Um bei der Abbildung auf den eindimensionalen Datenring die Datenlokaltät zu erhalten, verwendet HiSbase *raumfüllende Kurven* wie die Hilbertkurve [Hil91] oder die Z-Ordnung [OM84]. Raumfüllende Kurven wurden als Indexstruktur für mehrdimensionale Datenbanken bereits intensiv erforscht [OM84, Mar99]. Abbildung 1(ii) zeigt, wie die Bereiche des Datenraums mit Hilfe der Z-Kurve linearisiert werden.
2.3 Daten-Lastbalancierung durch Histogramme


2.4 Regionenzuordnung


2.5 Regionen-basierte Anfragen


3 Demonstrationsübersicht


604
4 Zusammenfassung und Ausblick


Literatur

1 Introduction

The deployment of data warehouses (DWs), which are integrated databases to support decision making, has become common practice in modern information technology landscapes, and the methodical design process for such databases has been advanced significantly in recent years. In this context, we are investigating issues concerning the quality of DW schemata which can be measured and algorithmically enforced via multidimensional normal forms (MNFs) during conceptual design. So far, however, tool support for DW design based on MNFs has been missing, a gap which we are about to close.

Compared to traditional database design, DW design poses several new challenges arising from the fact that DW design has to take a number of pre-existing data sources into account: On the one hand, DW end-user requirements need to be aligned with the information provided by these data sources. On the other hand, the data sources need to be integrated for analysis purposes based on a multidimensional representation. To address these challenges, we have previously proposed three MNFs [LV03] that formalize (i) correctness and completeness of DW schemata with respect to pre-existing data sources, (ii) avoidance of redundancies, and (iii) context sensitive summarizability in the presence of null values.

2 About the DWD Demo

We are currently developing and field-testing a tool called Data Warehouse Detective (DWD) to support the design of normalized data warehouse schemata. A prototypical version of this tool has been designed and implemented in the course of a project seminar involving six IS students during summer term 2006. Our demo for this tool consists of the following three parts.

First, we present how to design a new conceptual DW schema based on an analysis of pre-existing databases. To this end, DWD imports meta-data — i. e., tables, attributes, keys,
and foreign keys — of selected databases via ODBC. It then enriches these meta-data based on an analysis of database instances to detect missing keys, foreign keys, and functional dependencies, i.e., the meta-data which are often not declared properly in practice. DWD does that by creating and testing hypotheses about candidate constraints. It assumes that a candidate constraint holds if a systematic check does not exhibit any counterexamples. Afterwards, the designer chooses relevant attributes to populate the forthcoming data warehouse and, based on the functional dependencies previously recognized, DWD synthesizes multidimensional conceptual DW schemata for these relevant attributes, including dimension hierarchies, as sketched in [Lec03]. MNFs are used throughout this process to guarantee that all fact schemata under design really fit the pre-existing databases and to gain control over optional dimension levels with NULL values, which allows to avoid summarizability problems and inconsistent queries.

Second, we present a benchmarking of existing schemata against multidimensional normal forms, which is illustrated in the screenshot shown in Fig. 1. The lower part of the screen shows a hand-made DW fact schema in the banking domain, while the upper part shows a normalized schema synthesized by DWD based on an analysis of the underlying data sources. Both schemata represent account balances of individual accounts per day along with corresponding dimension hierarchies. Benchmarking these schemata, i.e., comparing...
the hand-made schema against the normalized one to detect violations of normal forms, leads to two observations highlighted in red by DWD in the lower schema: First, the hand-made schema is based on the assumption that regions can be assigned uniquely to countries, while the underlying data contradicts this assumption with regions that cross countries, e.g., the Alps. Thus, loading the dimension hierarchies of the lower schema is doomed to fail. Second, the assignment of the age of customers as a measure in an account related fact schema leads to redundancies, as the age of a customer will be repeated for each of his or her accounts. Here, normalization with DWD creates a second fact schema, reporting all customer related measures including age (not shown in the figure).

Finally, we demonstrate how to verify the consistency of DW schemata in the presence of (schema) changes in the pre-existing databases. As extracted and enriched database structures form a major input for DW design with or without DWD, one major challenge throughout DW projects arises from structural changes in data sources. We have started to analyze the effect of schema changes, which often lead to DW schema versions, in [GLRV06]. As an initial step towards analyzing the impact of such changes on DW schemata, DWD allows to check the consistency of a previously modeled DW schema with respect to the current state of data sources. The software reports any mismatches between data sources and DW schemata — e.g., deleted but necessary attributes, new or missing functional dependencies with an impact on multidimensional layout — and allows the user to revise the design accordingly.

As mentioned, DWD is currently under field testing in various DW production environments, where it shows promising performance. More information about DWD is available at http://www.dw-detective.de/ or directly from the authors.

Acknowledgments

We would like to thank Philipp Borgschulte, Philipp Dopjans, Katja Funke, Markus Heinrich, and Roland Reschka who together with the first author formed the team that did the initial implementation of DWD.

References


Managing the Desktop DocumentDataspace

Alexander Hilliger von Thile

DaimlerChrysler Research and Technology
P.O. Box 2360, 89013 Ulm, Germany
alexander.hilliger_von_thile@daimlerchrysler.com

Abstract: Documents such as spreadsheets are easy to create, edit, and exchange. However, their use causes a set of well-known problems such as poor data quality, lack of multi-user support and missing data in backend systems, e.g. for data analysis. In this paper we present our smart file prototype that extends proven concepts from DBMS to the desktop document dataspace. Firstly, we show how documents can be handled as db-objects using materialized external views. Secondly, we show how documents can be turned into a DBMS themselves.

1 Introduction

Many processes of an enterprise reach out to changing networks of external partners, are started spontaneously, are subject to frequent changes, and as a consequence need to be highly flexible. Due to development time and cost, for most of these processes no enterprise application and no database for data analysis exist. Furthermore, such processes are executed by non IT experts without deeper knowledge of DBMS. Thus, in these processes, documents (e.g. spreadsheets) are used for data exchange because they are easy to create, edit (even offline), and exchange. This causes problems such as poor data quality due to missing constraint checks, no up-to-date data in backend systems (e.g. for business intelligence (BI) analysis), and missing multi-user support.

Document management systems as well as recent technologies such as desktop search engines (only retrieve data without semantics) or personal data management systems (help to synchronize concurrent versions) do not tackle the underlying problem that data in documents is managed outside the control scope of a DBMS.

Recently, the data management community defined the concept of dataspaces [FH+05] to better address the problem space of ‘data everywhere’. Data in a dataspace (e.g. containing the documents mentioned above) can be queried and updated while the same data is accessible and updateable by existing backend systems that store their data in a DBMS, for instance. Here, the concept of dataspaces allows data to be managed by existing enterprise applications (to handle core processes) as well as through documents such as spreadsheets (to support frequently changing processes as described above).

In [HV05][HV06] we addressed this issue theoretically by introducing the concept of ‘materialized external views’ and ‘smart files’. While materialized external views handle documents as database objects, smart files turn documents conceptually into stand-alone DBMS which consist of a single file that can be exchanged by email as common documents. Both concepts aimed to extend the reach of proven DBMS
functionality to document driven processes. In this demo paper we present a prototype implementing these concepts.

2 Materialized External Views (MEVs)

From a DBMS perspective, a MEV can be thought of as a common materialized view that is updateable. Its content is defined by a relational expression (select-query). But, contrary to traditional materialized views, this content is materialized outside the DBMS’ tablespace in a file format that is used by desktop applications (e.g., an office or CAD product). In Figure 1, creating MEV-files, working with them and reintegrating changes performed on MEVs are depicted. These steps are described next.

Creation: A MEV is created using a statement comparable to a common ‘create view statement’ that is extended by a reference to a template document (see Fig. 2). This template document contains bijective mapping definitions to map rows within the RDBMS to regions in the MEV-file and vice versa. Using the data from the (existing) base relations and the template document, the RDBMS can create the document (Fig. 3) that can be edited by the user. MEV files are under control of the DBMS, but in contrast to datalinks [MED], the DBMS not only keeps a ‘link’ to a complete file but also handles its contents on an attribute based granularity.

Metadata can be attached to the MEV files (Fig. 4), e.g., to increase data quality by adding integrity constraints. Such metadata is managed in virtual directories (denoted by <filename>/) and can contain the metadata managed by the DBMS (e.g., SQL assertions) along with externally managed metadata (e.g., an XML schema document).
**Working:** Users can edit the documents using their existing, unmodified desktop applications by accessing the MEV-files provided on our managed file share.

**Reintegration:** In contrast to commonly used import/export processors, the mapped content of the view is not stored as a file. Instead, the MEV-document is a virtual file that is being managed by our prototype and that is accessible by desktop applications via a virtual (shared) drive. We use an optimistic concurrency control protocol that does not use locks. Therefore, the ‘same’ file can be edited by multiple users concurrently. To guarantee serializability of changes, we use a mechanism comparable to snapshot isolation we refer to as ‘activities’.

In Fig. 5 the user interface to start, commit, abort (rollback) and suspend activities is depicted. After an activity has been started by a user, all changes performed on files are stored within an isolated shadow copy (SC) that is invisible to other users. The SC stores all changed documents (after-images) during this activity by this user as well as the original documents (before-images). These images together with the mapping definition are used to reconstruct performed operations (insert, delete, update of rows). At the end of an activity – which is comparable to a commit in the transactional case – firstly, the change operations are reconstructed. Secondly, these changes are compared for conflicts with changes made by other users (see Fig. 5). Finally, the changes are checked for integrity (constraint-checks, Fig. 6). In contrast to DBMS, constraint violations do not roll back changes. Instead, the user has to correct the violations by editing the document again or by manually aborting the activity.

**3 Smart Files**

Downloading and exchanging MEV-files by e-mail still bears the problem that only data is being exchanged in documents. Thus, constraints cannot be checked and changes cannot be propagated to backend systems as soon as a user downloads a document. Since all benefits of a DBMS are lost if the document is being exchanged, we conceptually turned the document itself into a stand-alone DBMS by combining the document’s data with metadata and the execution logic of a DBMS.
The basic principle is based on a mobile DBMS consisting of a single file (Fig. 7) that can easily be exchanged by e-mail. This file is executable and contains a managed resource part where common documents and MEV-files can be stored. Metadata can be attached to these files to define integrity constraints or access rights as done in a data dictionary of a DBMS. Since this concept enhances traditional file-usage paradigms, it is called ‘smart’ file (SF) [HvT06]. A SF is a file-container, comparable to self-extracting zip-archives. It mounts its resource part as a virtual drive. However, any read/write access is managed by the SF itself, it controls who can do what, when, and where.

4 Related Work

In contrast to other approaches, we do not substitute document exchange, even though plenty of alternatives are available today (e.g. WIMS or groupware systems). As described in the introduction, documents are preferred by end-users for non-technical reasons that are not preserved by these alternatives (cost, development time). Today, working with documents in DBMS is handled with a mix-up of concepts from federated databases, data-integration (e.g. foreign tables in SQL/MED [MED]) and data-transformation by using import/export processors. Our approach extends the well known concept of views to bridge the gap between DBMS and documents. In the literature, plenty of work exists in the domain of view materialization (e.g. [KS+04]). However, since only data is exchanged in documents, all benefits of a DBMS are lost as soon as common documents are being exchanged. We turned documents into a stand-alone DBMS by combining the document’s data with DBMS’ execution logic and metadata. Related work can be found in the domain of electronic forms – but common documents such as CAD-files or spreadsheets can only be handled as attachments (e.g. InfoPath). A general approach is based on living [SK02] or active documents [AB+03]. However, those do not address DBMS/dataspace related issues (support for transactions, etc.).

5 References

[HvT05] Hilliger von Thile, A., Melzer, I.: Smart Files: Combining the advantages of DBMS and WIMS with the simplicity and flexibility of spreadsheets. BTW. Karlsruhe, Germany, 2005.
Efficient Access Control for Composite Applications

M. Wimmer¹, M.-C. Albutiu¹, A. Kemper¹, M. Rits², and V. Lotz²

¹ Technische Universität München, 85748 Garching b. München, Germany
² SAP Research, Font de l’Orme, 06250 Mougins, France
{wimmerma, albutiu, kemper}@in.tum.de, {maarten.rits, volkmar.lotz}@sap.com

1 Motivation

Composite applications rely on further sub-applications – also called sub-activities in the following – to implement their functionality. There are numerous examples including quite simple Web applications as well as large scale enterprise resource planning (ERP) systems that interact with database backends. Also, business processes that are realized as Web service workflows represent complex composite applications. Thereby, sub-activities can constitute composite applications themselves.

In general, sub-applications are self-contained software modules that autonomously enforce their own security policies. This autonomy of authorization can lead to significant performance drawbacks: On the one hand, the authorizations of legitimate users are evaluated repeatedly. On the other hand, requests of ultimately unauthorized users that lack authorizations at later stages of the workflow can lead to transaction rollbacks or demand for compensating transactions. Thus, it appears beneficial to evaluate the authorizations of users as soon as possible by shifting access control to the workflow layer instead of retaining it at the sub-activities.

Regarding composite applications this can be a non-trivial task, as the access control configurations of several autonomous sub-applications have to be taken into account. The key to success is a consolidated view onto the access control of composite applications providing answers to the following questions: (1) What are the least required privileges?, (2) Who is allowed to execute the composite application?, and (3) Are there possibilities to reduce policy evaluation costs?

The first issue addresses the principle of least privilege, denoting that only those privileges are granted which are required in the context of the sub-activities. Following this design paradigm reduces security vulnerabilities as it guarantees that no business resources other than the ones needed by the composite application can be accessed. As we showed in [WEK05, WEFK05], this restriction is of particular importance for the design of Web services that interact with database systems.

Knowing the group of authorized users allows to detect unintended configurations more easily. For instance, if only highly privileged users like managers are authorized to execute a business process, this might be an indication that the composite application itself has to be revised. We are addressing this issue from the single-user / single-role perspective, meaning that a user can execute the application by
the activation of one task specific role. This complies with many business processes
which are typically representing job specific tasks. Therefore, composite applica-
tions are to be distinguished from multi-user workflows which are business processes
that are executed by several users in a team.

Optimization capabilities for composite applications – as addressed by the third
issue – can be given in two ways: On the one hand, a consolidated policy allows
the early-filtering of requests. Application invocations which will lead to aborts at
later stages in the process due to missing privileges can be detected and averted.
On the other hand, repeated and redundant authorization checks by the individual
sub-activities can be omitted, in case the authorization decision can be inferred on
the composite application’s layer.

In this contribution, we show how consolidated policies of single-user workflows can
be generated. This optimization technique has been integrated into SAP Research’s
workflow management tool suite which allows to compare traditional and optimized
policy evaluation strategies.

2 Consolidating the Access Control of Composite Applications

In order to consolidate the access control of composite applications, the workflow
structure, dataflow dependencies, and external dependencies have to be taken into
account. Details about the consolidation process have been described in [WAK06]
and [WKRL06]. The workflow structure defines the control flow, i.e., the execution
order of the sub-activities as illustrated in Figure 1(a). From an access control
point of view, sequential or parallel executions denote that all sub-activities are
invoked. We represent this characteristic through the sequence pattern. Further-
more, conditional and event based executions are possible which – from the access
control perspective – denote that only one sub-activity will be invoked. We repre-
sent this aspect through the switch template. The access control dependencies of a
composite application can then be represented by means of a tree as illustrated in
Figure 1(b). The composite application’s policy is generated through a bottom-up
analysis, combining the policies of the individual sub-activities. Users need to be
granted execution privileges by all policies that apply to the sub-activities in order
to be able to execute a sequence pattern. That is, the combined policy for a se-
quence pattern consists of the intersection of subjects and the union of all privileges
defined in the policies of the autonomous tasks.

Regarding switch patterns, two different approaches can be applied: Following
the full-authorization approach, users have to be authorized to perform all sub-
activities, irrespective which one would actually be executed. Hence, policies are
combined in the same way as for sequence pattern. In contrast to this, the partial-
authorization approach considers each execution path individually. Regarding the
switch node in Figure 1(b), users can execute the left branch in case they are au-
thorized by policy $P_{3,4}$ and they can execute the right branch if they are authorized
by $P_2$. Consequently, separate policies will be generated for the different execution
paths of a workflow.

The resulting consolidated policy includes those privileges needed to execute the
composite application, respectively workflow path. Thus, in case the policies of the sub-applications realize the principle of least privilege, this paradigm can also be inferred for the combined policy. Furthermore, the consolidated policy constitutes the basis for an optimized access control. If the full-authorization approach is applied, authorization checks can be shifted to the workflow management system (WFMS) and be omitted at lower execution levels. Thus, policy evaluation costs can be saved significantly. In case of the partial-authorization approach, authorizations for the individual workflow branches have to be evaluated separately, meaning that the WFMS has to be capable of enforcing access control at workflow branches. Both approaches help to avoid situations that demand for transaction rollbacks and compensating transactions. This is because requests are filtered before the execution of a composite application (full-authorization) or before entering a workflow branch (partial-authorization), so that ultimately unauthorized requests are detected as soon as possible.

3 Demonstration

The policy consolidation approach has been integrated into SAP Research’s workflow management tool suite, including the three components Maestro, Nehemiah and Gabriel. Maestro is used to model business processes, by defining a set of sub-activities and their interdependencies, i.e., the control flow. Via drag-and-drop, components like sub-activity nodes or control flow nodes can be inserted and connected. Nehemiah is the workflow management engine that allows to execute business processes which have been designed using Maestro. At runtime, Nehemiah allows to supervise the state of the workflow by keeping track of active sub-activities. Thus, Maestro and Nehemiah are used for workflow modeling and activation. Individual sub-activities, on the other hand, are modeled and activated by Gabriel. Gabriel allows to specify the roles needed to execute sub-activities. At design time, sub-activity profiles are defined that describe which actions have to be performed when executing a certain task. For instance, an action can be the invocation of a Web service. When modeling a workflow with Maestro, sub-activity nodes can be associated with the corresponding sub-activity by means of the profile. Furthermore, subjects like roles and users can be modeled and these subjects can be granted the privileges required to execute respective sub-activities. The relationships between the three programs are illustrated in Figure 2.

Nehemiah supports the execution of multi-user workflows, denoting that the sub-
activities can be executed by teams. We complemented Nehemiah’s policy enforce-
ment strategy with the special treatment of single-user workflows and composite
applications. For this purpose, we integrated the full-authorization approach. In
the course of the demonstration, the theoretical backgrounds of our policy consoli-
dation approach are presented and illustrated by means of use cases. The consoli-
dation of Web service policies (coded in form of XACML policies) is demonstrated
and the optimized single-user execution is compared to the traditional approach
that relies on separate policy enforcements.

References

[WAK06] M. Wimmer, M.-C. Albutiu, and A. Kemper. Optimized Workflow Autho-
rization in Service Oriented Architectures. In Proceedings of ETRICS ’06,
volume 3995 of LNCS, pages 30–44, Freiburg, Germany, June 2006.

Autorisierung in Datenbank-basierten Web Service-F dérationen. IFE,

Service-F éderationen. In Proceedings of BTW ’05, pages 185–204, Karlsruhe,
Germany, February 2005.

[WKRL06] M. Wimmer, A. Kemper, M. Rits, and V. Lotz. Consolidating the Access
Control of Composite Applications and Workflows. In Proceedings of DBSec
’06, volume 4127 of LNCS, pages 44–59, Sophia Antipolis, France, August
2006.
Pathfinder:
A Relational Query Optimizer Explores XQuery Terrain

Jan Rittinger    Jens Teubner    Torsten Grust
Technische Universität München, Institut für Informatik
{jan.ritten,jens.teubner,torsten.grust}@in.tum.de

1 Purely Relational XQuery

Relational encodings of the static aspects of the XQuery data model, i.e., tabular representations for XML documents and ordered sequences of items, are widely used today. Since 2002, the Pathfinder and MonetDB/XQuery companion projects [BGvK + 06] pursue the primary goal to also embrace the complete dynamic semantics of XQuery (expression evaluation and runtime aspects) with the help of relational database systems.

In earlier work [GT04], we have shown that relational algebra makes for a suitable target language in an XQuery compiler. This purely relational approach to XQuery inherits the scalability advantages of the underlying relational database back-end and makes proven optimization techniques immediately applicable to the construction of XQuery processors. MonetDB/XQuery, an open-source system that implements this approach, is found among the fastest and most scalable XQuery processors available today [BGvK + 06].

This is a demonstration of the relational optimizer of Pathfinder¹, the query compiler behind MonetDB/XQuery. To account for the significant size and unusual shape of the relational query plans (see Figure 1) derived from input XQuery expressions, Pathfinder implements various optimization techniques in a peephole-style fashion and provides support for graph-shaped plans from the ground up.

2 Relational Query Optimization in an XQuery Compiler

Pathfinder’s XQuery compiler turns incoming XQuery expressions into relational query plans according to the loop-lifting compilation strategy we devised in [GT04]. In a nutshell, loop-lifting trades iteration (esp. the XQuery FLWOR construct) for efficient bulk-oriented processing. The compiler emits expressions of a relational algebra whose operators have been chosen to match the actual capabilities of modern SQL query engines. A few representative operators are shown in Table 1 (note that non-standard operators like the XPath step join ∈ are synonyms for relational “micro-plans” with an optimized implementation in Pathfinder’s back-end database system MonetDB).

¹MonetDB/XQuery and Pathfinder are available via http://www.pathfinder-xquery.org/.
To correctly reflect the complex XQuery semantics—different notions of order, side-effecting node construction and node identity, existential predicate semantics, and implicit casting and atomization—the resulting plans are of significant size (50–500 operators). Figure 1 gives an impression of the relational plan for XMark query Q8 [SWK+02] along with the total number of plan operators for each of the XMark queries.

Loop-lifted evaluation plans typically exhibit plenty of opportunities to share common subexpressions. Pathfinder represents query plans as directed acyclic graphs (DAGs) to account for this sharing. XQuery is a compositional expression-oriented language in which subexpressions are stacked upon each other to form complex queries. Note how the stretched shape of the plans in Figure 1—which is somewhat different from the well-known SQL-induced π–σ–⋈ query pattern—reflects this compositionality.

### 2.1 Rewriting Large Plan DAGs

To obtain a bird’s eye view of these large plans and maintain focus during optimization, Pathfinder identifies basic blocks (straight-line operator sequences in the DAG with no sideways entries), much like compilers for programming languages. A further technique that saves the optimizer from getting lost in large plan DAGs is peephole-style inspection that considers one operator at a time. To compensate for this restricted peephole view, a property inference phase precedes the actual rewriting. The inference carries additional information about the relevant vicinity of each plan node into the operator itself. We highlight some of these inferred plan characteristics in the following.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>π</td>
<td>column projection, renaming</td>
</tr>
<tr>
<td>σ</td>
<td>row selection</td>
</tr>
<tr>
<td>⊗</td>
<td>equi-join</td>
</tr>
<tr>
<td>×</td>
<td>Cartesian product</td>
</tr>
<tr>
<td>@</td>
<td>row numbering</td>
</tr>
<tr>
<td>⬤</td>
<td>arithmetic (UNION ALL)</td>
</tr>
<tr>
<td>⬦</td>
<td>XPath step join</td>
</tr>
</tbody>
</table>

Table 1: Subset of the relational algebra emitted by the loop-lifting compiler. Operator ⬤ is the equivalent of SQL:1999’s ROW_NUMBER operator—see [GT04] for details.

Figure 1: Plans for XMark query Q8. (a) before and (b) after relational optimization. Table: Operator counts for the 20 XMark queries.

<table>
<thead>
<tr>
<th>Q</th>
<th>(a)</th>
<th>(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>138</td>
<td>37</td>
</tr>
<tr>
<td>2</td>
<td>91</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>385</td>
<td>135</td>
</tr>
<tr>
<td>4</td>
<td>325</td>
<td>67</td>
</tr>
<tr>
<td>5</td>
<td>132</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>53</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>79</td>
<td>46</td>
</tr>
<tr>
<td>8</td>
<td>227</td>
<td>66</td>
</tr>
<tr>
<td>9</td>
<td>371</td>
<td>101</td>
</tr>
<tr>
<td>10</td>
<td>500</td>
<td>243</td>
</tr>
<tr>
<td>11</td>
<td>234</td>
<td>91</td>
</tr>
<tr>
<td>12</td>
<td>316</td>
<td>110</td>
</tr>
<tr>
<td>13</td>
<td>87</td>
<td>43</td>
</tr>
<tr>
<td>14</td>
<td>86</td>
<td>54</td>
</tr>
<tr>
<td>15</td>
<td>100</td>
<td>31</td>
</tr>
<tr>
<td>16</td>
<td>149</td>
<td>51</td>
</tr>
<tr>
<td>17</td>
<td>97</td>
<td>42</td>
</tr>
<tr>
<td>18</td>
<td>101</td>
<td>48</td>
</tr>
<tr>
<td>19</td>
<td>142</td>
<td>76</td>
</tr>
<tr>
<td>20</td>
<td>392</td>
<td>147</td>
</tr>
</tbody>
</table>

618
2.2 Column Pruning

Pathfinder’s fully compositional compilation may lead to relational plans that generate table columns whose contents may not be relevant to the semantics of a given query. Sequence order (maintained in column pos [GT04]), for example, is not relevant to compute the result of the XQuery general comparison operators (\(=, <, \ldots\)). To avoid the runtime overhead incurred by the generation and maintenance of such columns, Pathfinder annotates each relational plan operator with the set of columns that is strictly required to process its upstream plan. These annotations are then used to drive a variant of projection pushdown [Gru05, JK84].

Loop-lifted XQuery evaluation plans are very susceptible to this optimization. Columns carrying the type annotations derived by XQuery’s validate \{\} construct, for example, will only be retained in the optimized plans if the surrounding XQuery expressions indeed inspect these types (e.g., via typeswitch or instance of). Similarly, we are currently extending Pathfinder’s internal data model with a score column intended to support XQuery Full-Text retrieval [AYBB+06]. Column pruning will ensure that this will not negatively affect the performance of queries that do not use XQuery Full-Text features.

2.3 Functional and Multi-Valued Dependencies

Due to loop-lifting, optimization opportunities like the occurrence of invariant subexpressions inside (deeply nested) XQuery FLWOR blocks or the presence of value-based joins, surface as functional and multi-valued dependencies in the relational plans. During the property inference phase, Pathfinder derives the validity of degenerate functional dependencies of the form \(\emptyset \rightarrow c\) for all columns \(c\) produced by sub-plans of the query DAG. If \(\emptyset \rightarrow c\) holds for a relation, all its rows carry the same value in column \(c\)—an indicator for Pathfinder’s optimizer to initiate constant propagation and folding in the plan vicinity. Likewise, the presence of a degenerate multi-valued dependency \(\emptyset \rightarrow c_1, \ldots, c_n\) in a relation signals that its columns \(c_1, \ldots, c_n\) are independent of all its remaining columns.

In a loop-lifted query plan, this multi-valued dependency is the relational expression of the fact that a loop-invariant computation occurs inside an XQuery for-iteration. In such cases, Pathfinder performs a variant of loop hoisting to improve the original plan.

Note that relational dependency analysis is indifferent to XQuery’s syntactic diversity—in XQuery syntax, value-based joins are not as prominent as in SQL and come in various flavors—and will detect the value-based join in \(\text{let } \$d := \text{fn:doc(\ldots)} \text{ for } \$a \text{ in } \$d//a \text{ return } \$d//b[@c = \$a/@d]\), for example.

2.4 Data Flow Analysis Based on Active Domains

Pathfinder’s join recognition logic is further supported by data flow analysis on relational plan DAGs. For all intermediate result tables \(t\), the system infers an approximation \(\alpha_c\) of the active domain for each column \(c\) in \(t\) [Klu80]. The data flow may then be inferred from the inclusion (or disjointness) of these active domains: for two columns \(c_1\) and \(c_2\) of arbitrary intermediate result tables, the inclusion \(\alpha_{c_1} \subseteq \alpha_{c_2}\) indicates data flow between
the respective plan operators.

Among many other uses, data flow analysis helps to maintain the correspondence between the pair of branches that results from the compilation of an XQuery if-then-else clause: the active domain relationships remain intact even if extensive rewrites move the then and else DAG branches far apart. Further, active domain information is an essential building block of a procedure that provides cardinality estimates for arbitrary XQuery subexpressions (not just the overall query result).

3 Demonstration Setup

We demonstrate an instance of MonetDB/XQuery against which users may run arbitrary queries in an interactive fashion. The system will be preloaded with various XMark XML instances (of 100 KB–1 GB serialized size). Most importantly, users will be able to look under the hood of the Pathfinder compiler and experience the effect of the aforementioned relational optimization techniques. The demonstration system renders the relational query plans (much like the plans in Figure 1) to enable the inspection of plan characteristics at various stages of Pathfinder’s highly-configurable optimizer pipeline. Stages may be separately enabled to judge their impact on plan quality and XQuery evaluation performance.

Acknowledgment. This research is supported by the DFG (Deutsche Forschungsgemeinschaft) under grant GR 2036/2-1. We thank the MonetDB development team at CWI, Amsterdam, for the lasting and fun collaboration.

References


SmurfPDMS: A Platform for Query Processing in Large-Scale PDMS

Katja Hose Christian Lemke Jana Quasebarth Kai-Uwe Sattler
Department of Computer Science and Automation, TU Ilmenau
P.O. Box 100565, D-98684 Ilmenau, Germany

Abstract: As Peer Data Management Systems (PDMS) are a focus of current research, there are lots of approaches like query processing or routing issues that have to be evaluated. Since there is no common platform approaches are evaluated in separate. This is disadvantageous for research groups in two ways. First, it means a huge effort to build a simulation environment from scratch. Second, this makes a direct comparison of approaches more difficult. In this paper, we present SmurfPDMS an extensible system that means to provide a common platform for all researchers in that they can easily integrate their approaches and that allows for running large simulation experiments in distributed environments such as workstation clusters or even PlanetLab.

1 Introduction

Peer Data Management Systems (PDMS) – also known as schema-based P2P systems – are an important area of recent and current research. Emerging from federated database systems and applying the P2P paradigm, PDMS have to counteract the challenges coming along with peer autonomy. This means that all peers are equal in terms of issuing and processing queries, each peer possesses and owns its private local data, and each peer might have a local schema that is unique in the whole network. Furthermore, each peer can only communicate with those neighbor peers to which mappings exist.

This and the fact that we consider unstructured P2P systems as the basis for PDMS requires efficient distributed query processing strategies that do not need any kind of global knowledge. In contrast to structured P2P systems like Chord [SMK+01], PDMS do not have global indexes or hash functions that could help us find the data we are looking for. Since we are neither allowed to rearrange the peers’ data we have to find other possibilities to route queries efficiently through the network. A common means to do this are routing indexes [CGM02] that can be used to identify which neighbors hold data that matches the query.

Though aspects like routing indexes, query processing strategies, or dynamic behavior have a great influence on each other, they are usually considered independently from each other by different research groups. The use of different platforms, implementations, and assumptions hamper a direct comparison of similar concepts. Additionally, many existing simulators like ns-2 (http://www.isi.edu/nsnam/ns/) are too low-level for simulating PDMS appropriately. Most environments have another severe drawback: their lack of documentation and extensibility. Furthermore, they often do not have an intuitive user interface let alone a graphical one that might allow outsiders to configure and use the system.

In this paper, we do not present yet another PDMS system in addition to systems like Piazza [TIM+03] but SmurfPDMS (SiMUlation enviRonment For PDMS) a common plat-
form for simulating PDMS. Due to its architecture it can easily be extended so that it enables a direct comparison of approaches in the same environment using the same environmental setup. The remainder of this paper is structured as follows. After having discussed architectural issues in Section 2, Section 3 points out SmurfPDMS’ features and usage. Finally, Section 4 outlines what we are planning to show at the conference.

2 Architecture

Running SmurfPDMS as distributed simulator enables the user to combine the computational powers of several computers. For this purpose, each participating computer (e.g., a number of PlanetLab nodes (http://planet-lab.org)) has to run its own instance of SmurfPDMS. Hence, we have to distinguish between the (i) simulated PDMS network and (ii) the network that is formed by the participating computers. Each SmurfPDMS instance – running either on the local or on a remote computer – has the same capabilities and can act either as coordinator or participant – the coordinator is always that instance at which the user starts the simulation.

The simulated PDMS network is determined at startup by the coordinator in consideration of the user-defined configuration. The peers of the simulated network are then divided into partitions. Each partition is assigned to one of the participants to distribute load among the computers. In summary, the most important tasks of a coordinator are:

- Determine the setup, calculate a network topology, assign the peers to the participants, calculate data partitions, etc.
- Synchronize all participants
- Choose queries and determine peers to initiate them
- Have peers update their local data
- Determine peers to leave or join the network
- Simulate communication and processing delays
- Log events, messages, results, etc.
- Collect statistics and create diagrams

SmurfPDMS implements several central concepts: managers that control the simulation and the communication, messages for information exchange, class hierarchies for a straightforward extensibility, configuration files and logging to achieve repeatability. SmurfPDMS has three logical layers that we sketch in the following.

Network Layer. All communication and all messages are sent via this layer using TCP/IP. The corresponding configuration of an instance like listening port, instance name, or the list of known computers is held by the Network Manager. Based on the list of available computers the user selects those computers that later on participate in the simulation.

Simulation Layer. After having started the simulation the coordinator’s Startup Manager determines the initial setup by calculating a network topology, peer objects (representing peers of the simulated network), etc. The simulator currently already provides 8 different algorithms for topology generation (including one to read the topology from
existing config files for the sake of repeatability) – others can be added easily. Local and Remote Peer Managers provide an appropriate interface for the communication between peers so that the simulation engine does not have to take care on which computer the receiving peer object is actually located. The statistics component gathers statistics during the simulation. After the simulation is finished all participants send their local statistics to the coordinator which finally calculates the total statistics, serializes them, and calculates diagrams (Figure 3). Especially the Simulation Layer is affected when new approaches are to be integrated into the system. Figure 1 illustrates the Simulation Core with the entry points for query operators, query processing strategies, query rewriting algorithms, routing indexes, and event processing strategies.

GUI and Configuration Layer. This layer operates on top of the other two and makes working with the simulator comfortable (Figure 2). It visualizes the selection of participating computers and can be used to set all necessary configuration parameters. It also visualizes the simulation itself by displaying the network with peers, links and messages – revealing detailed information by simply clicking on the corresponding symbols. The same window enables the user to control the simulation (issue a query, start, halt, or stop the simulation). Finally, the GUI Layer illustrates the simulation statistics (Figure 3).

3 Distinguishing Characteristics

As already mentioned above SmurfPDMS aims to provide a common platform that can be used by multiple research groups in order to evaluate their approaches and to accomplish comparisons and interactions to other approaches with low effort. The following aspects can currently be evaluated using SmurfPDMS: (i) Query Routing, (ii) Query Processing Strategies, (iii) Routing Indexes, (iv) Mapping Definitions, and (v) Query Rewriting. In

Figure 2: Simulation Window

Figure 3: Statistics Window and Diagrams Resulting from Test Series
we have already demonstrated the former three aspects. Recently, we have completed the latter two. Additionally, we enhanced simulating data updates, enabled SmurfPDMS to run test series automatically, and integrated the generation of gnuplot files. Of course, the parameters of these features can be adapted to the users needs. Eventually, the simulator could also be augmented to support the evaluation of strategies with the aim of changing the network topology, e.g., in order to build semantic clusters. Furthermore, schema matching techniques could also be integrated. In summary, the specialties that distinguish SmurfPDMS from other systems are:

- extensibility in a variety of aspects,
- evaluation of approaches not only in separate but also in interaction with others,
- comfortable configuration and controlling thanks to the graphical component,
- providing a whole variety of algorithms for repeating tests and for generating initial setups,
- offering a powerful evaluation component that allows for running whole test series and creating the corresponding diagrams in gnuplot format that can be converted into various formats (fig, eps, pdf, etc.),
- platform independence by using Java as programming language,
- scalability using PlanetLab as platform.

Finally, let us emphasize that SmurfPDMS was especially designed for the needs and specialties of PDMS. With XML as native data format SmurfPDMS integrates another interesting aspect of current research.

4 Demonstration

At the demonstration site we want to show how to work with SmurfPDMS and how it could help research groups to conduct experiments. This includes all steps starting with configuration and ending with the evaluation of statistics. In contrast to [HJKS06] we want to emphasize the distinction of (i) running predefined test series and (ii) using SmurfPDMS interactively in the single-step mode where the user can monitor the execution of each message, each rewriting step and each event. We will also show how to set up a configuration, how to compare different strategies (query processing, routing indexes, etc.), and how to evaluate them properly. Finally, we will show how to create evaluation diagrams for test series that can serve as evaluation results for publications without further adaptations.

References


System P: Completeness-driven Query Answering in Peer Data Management Systems

Armin Roth
aroth@informatik.hu-berlin.de
Humboldt-Universität zu Berlin, Germany

Felix Naumann
naumann@hpi.uni-potsdam.de
HPI, Universität Potsdam, Germany

Abstract: Peer data management systems (PDMS) are a highly dynamic, decentralized infrastructure for large-scale data integration. They consist of a dynamic set of autonomous peers inter-connected with a network of schema mappings. Queries submitted at a peer are answered with local data and by data that is reached along paths of mappings. Due to redundancies in the mapping network, query answering in PDMS can be very inefficient if the complete query result is to be computed.

System P, a fully functional PDMS, compromises the completeness of the query result and reduces cost by pruning the query plan at mappings that are estimated to yield only few result tuples. The demo illustrates the following main components of System P: (1) adaptive estimation of result cardinalities of intermediate queries using histograms, (2) completeness-driven query planning under limited resources using specialized heuristics, and (3) the automatic generation of heterogeneous PDMS test instances, controlled by a rich set of parameters.

1 Optimizing completeness in large-scale data integration

Sharing and integrating relevant information is a pressing problem. Peer data management systems (PDMS) have been proposed to solve this task, because they can be extended to large scales solely by de-centralized coordination [HHNR05]. Query answering in PDMS can be very inefficient due to high redundancy in the network of mappings (Fig. 1). Additionally, the schema mappings between the peers suffer from information loss due to selections, which capture implicit knowledge about neighboring peers, and projections, which reflect the fact that different peers may offer different attributes about a certain real-world entity type.

In contrast to recent work [TH04], System P Consequently meets the important practical requirement of peers to retain as much autonomy as possible. In our PDMS, peers only exchange queries with an associated budget and resulting tuple sets. The main idea of completeness-driven query planning in System P is to prefer mappings to peers with potentially large cardinalities and to prefer mappings to peers that preserve much of the data collected by the peers “behind” a certain mapping. To prepare such decisions, each peer ranks all of its outgoing peer mappings according to the potential amount of data returned. This

1See animated demo at www.hpi.uni-potsdam.de/~naumann/projekte/system_p
Figure 1: Screenshot of the monitor peer of System P with automatically generated PDMS (left) and architecture of System P with monitor peer and PDMS peer (right).

is achieved by estimating result cardinalities using multi-dimensional histograms. Second, the influence of data returned by mappings on the completeness of the local query plan is assessed using a completeness model (Sec. 2.1). System P implements different strategies to control query planning referring to these completeness estimates (Sec. 2.2). Examples for application areas of PDMS are cooperations of scientific institutions or disaster management, which will be covered in our demo. In this field, the computing and networking resources may be restricted due to damaged infrastructure. Moreover, flexible integration of new peers is crucial. Especially the completeness of queries with aggregation may be compromised to get a rough overview about the need of relief supplies. Also, the computing and networking resources may be restricted due to damaged infrastructure, PDMS can also serve as a de-centralized infrastructure for query mediation over ontologies in the Semantic Web [HHNR05]. In general, PDMS tend to extreme inefficiency or even intractability when scaled up to tens or hundreds of peers. Therefore, techniques optimizing query answering even at cost of completeness are crucial for PDMS.

2 Components of System P

A query reformulated at a certain peer heavily branches into many more downstream queries. If there is no pruning, the results of all these subsequent queries are transported back. However, projections in the mappings may lead to a significant decrease of the amount of data collected by all peers involved in downstream query answering. So assessing that a certain mapping probably returns only few tuples can significantly save effort.

2.1 Estimating cardinality contributions

In [RN05] we discuss how a completeness model drawn from previous work can be applied to PDMS. It provides formulas to locally calculate the completeness of query plans based
on projections and estimated result sizes of the mappings. Many peers are only reachable indirectly from a particular peer. System P employs multi-dimensional histograms \[AC99\] to estimate the cardinality of query results accessible by a certain mapping. This approach uses query feedback to maintain the histograms and is especially suited for PDMS, because their dynamic nature forces to continuously adapt histograms to changing data distributions. Because pruning of the search space and exploiting query feedback are conflicting goals, System P trades off between them to sufficiently keep track of data “behind” a mapping.

2.2 Completeness-driven query answering

Query reformulation in System P adapts to the data distribution in the PDMS and the information loss in mappings. Basically, a peer receiving a query must rank different local query plans according to their potential result sizes and prune some.

**Pruning subplans.** The approach shown in the demo valuates each mapping in isolation by comparing the result sizes assessed for local query plans (1) with and (2) without the mapping. Intuitively, this provides us with a measure for the data contribution of a certain mapping in the context of a certain local query plan. Our simple threshold-based pruning approach only uses mappings, whose potential amount of returned data is above a certain threshold \[RN05\].

**Limited resources.** To additionally bound resource consumption, we implemented query planning strategies, which assume that peers accept a budget to be spent for answering a query \[RNHS06\] . We distinguish a depth-first strategy and an approach that distributes budget in proportion to the potential data contribution of a mapping. Both can be combined with the refunding of budget not usable at a peer. We experimentally compared all of these strategies and found that the efficiency gains of the budget-driven approach are comparable to the threshold-based strategies. Our algorithms lead to interesting effects in exploring the search space, which will be shown in the demo.

3 System P Architecture and Demonstration

The main components of System P are depicted in Fig. 1. PDMS Peers form the actual, physically distributed PDMS. They run as stand-alone Java programs on any machine in the network. The monitor peer serves as a central unit for controlling the creation of the PDMS, providing a GUI, and collecting statistics about experiments. Additionally, System P includes a powerful generator to automatically create large PDMS test instances. The monitor peer serves as GUI to pose queries to any PDMS peer and to visualize how queries propagate in the system and how query results are passed along.

**Demonstration.** The demonstration consists of three phases: (1) Generation of PDMS instances, (2) initial adaption of the histograms, and (3) user queries and dynamic changes of the PDMS instance.
**PDMS Generation.** In the first phase, the viewer can create a PDMS test instance by using a wizard provided by the monitor peer. Many parameters influencing properties of the resulting PDMS can be varied. We use a complex reference schema from disaster management dealing with concepts like hospitals, medications, victims, and relief personnel. Next, the resulting PDMS instance is distributed to PDMS peers running in a physically distributed network: System P will run on several connected PCs to demonstrate that the demo is not a simulation but a full-fledged PDMS.

**Histogram adaption.** Before user queries can be answered efficiently, the histograms related to the mappings are adapted to the skewed multi-dimensional data distribution in the PDMS. During this phase, the monitor peer reports the estimation accuracy of the histograms.

**Query answering in a dynamic PDMS.** Now the system will answer user queries, optionally controlled by a user-defined budget. The viewer can choose to pose conjunctive queries with semi-interval comparison predicates to any peer or observe a randomly generated workload. System P graphically visualizes the propagation of intermediate peer queries as well as the transfer of query results back to the peer the user query originated. This intuitively displays how the search space of the PDMS is explored over time and thus shows the differences between our strategies for budget spending. Additionally, we will present PDMS instances illustrating the advantages and limitations of our strategies. The efficiency of query answering is determined by comparing the cost in terms of mappings used and the result size. Moreover, the monitor peer displays the resulting overall query plan for each user query. During this main phase, the viewer can change the PDMS topology and data distribution by selectively turning peers offline and observe how the histograms and the query answers of System P adapt to these changes.

**Acknowledgments.** This research was supported in part by the German Research Society (DFG grant no. NA 432).

**References**


Visually Exploring and Querying XML with BaseX

Christian Grün, Alexander Holupirek, Marc H. Scholl

Databases and Information Systems Group
University of Konstanz
<Firstname>.<Lastname>@uni-konstanz.de

Abstract: XML documents are widely used as a generic container for textual con-
tents. As they are increasingly growing in size, XML databases have come up to
efficiently store and query their contents. Besides, due to the hierarchic structure of
XML documents, hierarchic visualizations are needed to facilitate cognitive access
to query results. BaseX is a simple database prototype, mapping XML documents
to a table-based tree encoding. An integrated treemap visualization and a query in-
terface allow visual access to the documents and demonstrate the interactive
response time of the underlying data storage.

1 Introduction

XML has since long become the standard format for storing arbitrary text contents.
Whereas many XML instances are rather small, more and more documents are available
which exceed main memory constraints. All major database vendors have already inte-
grated XML as a fixed datatype. Still, the versatility of the XML format leads to a
general mistrust, concerning the efficiency of storing and querying. However, XML
documents are simple tree structures, and plenty of well-known and proven techniques
exist to handle the data structure. Furthermore, the idea to store XML in relational tables,
such as e.g. discussed in [FK99], generalized by the XPath Accelerator [Gr02] and real-
ized in the MonetDB/XQuery database [Bo06], has proven to be surprisingly efficient
both in terms of speed and memory usage.

With BaseX, we present a database prototype of a table-based tree encoding of XML
nodes [Gr06]. The current version of BaseX is disk-oriented, but can also be run in
main memory for volatile document processing. A major focus is set on a memory sav-
ing representation of the original document, so the chosen encoding consists of just the
essential node attributes which are needed to allow a complete and yet quick traversal of
all XPath axes. In contrast to other approaches such as [Bo02], the table representation is
schema-oblivious, i.e., no DTD or XML Schema is required to encode a document. The
integrated XPath processor applies query algorithms that are partly derived from the

1 Homepage and download of BaseX:
http://www.inf.uni-konstanz.de/dbis/research/baseX
Staircase Join operator [Gr03]. Very similar algorithms are in fact used to build the
treemap visualization and to allow user interactions in interactive response time.

Section 2 starts with an overview of the applied tree encoding and introduces some sam-
ple query algorithms. In Section 3, the treemap visualization and its on-the-fly
construction and interaction techniques are presented. Section 4 summarizes our contri-
bution and gives a brief outlook on future work.

2 Tree Encoding

XML documents can be represented as an ordered set of nodes. All nodes can have nu-
merous attributes, such as a unique node ID, references to parent and children nodes,
node kinds (element/text), tag names or text content, number of descendant nodes, etc.
Some of these attributes are redundant, e.g., references to child nodes and the childrens’
parent references. Hence, it is advisable to skip such information as long as two demands
are still met:

- the original document can be reconstructed
- query efficiency is preserved

Storing three attributes actually suffices to restore XML instances and to process all
XPath axes (see Fig. 1): a reference to the parent ID, the node kind and a string refer-
ence\footnote{note that special XML features such as namespaces or processing instructions are disregarded in this paper
for the sake of simplicity.}. The node ID is implicitly given by the table position, and the string is further
numerically encoded, pointing to a kind specific text index. Instead of the absolute par-
ent ID, the distance to the parent is stored; this is especially beneficial for performing
updates. Due to some characteristics of the stored table attributes, the resulting represen-
tation can be compressed by merging attributes together [Gr06]. Thus, each tuple is
internally stored in just eight bytes\footnote{tuple size grows to 16 bytes for namespace support and node IDs > 2^{32}.}. XML attributes are represented in the table as well;
they consist of a name and value combination. As the number of different attribute

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure1}
\caption{Mapping of an XML document.}
\end{figure}

left: original XML document, center: tree representation, right: table encoding
names and the distance to the element (parent) node is small enough, they can also be stored together in mentioned eight bytes.

The fixed and compact size of the nodes allows for a very efficient and straightforward access in main memory as well as on disk. All table operations that sequentially traverse tuples additionally benefit from common prefetching strategies of hard disks. Node processing is shown here for the descendant axis: starting a sequential scan at node $x$, all following nodes are regarded as descendant nodes. As soon as the first node $y$ is encountered with $\text{Par}(y) < \text{ID}(x)$, all descendant nodes have been found, and further node traversal is skipped. If descendants of multiple nodes are found, pruning and partitioning concepts are applied to preserve linear complexity of traversal (see [Gr03] for pruning, partitioning and skipping XML tuples based on the $\text{Pre/Post}$ encoding).

To accelerate value- and text-based queries, indexes can be constructed for attribute values and text nodes. Queries are internally rewritten to first access the indexes before the remaining XPath is traversed in a backward manner. As an example, in the query
\[
//\text{address[@town="Washington"]}
\]
all attribute nodes matching the value "Washington" are assembled before the attribute name town and the parent address tag is evaluated. This often leads to a speedup of several orders of magnitude and is especially helpful for content-based queries as they are offered in our visualization (see Fig. 2).

Figure 2: TreeMap Visualization. left: complete view with focused rectangle, right: zoomed view

3 TreeMap

The space-filling TreeMap was initially introduced by [JS91]; its data structure mainly consists of a rectangle array. The implemented layout algorithms change the rectangle orientation as soon as more vertical than horizontal space is given and vice versa. Colors are used to visualize the tree depth of a node. Indeed, the algorithms needed for hierarchic visualizations – such as the TreeMap – are very similar to the applied query algorithms for traversing XPath axes. A sequential scan of the node table allows a straightforward calculation of the rectangles to be painted. The calculation is restricted to the given screen space, i.e., as soon as rectangles become too small to be displayed, they are skipped. As rectangles are added sequentially, they have the same order as the node
table. This implies descendants to always be arranged after their parents. This property comes in handy if, for instance, the according rectangle beneath the current mouse position is searched to be highlighted (see Fig. 2, left side). It is sufficient to traverse the array in reverse order and to terminate at the first rectangle which contains the coordinates of the mouse cursor as this is always the smallest rectangle displayed.

Two general interactions are offered to change the currently shown nodes. If a rectangle is selected via double click, a new array with its descendant nodes is calculated and displayed (zooming, shown in Fig. 2, right side). Secondly, a text field is offered in which tag names, text nodes and attributes may be entered by the user. The input is interactively converted to XPath after each key event, and the results of the executed query are highlighted in the visualization. Content queries clearly profit from activated indexes. Next, a filter operation allows to recalculate the TreeMap for the currently highlighted nodes, and a history function, as integrated in common internet browsers, allows to return to the previously shown views.

4 Main Contributions

BaseX is a database prototype, working on a compact table based representation of XML documents. The fixed node size is equally suitable for main and secondary storage. Originally supposed to demonstrate the efficiency of the underlying data structure, the TreeMap additionally allows a visual access to the represented documents. With the integrated query interface, users can filter query results in interactive response time. As the applied encoding has also been designed to allow update operations such as the deletion or insertion of XML contents, major future work will now be the integration of update capabilities and their visual support, realized as drag & drop operations. Moreover, the text indexes will be enhanced to support a broader range of full text requests.

Literaturverzeichnis

GI-Edition Lecture Notes in Informatics


P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Language to Information Systems, NLDB’2001.


P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.


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-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.


P-12 Martin Glinz, Günter Müller-Luchsnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni (Hrsg.): The Changing Universities – The Role of Technology.


P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Busi-ness 2002.


P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.


P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit.


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: Biometric 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.): DelFli 2003: e-Learning Fachtagung Informatik
P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationsysteme – 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.): Persistence, Scalability, Transactions – Database Mechanisms for Mobile Applications
P-44 Jan von Knop, Wilhelm Havercamp, Eike Jessen (Hrsg.): Security, E-Learning, E-Services
P-45 Bernhard Rumpe, Wolfgang Hesse (Hrsg.): Modellierung 2004
P-46 Ulrich Fleigel, 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 Daten sicherheit – 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.): DELFli 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 Havercamp, Eike Jessen (Hrsg.): E-Science and 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 Schweigert (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 informatische 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 its 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 Kowalczyk, 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.): Marktmit Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Hauekamp, 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-Ries, Key Poussutchi, 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. Luxi, P. Eiermann (Hrsg.): Land-und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Haeumer, 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 Hunos, 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, Benoit 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 Nédon (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)

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