B. Mitschang et al. (Hrsg.): BTW 2017 – Workshopband,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 321

Duplikaterkennung in der Graph-Processing-Platform GRADOOP

Florian Pretzsch[1]

Abstract: Die zunehmende Bedeutung von Graphdaten im Kontext von Big Data erfordert wirksame Verfahren zur Erkennung von Duplikaten, d. h. Knoten, welche das selbe Realweltobjekt repräsentieren. Dieser Beitrag stellt die Integration von Techniken zur Duplikaterkennung innerhalb des Graphverarbeitungs-Frameworks GRADOOP vor. Dazu werden dem GRADOOP-Framework neue Operatoren zur Duplikaterkennung hinzugefügt, die u. a. in der Lage sind, Ähnlichkeiten zwischen Knoten von einem oder mehreren Graphen zu bestimmen und ermittelte Duplikate als neue Kanten zu repräsentieren. Das vorgestellte Konzept wurde prototypisch implementiert und evaluiert.

Keywords: GRADOOP, Duplikaterkennung, Similarity, Blocking, Lastbalancierung, Graphen

1 Einleitung

Die graphbasierte Speicherung und Verarbeitung großer Datenmengen gewinnt zunehmend an Bedeutung, z. B. zur Analyse großer sozialer Netzwerke [Cu13] oder im Bereich Business Intelligence [Gh15]. Die Verarbeitung großer Graphen hat eine Vielzahl von Forschungsarbeiten hervorgebracht, u. a. zur Speicherung von Graphen in Graphdatenbanken (z. B. Neo4J [neo13]) oder verteilte Systeme zur Definition und Ausführung von Graphalgorithmen (z. B. Pregel [Ma10]). Analog zum Data Warehousing benötigt die effektive und effiziente Graphanalyse jedoch die Unterstützung eines kompletten Prozess von der Erstellung eines Graphen, seiner Verarbeitung und der Analyse innerhalb eines Workflows. Das Graph-Analyse-Framework GRADOOP (Graph analytics on Hadoop) [Ju15] unterstützt die Definition solcher Workflows, welche mittels existierender Big Data Technologien (u. a. Apache Flink und HBase) effizient und verteilt ausgeführt werden können. Die Definition von Workflows erfolgt dabei unter Verwendung vordefinierter Operatoren.

Neben der Erstellung und Verknüpfung von Graphen müssen solche Workflows Vorkehrung zur Sicherung der Datenqualität treffen, damit aufbauende Analysen valide sind. Ein wesentlicher Schritt zur Sicherung der Datenqualität ist Duplikaterkennung [FS69], d. h. das automatische Identifizieren von Knoten in einem oder mehreren Graphen, welche das selbe Realweltobjekt repräsentieren. Eine Vielzahl von Forschungsarbeiten beschäftigen sich mit der automatischen Erkennung von Duplikaten (siehe z. B. [KR10] für einen Vergleich von Frameworks), u. a. in einem verteilten Shared-Nothing-Cluster [KTR12].

Dieser Beitrag stellt eine Konzept zur Duplikaterkennung innerhalb des GRADOOP-Frameworks vor. Dazu wurden neue GRADOOP-Operatoren für die Duplikaterkennung

[1] Institut für Kommunikationstechnik, HfTL, Gustav-Freytag-Str. 43-45, 04277 Leipzig, s06626@hft-leipzig.de

konzipiert und implementiert, welche in GRADOOP-Workflows (gemeinsam mit den bisherigen Operatoren) verwendet werden können. Die neuen Operatoren erweitern GRADOOP um die Möglichkeit zur Berechnung von Ähnlichkeiten zwischen Knoten, die mit Kanten verbunden werden, anhand derer dann eine Selektion von Duplikatpaaren durchgeführt werden kann. Auch soll es möglich sein, zur Ähnlichkeitsberechnung Nachbarknoten mit einzubeziehen. Der Fokus dieser Arbeit liegt jedoch auf der Ähnlichkeitsberechnung anhand von Knotenattributen. Die rechenintensive Ähnlichkeitsberechnung kann dabei analog zu [KTR12] auf verschiedene Server verteilt werden, um eine gleichmäßige Auslastung aller Server zu erreichen (Lastbalancierung). Zur Repräsentation von Duplikaten werden Kanten zwischen denen als Duplikat erkannten Knoten eingefügt, so dass das Ergebnis der Duplikaterkennung für die weiteren Schritte des Graph-Analyse-Workflows (z. B. Datenfusion, Berechnung abgeleiteter Werte) zur Verfügung steht.

Abschnitt 2 stellt die Grundlagen der Duplikaterkennung sowie von GRADOOP vor, ehe Abschnitt 3 das Konzept der neuen Operatoren vorstellt. Abschließend beschreibt Abschnitt 4 die prototypische Implementation inklusive einer kurzen Evaluation.

2 Grundlagen

2.1 Duplikaterkennung

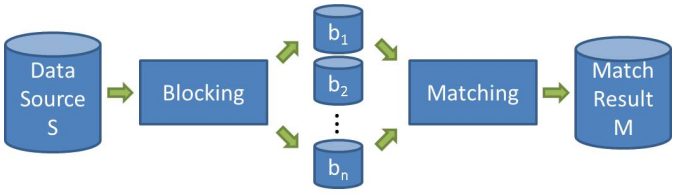

Abb. 1: Schematische Darstellung der Duplikaterkennung [Ko14]

Das Problem der Duplikaterkennung (u. a. auch bekannt als Entity/Object Matching) wurde 1969 von Felligi und Sunter formuliert [FS69]. Ziel ist es, gleiche Objekte trotz ihrer unterschiedlichen Repräsentation in einer oder mehreren Datenquellen effizient zu identifizieren. Abbildung 1 zeigt die schematische Darstellung der Duplikaterkennung innerhalb einer Datenquelle. In einem ersten Schritt (Blocking) werden aus Performancegründen die Datenobjekte in (ggf. überlappende) Blöcke eingeteilt, so dass die anschließende Duplikaterkennung (Matching) nur innerhalb der jeweiligen Blöcke realisiert wird. Beispielsweise könnten in einem Graphen, der Bücher und Autoren beinhaltet, die Datenobjekte nach diesen beiden Kriterien unterteilt und innerhalb der Blöcke nach Duplikaten gesucht werden. Die meisten Verfahren [KR10] setzen beim Matching auf Ähnlichkeitsberechnungen mittels Attributwerten sowie Kontextinformationen, anhand derer eine Gesamtähnlichkeit im Intervall [0,1] bestimmt wird. Ein Wert von *0* würde dabei völlige Unähnlichkeit ausdrücken, während *1* für totale Übereinstimmung (Gleichheit) steht. Der Einsatz eines Blocking-Verfahrens reduziert damit im Allgemeinen die Anzahl der notwendigen Ähnlichkeitsberechnung deutlich. Das finale Ergebnis (Match Result) ergibt sich aus einer

Filterung der paarweisen Ähnlichkeitswerte. Ein typisches Beispiel ist die Verwendung aller Datenobjektpaare, deren Ähnlichkeit über einem definierten Schwellwert liegt.

2.2 GRADOOP-Framework

GRADOOP unterstützt die effiziente Integration, Analyse und Repräsentation von Graphenstrukturen [Ju15]. Basierend auf dem *Extended Property Graph Data Model* (EPGM), welches Graphen mit Knoten, Kanten und Attributen abbildet, werden Operatoren zum Analysieren einzelner Graphen und Verarbeiten von Graph-*Collections* zur Verfügung gestellt. Neben einer domainspezifischen Sprache (*Graph Analytical Language* – GrALa) zur Definition von Workflows ist ein weiteres Merkmal die verteilte Speicherung der Graphen in HBase, wodurch mittels Hadoop die parallele Abarbeitung der Workflows auf unterschiedlichen Servern ermöglicht wird.

Abbildung 2 (links) zeigt die elementaren Schritte eines Ende-zu-Ende-Graphanalyse-Workflows, welche die Integration der Daten aus heterogenen Quellen in ein Graphdatenformat (*Data Integration*), die Graphanalyse mittels Operatoren (*Graph Analytics*) und die grafische Darstellung der Ergebnisse (*Representation*) umfasst. Die Duplikaterkennung wird demnach in der Phase *Data Integration* durchgeführt.

Das Framework ist in Java programmiert und nutzt Apache Flink als Datenstreaming-Engine. Die Architektur von GRADOOP ist in Abbildung 2 (rechts) dargestellt. Neben den bereits genannten Hauptkomponenten HBase (*Distributed Graph Data Store*) und EPGM stellt die *Operator Implementations*-Komponente die wesentlichen Operatoren zur Datentransformierung und Ergebnisausgabe zur Verfügung. Die *Workflow Execution* ermöglicht die parallele Ausführung von Datenintegrations- und Analyse-Workflows. Über die *Operator Implementation*-Komponente wird die Ausführung überwacht und stellt Ausführungsinformationen für die Benutzer zur Verfügung. Die oberste Schicht (*Workflow Declaration* und *Representation*) der Architektur stellt eine Umgebung zur deklarativen bzw. visuellen Erstellung und Definition von Workflows bereit. Des Weiteren werden hier die Ergebnisse als Graph visualisiert oder in Form von Tabellen und Diagrammen dargestellt.

GRADOOP stellt eine Reihe von Operatoren zur Verfügung, die ein oder zwei Graphen bzw. Graph-*Collections* verarbeiten. Beispielsweise kombiniert der Operator *Combination* zwei Graphen zu einem Graphen. In GrALa-Notation schreibt man dafür `result = graph1.combine(graph2)`, wodurch aus den zwei Graphen (`graph1` und `graph2`) der Graph (`result`) als Rückgabewert zurückgeliefert wird. Eine vollständige Liste aller Operatoren ist in [Ju15] zu finden.

3 Konzeption

3.1 Workflow

Abbildung 3 zeigt den schematischen Workflow zur Duplikaterkennung mit GRADOOP, welcher sich an dem allgemeinem Verfahren zur Duplikaterkennung (siehe Abbildung 1)

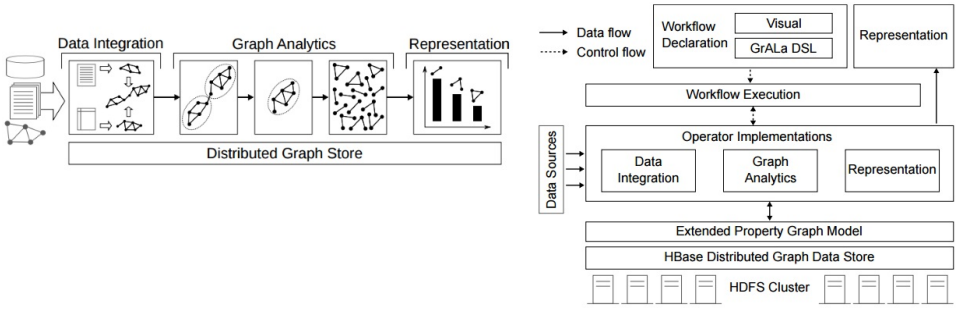

Abb. 2: GRADOOP: Schematische Darstellung eines Workflows (links) sowie der Architektur (rechts) [Ju15]

orientiert. Die Eingabe sind zwei Graphen, in denen nach Duplikaten bei den Knoten gesucht wird. Als Ergebnis entsteht ein gemeinsamer Graph, der beide Eingabegraphen enthält und um Kanten zwischen als Duplikat erkannten Knoten ergänzt wird.

Der Workflow ist modular aufgebaut und die einzelnen Schritte werden in separaten, parametrisierten Operatoren ausgeführt, welche (neben Operator-spezifischen Parametern) als Ein- und Ausgabedaten Graphen bzw. Graph-*Collections* verwenden. Dadurch wird eine nahtlose Integration in das GRADOOP-Konzept erreicht, da bestehende GRADOOP-Operatoren wiederverwendet werden können (z. B. *Union*) und neu eingeführte Operatoren auch in anderen Workflows zur Anwendung kommen können. Gleichzeitig kann der gesamte Workflow als ein *großer* Operator gekapselt werden. Die einzelnen Operatoren werden zur Laufzeit mit Hilfe von Apache Flink-Operatoren abgebildet und umgesetzt. Die effiziente und effektive Verarbeitung sowie Verteilung auf die verschiedenen Serverknoten übernimmt dabei Apache Flink abhängig von der Anzahl der Server und konfigurierten Parallelität.

Der Workflow beginnt mit dem *Blocking*-Operator, der die zwei Eingabe-Graphen entgegen nimmt. Als Rückgabe werden die anhand eines Blockschlüssels gruppierten Knoten innerhalb einer Graph-*Collection* als eigener Subgraph je Block zurückgegeben, die sich ggf. auch überlappen können. Die einzelnen Graphen der Graph-*Collection* werden im parametrisierten *Similarity*-Operator weiterverarbeitet. Die Ähnlichkeitsberechnung von Knoten, die als Entitäten angesehen werden, erfolgt z. B. anhand der String-Ähnlichkeit der Werte gleichnamiger Knotenattribute. Das Ergebnis der Ähnlichkeitsberechnung wird als Kante zwischen den zwei beteiligten Knoten dargestellt, wo der Ähnlichkeitswert als Attribut der Kante hinzugefügt wird. Die Vorteile der Kantenrepräsentation sind, dass Duplikate im weiteren Verlauf der Workflows leicht erkannt werden können und dass der Graph gesamtheitlich weiterhin besteht, ohne das Informationen verloren gehen. Im anschließenden *Selection*-Schritt werden die paarweisen Ähnlichkeitswerte gefiltert und das Match Result als Graph-*Collection* zurückgegeben. In zukünftigen Arbeiten kann hier auch eine Informationsfusion erfolgen, d. h. als Duplikat erkannte Knoten zu einem (Super-)Knoten zusammengeführt werden. Die Rückgabe-*Collection* kann durch den bereits existierenden *Union*-Operator mit den beiden Eingangsgraphen beim *Blocking*-Operator zu einem Ergebnisgraph zusammengefügt werden.

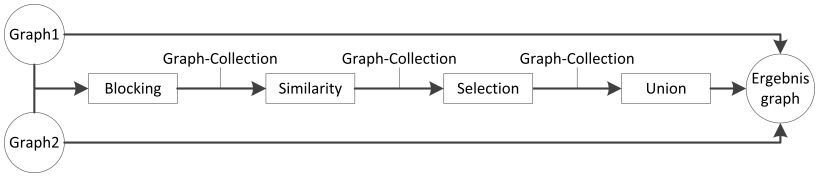

Abb. 3: Konzeptioneller Entwurf des Operator-basierten Workflows zur Duplikaterkennung

Operator	Parameter	Mögliche Werte
Blocking	Angabe des Blocking-Verfahrens	kein Blocking, Attributwert, Blockschlüssel
	Angabe eines Blocking-Attributs	*name*
	Angabe des Knotentyp-Mappings	originaler Knotentyp
	Definition Lastbalancierung je Knotentyp	ja, nein
Similarity	Ähnlichkeitsalgorithmus	Ähnlichkeitsmaß, z. B. Jaro-Winkler
	Angabe von Vergleichsattributen und Schema-Mapping je Knotentyp	Paare von Attributnamen (Standard: alle gleichnamigen Attribute)
	Anzahl Tiefe Nachbarschaftsknoten	0 (= keine Nachbarknoten) oder höher
Selection	Definition Selektionsverfahren	schwellwertbasiert (0<=s<=1), Top-k (k>0)
	Label	*duplicate*
	Kantenattribut	*value* = "Ähnlichkeitswert"
Union	*parameterlos*	

Tab. 1: Übersicht Teil-Operatoren und zugehörige Parameter

3.2 Parametrisierung

Die Operatoren können durch Parameter an die spezifischen Daten (Domäne) angepasst werden, um z. B. zu definieren, für welche Knotentypen Duplikate erkannt werden sollen und welche Ähnlichkeitsfunktion verwendet werden soll. Tabelle 1 gibt für jeden Operator mögliche Parameter sowie ggf. beispielhafte Werte an.

Der Blocking-Operator unterstützt die Bildung von Blöcken anhand eines Knotentyps und/oder eines Knotenattributs (alle Knoten mit dem gleichen Typ bzw. Attributwert bilden einen Block) sowie eines Blockschlüssels, der sich aus verschiedenen Attributwerten zusammensetzt. Dabei ist es möglich, für jeden Knotentyp ein individuelles Blocking zu ermöglichen, d. h. z. B. Autoren werden nur nach Knotentyp gruppiert, während die Publikationen mittels eines zusammengesetzten Blockschlüssels in Blöcke eingeteilt werden. Da es vorkommen kann, dass die beiden Eingangsgraphen für inhaltlich gleiche Knoten unterschiedliche Knotentypbezeichnungen verwenden, kann die Angabe des Knotentyps um ein Typ-Mapping erweitert werden. Dabei kann angegeben werden, welcher Knotentyp aus dem ersten Graphen zu welchem Knotentyp des zweiten Graphen korrespondiert. Zusätzlich kann für den Blocking-Operator eine Untergruppierung angegeben werden, welche eine Lastbalancierung ermöglicht (Details siehe Abschnitt 3.3).

Zur Ähnlichkeitsberechnung müssen die zu vergleichenden Attribute je Knotentyp angegeben werden. Dabei kann die Bezeichnung der Attribute unterschiedlich sein, wodurch ein Schema-Mapping erforderlich wird. Hierbei werden Paare von den zu vergleichenden Attributen beider Knoten angegeben. Standardmäßig sollen alle Attribute verwendet

werden, die die gleiche Bezeichnung haben und in beiden Knoten vorkommen. Auch die Angabe, welcher Vergleichsalgorithmus für welches Attribut verwendet werden soll, kann realisiert werden, da die verschiedenen Vergleichsalgorithmen [GM13] unterschiedlich gut geeignet sind. So ist z. B. für Namen die Jaro-Winkler-Ähnlichkeit [Wi06] und für längere Zeichenketten die Jaccard-Ähnlichkeit geeignet. Des Weiteren kann die Tiefe der Nachbarschaft, also alle Nachbarknoten innerhalb eines Abstands vom zu vergleichenden Knoten, die dann zur Ähnlichkeitsberechnung mit herangezogen werden sollen, angegeben werden.

Die Angabe des Selektionsverfahren ist ebenfalls über einen Parameter steuerbar. Standardmäßig wird ein schwellwertbasiertes Verfahren verwendet, welches alle Duplikatkanten mit einem Ähnlichkeitswert über einem Schwellwert (z. B. 90%) als Duplikatkante identifiziert. Die Eigenschaften der Duplikatkanten (Name, Gewicht) sind ebenfalls konfigurierbar.

3.3 Blocking & Lastbalancierung

Durch das Blocking soll der Suchraum durch Einteilung in Gruppen (Blöcke) reduziert werden, da die Anzahl der paarweisen Vergleiche quadratisch mit der Anzahl der Knoten ansteigt (siehe dazu [Ko14], S. 58). Die Aufteilung in Gruppen erlaubt es, dass die Ähnlichkeitsberechnungen nur noch zwischen Knoten der selben Gruppen durchgeführt werden müssen.

Anhand eines Attributs *gender* könnten bspw. in Personengraphen die Knoten nach dem Geschlecht gruppiert werden. Problem ist, dass nicht jeder Knoten die gleichen Attribute enthalten muss. Durch einen Knotentyp, den jeder Knoten in GRADOOP besitzt, könnten Knoten in einem Autorengraphen nach Autoren, Büchern und Verlagen gruppiert werden. Die Anzahl der Gruppen ist hier jedoch eingeschränkt. Die Blockanzahl erhöht werden kann durch einen zusammengesetzten Blockschlüssel, bestehend aus dem Knotentyp sowie einem einheitlichen Attribut innerhalb des jeweiligen Knotentyps. Zum Beispiel könnte die Autorengruppe durch Hinzufügen des Anfangsbuchstaben vom Namen um den Faktor 26 vergrößert werden. Des Weiteren könnten Knoten anhand von Nachbarknoten eingeteilt werden, um bspw. alle Bücher einer Gruppe zuzuweisen, die den gleichen Autor besitzen.

Tabelle 2 stellt die einzelnen Verfahren anhand bestimmter Merkmale gegenüber. Durch das Blocking werden potenziell weniger Duplikate gefunden, falls gleiche Knoten in unterschiedlichen Gruppen einsortiert werden. Ein auf die Daten angepasstes Blockingverfahren ist dabei entscheidend.

Neben der Reduzierung der Anzahl der paarweisen Vergleiche ermöglicht Blocking auch die effiziente Ausführung eines Workflows in einer verteilten Cloud-Umgebung. Abbildung 4 illustriert die Ausführung schematisch. Nach der Partitionierung der Eingangsdaten durch Blocking ist eine parallele (d. h. unabhängige) Bearbeitung der Blöcke voneinander möglich.

Insbesondere wenn Attributwerte sowie Knotentypen, welche zum Blocking herangezogen werden, sehr ungleich verteilt sind (Data Skew), entstehen jedoch (wenige) große Blöcke, die einer effizienten Verarbeitung in einem Cluster entgegenstehen. Da die Bearbeitung

Merkmal	Möglichkeiten Blocking			
	Knotenattribut	**Knotentyp**	**zusammengesetzter Blockschlüssel**	**ohne Blocking**
Anzahl Gruppen	mittel (abhängig von Wertebereich)	wenig	hoch	eine
Reduction Ratio	mittel	niedrig (bei Ungleichverteilung von Knotentypen sehr niedrig)	hoch	keine
Pairs Completeness	hoch, abhängig von Attribut	hoch	mittel	maximal

Tab. 2: Möglichkeiten für das Blocking im GRADOOP-Umfeld

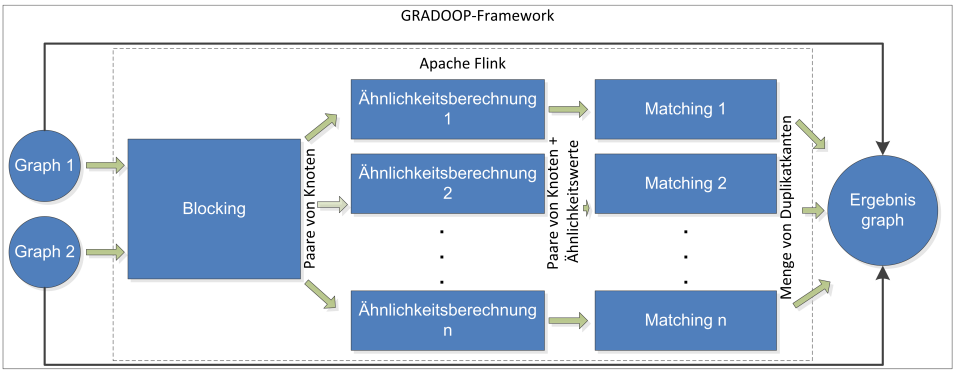

Abb. 4: Schematische Darstellung der parallelen Ausführung von Dupliklaterkennungs-Workflows mit Apache Flink

eines Blocks (paarweiser Vergleich) auf einem Serverknoten erfolgt, kann ein Data Skew zu erheblich unterschiedlichen Laufzeiten auf den einzelnen Servern führen. Basierend auf den Arbeiten zu Dedoop unterstützt der Blocking-Operator eine Untergruppierung ähnlich der in Dedoop vorgestellten Lastbalancierungsstrategie BlockSplit. [KTR12]

Dabei werden (große) Blöcke logisch in disjunkte Teilblöcke (Untergruppen) zerlegt, die dann paarweise miteinander verglichen werden. Abbildung 5 zeigt das Konzept für einen Block, welcher in vier Teilblöcke unterteilt wird. Die Knoten des Blocks aus dem ersten Graphen (A) sowie des zweiten Graphen (B) werden zunächst in insgesamt vier (gleichgroße) Hilfsgruppen zerlegt, die dann paarweise zu Untergruppen zusammengeführt werden. Damit wird sichergestellt, dass weiterhin alle Knotenpaare eines Blocks miteinander verglichen werden, aber nicht Knoten aus dem gleichen Graphen zusammengefügt werden. Das vorgestellte Verfahren wird durch die Angabe der Anzahl der Hilfsgruppen pro Block definiert und stellt eine vereinfachte Variante der BlockSplit-Strategie [KTR12] dar, welche die Zerlegung anhand der Blockgröße durchführt (und damit nur große Blöcke zerlegt). Die Übertragung der Lastbalancierungsstrategien aus [KTR12] ist Teil zukünftiger Arbeiten.

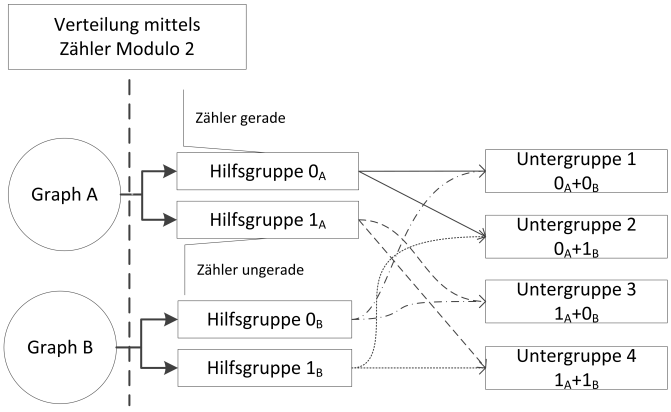

Abb. 5: Gruppierung in vier Untergruppen

4 Umsetzung & Evaluation

Die prototypische Implementierung der Konzeption aus Kapitel 3 erfolgte als *duplicate-linking*-Operator im GRADOOP-Framework. Der Operator wurde zur Demonstration der Funktionsweise monolithisch implementiert. Zur Evaluation der korrekten Funktionsweise und Skalierbarkeit des Operators wurden folgende Größen nach der prototypischen Umsetzung überprüft:

- Qualität der Duplikaterkennung (Precision, Recall und F-Measure)
- Laufzeitverhalten bei Vergrößerung der Taskanzahl sowie der Datengröße
- Laufzeitverhalten bei Untergruppierung (Lastbalancierung)

Als Datenquelle wurden analog zu [KTR10] Publikationsdaten der DBLP verwendet, welche Publikationen und Autoren im Bereich der Informatik auflistet. Für die Ermittlung von Precision und Recall wurde das manuell erstellte Match-Result[3] verwendet. Für die Messungen wurde aus DBLP für jede Publikation, wenn die Daten vorhanden waren, ein Verlags-Knotentyp (*publisher*), die beteiligten Autoren (*author*) und die Publikation selbst (*publication*) erstellt. Daraus sind zwei Graphen entstanden. Die Tabelle 3 zeigt die Eigenschaften der resultierenden Test-Graphen, welche – bis auf die Messung mit Erhöhung der Datensatzanzahl – als Testgrundlage verwendet wurden. Die Testgraphen sind so gewählt, dass nur in den Publikationen Duplikate vorhanden sind, da diese in beiden Graphen existieren. Für die Knotenvergleiche werden bei den Laufzeituntersuchungen alle gleichen Attribute verwendet.

In den folgenden Abschnitten wurden verschiedene Testläufe durchgeführt. Die Messungen sind mindestens zwei mal durchgeführt und jeweils der Durchschnitt der Laufzeiten

[3] http://dbs.uni-leipzig.de/de/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution/, Zugriff: 10.04.2016

Eigenschaft	Graph g1	Graph g2
Anzahl Knoten	35173	3616
Anzahl Autoren	19543	0
Anzahl Publikationen	15630	3597
Anzahl Verleger	0	19
Anzahl Matches	**40012**	

Tab. 3: Eigenschaften der DBLP-Graphen `g1` und `g2`

Standardmaße	Schwellwert						
	80 %	**85 %**	**90 %**	**92,5 %**	**95 %**	**97,5 %**	**100 %**
Precision	30,5 %	55,4 %	70,8 %	74,5 %	77,3 %	77,1 %	77,5 %
Recall	81,2 %	78,9 %	74,7 %	71,5 %	68,2 %	62,0 %	46,9 %
F-Measure	44,3 %	65,1 %	72,7 %	73,0 %	72,4 %	68,8 %	58,5 %
Laufzeit	85 s	85 s	85 s	82 s	82 s	82 s	83 s

Tab. 4: Standardmaße mit Blocking bei verschiedenen Schwellwerten (Standard-Hardware)

Standardmaße	Schwellwert						
	80 %	**85 %**	**90 %**	**92,5 %**	**95 %**	**97,5 %**	**100 %**
Precision	30,9 %	55,5%	70,9 %	74,5 %	77,3 %	77,1 %	77,8 %
Recall	84,9 %	79,5 %	75,1 %	71,8 %	68,4 %	62,1 %	47,8 %
F-Measure	45,3 %	65,4 %	72,9 %	73,2 %	72,6 %	68,8 %	59,2 %
Laufzeit	1091 s	1079 s	1081 s	1085 s	1067 s	1082 s	1087 s

Tab. 5: Standardmaße ohne Blocking bei verschiedenen Schwellwerten (Standard-Hardware)

bestimmt wurden. Für die einzelnen Testläufe kam zu Demonstrationszwecken die gleiche Standard-Hardware[4] mit vier CPU-Kernen(== vier Cores) zum Einsatz. Falls nicht anders angegeben, wurden die Testläufe ohne Parameter mit vorkonfigurierten Einstellungen des Operators durchgeführt.

4.1 Standardmaße

In der Tabelle 4 sind die Standardmaße mit steigendem Schwellwert abgebildet, die bei der Ausführung des Operators mit Blocking (Knotentyp+Anfangsbuchstabe vom Namen) erzielt wurden. In Tabelle 5 wurden die Untersuchungen ohne Blocking durchgeführt. Deutlich ist in den Tabellen 4 und 5 zu erkennen, dass mit steigendem Schwellwert der Precision-Wert wächst und der Recall-Wert reduziert wird. Der F-Measure-Wert hat seinen maximalen Wert von ca. 73% bei 92,5%.

Demzufolge wird dieser Schwellwert als Standard hinterlegt. Der Höchstwert bei der Precision und gleichzeitig der niedrigste Wert des Recalls liegt bei einem Schwellwert

[4] CPU: Intel Core i5-4300M, 2,6 GHz; Arbeitsspeicher: 8 Gigabyte

von 100 %. Durch den Vergleich der Attribute können nicht alle enthaltenen Duplikate gefunden werden und der Anteil falsch erkannter Duplikate ist dabei am größten. Die geringen Laufzeitschwankungen in Tabelle 4 sind Messungenauigkeiten. Die Laufzeit im Vergleich zu Tabelle 5 beträgt nur ca. 7,8 % bei fast gleichen Standardmaßen.

4.2 Skalierbarkeit

Die Skalierbarkeitsuntersuchung bei Erhöhung der Taskanzahl wurde mit den Graphen aus Tabelle 3 durchgeführt und in Abbildung 6 zusammenfassend dargestellt. Bei der Berechnung ohne Blocking wurde der *Cross*[5]-Operator verwendet und schrittweise die Taskanzahl erhöht sowie die Ausführungszeit gemessen. Dies ist die Zeit für die Erstellung des Ergebnisgraph sowie der Zählung von Kanten und Knoten. Außerdem wurde das Blocking anhand des Knotentyps und mit zusammengesetzten Blockschlüssel (Knotentyp+Anfangsbuchstabe vom Namen) untersucht.

Die Ergebnisse zeigen, dass die Laufzeit mit Erhöhung der Taskanzahl bei allen Varianten reduziert wird und damit skaliert. Annähernd ausgeglichen sind die Tasklaufzeiten ohne Blocking, sodass damit der größte Laufzeitgewinn erzielt wurde. Der *Cross*-Operator von Apache Flink skaliert dabei effizient. Bei der Variante mit einem zusammengesetzten Blockschlüssel ist die Laufzeit am geringsten, jedoch sind die Tasklaufzeiten nicht so ausgeglichen. Hier verteilt Apache Flink mittels *CoGroup*-Operator die Gruppen nicht optimal. Beim Blocking anhand des Knotentyps reduziert sich die Laufzeit durch eine geringere Paaranzahl im Vergleich zur Variante ohne Blocking. Die Vergleiche werden nur durch eine Task durchgeführt und muss daher optimiert werden. Durch das Blocking mit einem zusammengesetzten Blockschlüssel werden rund 8100 Duplikatpaare gefunden, was ca. 20% der Duplikate der anderen beiden Varianten entspricht. Ist die Anzahl von Tasks größer als die Anzahl zur Verfügung stehender CPU-Kerne, steigen durch einen Overhead die Laufzeiten wieder an. Die Anzahl der Tasks sollte im Optimalfall genauso groß eingestellt werden, wie die Anzahl zur Verfügung stehender CPU-Kerne. Des Weiteren sollte eine Untergruppierung bei beiden Blocking-Varianten angewendet werden, um die Lastbalancierung zu optimieren.

4.3 Laufzeitverhalten bei Untergruppierung

Zur Demonstration der Lastbalancierung wurde nur der parametrisierte Operator getestet. Je Durchlauf wurde die Anzahl b der Gruppen verändert sowie die Einstellung der zu untersuchenden Blockingvariante. Für die Taskanzahl von vier ist das Ergebnis in Abbildung 7 dargestellt. Als Referenzwert ist die jeweilige Laufzeit der Blocking-Varianten ohne Untergruppierung bei gleicher Taskanzahl eingezeichnet.

Erkennbar an den Verläufen ist, dass die Gesamtlaufzeit im Vergleich zur Variante ohne Untergruppierung bis auf $b = 1$ reduziert werden kann. Der Grund für die etwas größeren

[5] *DataSet Transformations - Operatorenübersicht.* https://ci.apache.org/projects/flink/flink-docs-release-1.1/apis/batch/dataset_trans-formations.html, Zugriff: 14.10.2016

Laufzeiten bei $b = 1$ ist der Umstand, dass mehr Prüfungen beim Blocking und der Ähnlichkeitsberechnung durchgeführt werden müssen, die sich jedoch bei $b > 1$ als vorteilhaft erweisen. Des Weiteren ist speziell beim Blocking anhand des Knotentyps die Gesamtlaufzeit auf ca. 33% im Vergleich zur Variante ohne Untergruppierung zurückgegangen. Beim zusammengesetzten Blockschlüssel ist der Laufzeitgewinn geringer, da der Overhead einen größeren Anteil an der Gesamtlaufzeit hat.

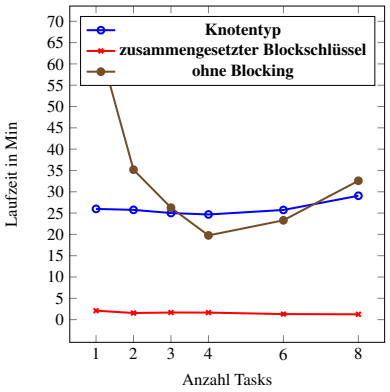

Abb. 6: Laufzeitverhalten bei Erhöhung der Taskanzahl (Standard-Hardware)

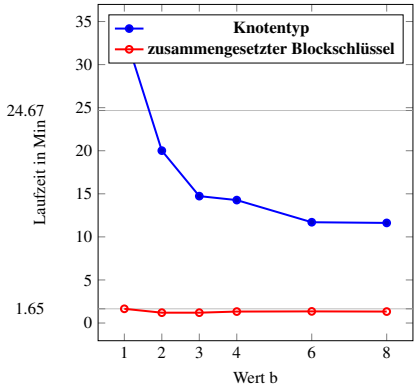

Abb. 7: Laufzeitverhalten bei Erhöhung der Blockanzahl durch Untergruppierung (Standard-Hardware, Taskanzahl = 4)

4.4 Laufzeitverhalten bei Erhöhung der Datensatzanzahl

Für die Laufzeituntersuchung mit Erhöhung der Datensatzanzahl wurden die Varianten ohne Blocking mittels *Cross*-Operator und parametrisierten Operator sowie die Varianten mit Blocking anhand des Knotentyps und mit einem zusammengesetzten Blockschlüssel jeweils mit der gleichen Anzahl an Datensätzen gegenübergestellt. Bei den Blocking-Varianten kam eine Untergruppierung mit $b = 4$ zum Einsatz, die übrigen Parameter blieben gleich. Als Datensatz sei ein Knoten innerhalb der Graphen anzusehen und die Gesamtzahl ist demnach die Summe der Knoten beider Eingangsgraphen, die jeweils ca. um den Faktor 10 vergrößert wurde. Die Ergebnisse sind in Tabelle 6 dargestellt, wobei die Werte in Klammern bei der Variante ohne Blocking die entsprechenden Ergebnisse des parametrisierten Operators sind.

Ersichtlich ist, dass die Laufzeiten unterschiedlich stark mit zunehmender Datensatzanzahl ansteigen und die Laufzeit des parametrisierten Operators etwas größer ist, als die des *Cross*-Operators. Die erhöhte Anzahl von Prüfungen der einzelnen Parameter fällt hier erkennbar auf. Bei den einzelnen Vergleichen wurden immer alle gleichen Attribute herangezogen. Die hohe Anzahl von Matches bei mehr als 100.000 Knoten kann durch den Schwellwert von $92,5\%$ begründen werden. Bei einem Schwellwert von $99,9999\%$ werden 10713 Matches gefunden und zwar genau die, die vorhanden sind. Die beiden Graphen sind so gewählt, dass nur die Schnittmenge identische Knoten besitzt, da auf der einen Seite Autoren und zugehörige Publikationen abgebildet sind und auf der anderen Seite Publikationen mit dem Verlag/Verleger. Demzufolge konnte der Verleger-Graph nur aus Publikationen

erstellt werden, die dieses Attribut gepflegt haben. Das sind im Autoren-Graph 10713 Publikationen.

Statt den *CoGroup*-Operator in Apache Flink zu verwenden, wurde bspw. auch der *Join*-Operator für einige Testläufe untersucht. Die Laufzeit lag dabei rund 25 % höher als beim Blocking anhand des Knotentyps mittels *CoGroup*-Operator. Abschließend lässt sich sagen, dass der Erhöhung der Laufzeiten in dem Fall mit mehr parallelen Servern und der damit verbundenen vergrößerten Taskanzahl entgegen getreten werden muss.

Knotenanzahl	ohne Blocking		Knotentyp		Knotentyp & 1. Buchstabe Name	
	Anz. Duplikate	Laufzeit	Anz. Duplikate	Laufzeit	Anz. Duplikate	Laufzeit
135	14	12s (13s)	14	11s	13	10s
1237	55	16s (16s)	55	15s	17	11s
11.683	830	4m21s (4m23s)	829	3m29s	63	24s
103.520	167786	4h14m (4h22m)	167752	3h18m	31210	13m53s

Tab. 6: Laufzeitverhalten bei Erhöhung der Datensatzanzahl (Standard-Hardware, Taskanzahl = 4)

5 Zusammenfassung und Ausblick

Im Rahmen dieses Beitrages wurde in Kapitel 3 ein Konzept vorgestellt, welches die Duplikaterkennung in Graphen innerhalb des GRADOOP-Frameworks ermöglicht. Der vorgestellte Workflow umfasst dabei Blocking, Lastbalancierung durch Untergruppierung sowie die Berechnung von Ähnlichkeiten zwischen Knoten. Das Konzept wurde prototypisch als ein GRADOOP-Operator implementiert und evaluiert. Die Ergebnisse der Evaluation haben gezeigt, dass der Höchstwert der Gesamtqualität bei den gefundenen Duplikaten bei ca. 80 % liegt und die Laufzeit mit Erhöhung der Task- und Gruppenanzahl skaliert. Aus Zeit- und Ressourcenmangel war eine Untersuchung auf einen Servercluster nicht möglich, jedoch ist zu vermuten, dass mit Erhöhung der Cores die Laufzeiten weiter skalieren.

Der Operator-basierte Workflow ermöglicht eine nahtlose Integration in andere GRADOOP-Workflows und damit in komplexe Datenanalyseszenarien (Ende-zu-Ende-Analysen). Weiterentwicklungen fokussieren auf die Übertragung von Lastbalancierungsalgorithmen, um die Effizienz bei der parallelen Ausführung von Duplikaterkennungs-Workflows in GRADOOP zu steigern.

Anmerkung: Die vorliegende Arbeit wurde durch Prof. Dr. Andreas Thor (Hochschule für Telekommunikation Leipzig, HfTL) und Dr. Eric Peukert (Universität Leipzig) im Rahmen des Competence Center for Scalable Data Services and Solutions (ScaDS) Dresden/Leipzig (BMBF 01IS14014B) betreut.

Literatur

[Cu13] Curtiss, Michael; Becker, Iain; Bosman, Tudor; Doroshenko, Sergey; Grijincu, Lucian; Jackson, Tom; Kunnatur, Sandhya; Lassen, Soren; Pronin, Philip; Sankar, Sriram; Shen,

Guanghao; Woss, Gintaras; Yang, Chao; Zhang, Ning: Unicorn: A System for Searching the Social Graph. PVLDB, 6(11):1150–1161, 2013.

[FS69] Fellegi, I. P.; Sunter, A. B.: A Theory for Record Linkage. Journal of the American Statistical Association, 64:1183–1210, 1969.

[Gh15] Ghrab, Amine; Romero, Oscar; Skhiri, Sabri; Vaisman, Alejandro A.; Zimányi, Esteban: A Framework for Building OLAP Cubes on Graphs. In: Advances in Databases and Information Systems - 19th East European Conference, ADBIS 2015, Poitiers, France, September 8-11, 2015, Proceedings. Jgg. 9282 in Lecture Notes in Computer Science. Springer, S. 92–105, 2015.

[GM13] Getoor, Lise; Machanavajjhala, Ashwin: Entity resolution for big data. In: The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013. ACM, S. 1527, 2013.

[Ju15] Junghanns, Martin; Petermann, André; Gómez, Kevin; Rahm, Erhard: GRADOOP: Scalable Graph Data Management and Analytics with Hadoop. CoRR, abs/1506.00548, 2015.

[Ko14] Kolb, Lars: Effiziente MapReduce-Parallelisierung von Entity Resolution-Workflows. Dissertation, University of Leipzig, 2014.

[KR10] Koepcke, Hanna; Rahm, Erhard: Frameworks for entity matching: A comparison. Data & Knowledge Engineering, 69(2):197–210, 2010.

[KTR10] Köpcke, Hanna; Thor, Andreas; Rahm, Erhard: Evaluation of entity resolution approaches on real-world match problems. PVLDB, 3(1):484–493, 2010.

[KTR12] Kolb, Lars; Thor, Andreas; Rahm, Erhard: Dedoop: Efficient Deduplication with Hadoop. PVLDB, 5(12):1878–1881, 2012.

[Ma10] Malewicz, Grzegorz; Austern, Matthew H.; Bik, Aart J.C; Dehnert, James C.; Horn, Ilan; Leiser, Naty; Czajkowski, Grzegorz: Pregel: A System for Large-scale Graph Processing. SIGMOD '10, S. 135–146, 2010.

[neo13] Graph Database Applications and Concepts with Neo4j, SAIS, 2013. http://aisel.aisnet.org/sais2013/24.

[Wi06] Winkler, William E: Overview of record linkage and current research directions. Bericht, BUREAU OF THE CENSUS, 2006.