Effiziente Integration von Data- und Graph-Mining-Algorithmen in relationale Datenbanksysteme

Manuel Then, Linnea Passing, Nina Hubig,
Stephan Günnemann, Alfons Kemper und Thomas Neumann

TU München, Lehrstuhl für Datenbanksysteme,
Boltzmannstraße 3, 85748 Garching bei München
{then,passing,hubig,guennemann,kemper,neumann}@in.tum.de

Zusammenfassung. Die Nutzung komplexer Algorithmen zur Analyse oft hochdimensionaler Datensätze gerät im Kontext von „Big Data“ immer mehr in das Zentrum der Aufmerksamkeit. Um diese komplexen Datenanalysen effizient zu ermöglichen liegt es nahe, sie in die am weitesten verbreiteten Datenspeicher zu integrieren – in relationale Datenbanksysteme. Dies führt zu interessanten Fragestellungen nicht nur im Bereich der technischen Integration sondern besonders auch in der Anfragespezifikation und -auswertung. In diesem Kurzbeitrag beschreiben wir, wie Algorithmen zur Datenanalyse effizient und nutzerfreundlich in das relationale Hauptspeicherdatenbanksystem HyPer integriert werden können. Wir evaluieren unseren Ansatz anhand eines Vergleichs mit zwei verbreiteten Datenanalysesystemen auf Graph- und Vektordaten.

Schlüsselwörter: Data Mining, Graph, SQL, HyPer, RDBMS

1 Motivation

Die gegenwärtige Datenexplosion stellt stand-alone Data-Mining-Programme vor Schwierigkeiten: zur Analyse großer Datenmengen sind sie durch ihre eingeschränkte Datenverwaltungsfunktionalität kaum geeignet. Besonders da die zu analysierenden Daten in die Applikationen kopiert werden müssen, sind diese für sich ändernde Daten ineffizient. Im Gegensatz hierzu bieten (relationale) Datenbanksysteme eine effiziente und update-freundliche Datenspeicherung. Laut Aggarwal et al. [1] ist die nahtlose Integration von Data-Mining-Technologien in Datenbanksysteme daher eine der momentan wichtigsten Herausforderungen. Einige Datenbanksysteme, etwa SAP HANA [3] und HyPer [4], integrieren bereits die verschiedenen Workloads OLAP und OLTP in ein einzelnes System, sodass die Datenbasis nur einmal vorgehalten werden muss und ETL-Zyklen entfallen. Durch das Paradigma „Data Mining in the database“ [6] entsteht so eine Datenbasis, die für sämtliche Anfragen genutzt werden kann.

 In: R. Bergmann, S. Görg, G. Müller (Eds.): Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB. Trier, Germany, 7.-9. October 2015, published at http://ceur-ws.org

1.1 Stand der Technik

SAP HANAs *Predictive Analytics Library* [3] und Oracle *Data Miner* [6] erlauben es Data-Mining-Algorithmen ähnlich zu SQL-Anfragen einzeln auszuführen. Die Ergebnisse der Algorithmen werden jeweils in zu spezifizierenden Tabellen abgelegt und können damit in separaten SQL-Anfragen genutzt werden. Eine interaktive Weiterverarbeitung der Ergebnisse in derselben Anfrage ist somit nicht möglich. Oracle Data Miner legt zudem den Fokus auf *supervised* Machine-Learning-Algorithmen. Es wird hier zunächst ein Modell mit Trainingsdaten angelegt, das anschließend mithilfe von SQL-Funktionen auf Testdaten angewandt. Für *unsupervised* Algorithmen erscheint dies umständlich, da ebenfalls ein persistentes Modell angelegt werden muss. Beide Produkte benennen als Vorteil, dass die Daten nicht mehr kopiert werden müssen, sondern innerhalb der Datenbank analysiert werden können. Wie in diesem Abschnitt gezeigt, ist die Integration in SQL-Anfragen bei beiden Lösungen jedoch nur oberflächlich gegeben.

Im Gegensatz hierzu streben wir eine tiefere Integration mit SQL an, sodass SQL- und Data-Mining-Anfragen nahtlos miteinander verwendet und kompiliert werden können.

1.2 Wissenschaftlicher Beitrag

In diesem Kurzbeitrag stellen wir am Beispiel von HyPer dar, wie Data-Mining-Algorithmen effizient in relationale Datenbanksystemen integriert werden können. Die wichtigsten Kontributionen unseres Beitrags sind zusammengefasst:

- Mehrschichtiges Spezifikationsmodell zur Erstellung und Nutzung der Algorithmen, bestehend aus Laien-, Domänenexperten- und Programmierer-Sicht
- SQL-Erweiterung zur Spezifikation von effizienten iterativen Algorithmen
- Laufzeitvergleiche mit Data-Mining-Anwendungen am Beispiel der Algorithmen *PageRank* (für Graphdaten) und *K-Means* (für Vektordaten)

2 Arten der Integration von Algorithmen in HyPer

Existierende Datenanalysesysteme nutzen häufig eigene, meist proprietäre, Sprachen oder APIs, um Analysen zu spezifizieren. Dies hat diverse Nachteile. Unübliche Anfragesprachen machen es nötig, die Nutzer – häufig Datenanalysten aus der Anwendungsdomäne – aufwändig zu schulen. Werden Hochsprachen-APIs – z.B. in Java – verwendet, gibt es zwar viele erfahrene Programmierer, jedoch haben diese selten das nötige Domänenwissen. Bei in Hochsprachen spezifizierten Anfragen ist es zudem für das Datenanalysesystem sehr schwierig, die Anfrage zu optimieren, um eine effiziente Ausführung zu ermöglichen.

Wir wählen daher einen neuartigen, mehrstufigen Ansatz für die Integration von Data Mining in HyPer. Unser Ziel ist es, Domänenspezialisten auf einfache Art und Weise effiziente Anfragen spezifizieren zu lassen, während Spezialisten alle Freiheitsgrade behalten. Die vier im folgenden vorgestellten Stufen der Integration unterscheiden sich daher sowohl in der Mächtigkeit der Spezifikation, als auch in den Möglichkeiten des DBMS, die Anfragen zu optimieren.

2.1 Externer Zugriff auf die Datenbank

Um allgemeine Data-Mining-Funktionalität anzubinden, bei der das DBMS nur als Datenspeicher verwendet wird, bietet HyPer PostgreSQL-kompatible Datenbankschnittstellen, u.a. JDBC. Zwar ermöglichen diese beliebige Berechnungen, jedoch verhindert der Zugriff über sie umfassende Anfrageoptimierungen und führt potentiell zu teurem Datenaustausch zwischen den beteiligten Systemen.

2.2 Programmausführung in der Datenbank

Als tiefergehende Integration von Data Mining erlaubt HyPer die Ausführung von Nutzercode als *User-defined Functions (UDFs)*. Wie bei anderen Datenbanksystemen können berechtigte Nutzer dabei beliebige Funktionalität hinzufügen. Diese wird dann entweder direkt innerhalb des Datenbanksystems (*unfenced*) oder in einer Sandbox (*fenced*) ausgeführt. Dadurch ist es nicht mehr nötig, Daten in externe Systeme zu kopieren.

2.3 SQL-Spracherweiterungen

Oft lassen sich Data-Mining-Algorithmen nur umständlich in SQL ausdrücken. Dies liegt unter anderem daran, dass viele Verfahren iterativ sind. Um diese in SQL abzubilden kommen häufig rekursive *Common Table Expressions* (`WITH`-Statements) zum Einsatz, die eine monoton wachsende Relation berechnen. Da iterative Algorithmen im Normalfall jedoch nur auf die Daten der vorherigen Iteration zugreifen, um die aktuelle Iteration zu berechnen, wird bei diesem Vorgehen viel Speicher unnütz belegt. Dies ist vor allem für Hauptspeicherdatenbanksysteme ein Problem, da Speicher hier eine besonders wertvolle Ressource ist. Als Lösung für dieses Problem schlagen wir ein Iterationskonzept für SQL vor. Syntaktisch ist dieses an `WITH` angelehnt:

```
with recursive [Algo] as ([Initialization] iterate [Step]
                          until [Condition])
                         select * from [Algo]
```

Es wird hier eine temporäre Relation *Algo* erstellt, die anfangs das Resultat der Unteranfrage *Initialization* enthält und auf die iterativ *Step* angewendet wird, bis der boolsche Ausdruck *Condition* wahr ist. Diese Spracherweiterung erlaubt es uns, iterative Data-Mining-Verfahren auf einfache Weise direkt in SQL auszudrücken. Dies ermöglicht nicht nur die direkte Verwendung des ausgereiften state-of-the-art relationalen Anfrageoptimierers von HyPer, sondern auch die Nutzung der hochoptimierten parallelen Codegenerierungs- und Ausführungsengine [4].

2.4 Data Mining im Datenbankkern

Im Gegensatz zu anderen Datenbanksystemen integriert HyPer wichtige Data-Mining-Funktionalität direkt im Datenbankkern. Für den Nutzer sind diese

syntaktisch nicht von den zuvor beschriebenen UDFs zu unterscheiden. So berechnet folgende Anfrage für jeden Knoten des durch die Kanten in *edges* gebildeten Graphen die parametrisierte PageRank-Metrik:[1]

```
select * from pagerank((select src,dest from edges), 0.85, 0.001)
```

Intern wird die Berechnung jedoch von spezialisierten Operatoren ausgeführt, in diesem Fall durch einen Sort- gefolgt von einem PageRank-Operator.

HyPer wählt dabei u.a. eine effiziente interne Graphrepräsentation und führt weitere Vorverarbeitungsschritte durch, um die Metrik zu berechnen. Des Weiteren kennt der Anfrageoptimierer die genauen Eigenschaften des PageRank-Operators und kann somit den optimalen Ausführungsplan wählen.

Lambda-Ausdrücke Vordefinierte Funktionen allein decken jedoch nicht alle Einsatzzwecke ab. HyPer erlaubt daher die Verwendung von Lambda-Ausdrücken in SQL-Anfragen. Dies ermöglicht etwa im K-Means-Algorithmus den Einsatz benutzerdefinierter Distanzfunktionen, wobei die volle Optimierbarkeit der Anfrage erhalten bleibt. Bei entsprechend gewählter Distanzfunktion können dabei numerische und kategorische Daten kombiniert analysiert werden, was unverzichtbar für Datenbanksysteme mit ihren verschiedenen Datentypen ist.

3 Experimentelle Evaluierung

In diesem Abschnitt evaluieren wir unsere Ansätze aus den Abschnitten 2.3, nachfolgend *HyPer SQL*, und 2.4, im Folgenden *HyPer Op*. Wir implementieren dazu jeweils einen Graph- und Vektoralgorithmus. *PageRank* wählen wir als bekannten Vertreter der Graphverfahren und als Basis weiterer iterativer Algorithmen. Für Vektordaten verwenden wir *K-Means*, ein häufig genutztes Clusteringverfahren [7].

Als Vergleichssysteme verwenden wir Apache Spark 1.4.0 [8] und MATLAB R2015 [5] mit litekmeans [2]. Alle Tests wurden auf einem Intel Core i7-5820K (6x3,3 GHz) mit 32 GB Hauptspeicher unter Ubuntu Linux 15.04, Kernel 3.19 durchgeführt. Die Datensätze, Tabelle 1, passen auch mit zusätzlichen programmspezifischen Datenstrukturen noch in den Hauptspeicher. Die LDBC-Graphdatensets wurden mit dem gleichnamigen Datengenerator[2] erstellt.

In unseren Tests, Tabelle 2, zeigt MATLAB die längsten Laufzeiten und das schlechtere Skalierungsverhalten[3], weshalb wir uns in der weiteren Auswertung auf den Vergleich mit Spark fokussieren. Im Bezug auf die Vektordatensätze zeigen HyPer und Spark ein ähnliches Skalierungsverhalten, wobei HyPer im Datenbankkern jedoch um Faktor 1–2 schneller ist. Für die PageRank-Graphanalyse zeigt sich, dass HyPer sowohl besser skaliert als Spark, als auch 1–2 Größenordnungen schneller ist. HyPer mit in SQL spezifizierten Data-Mining-Anfragen zeigt Potential, erzeugt aber im Fall von K-Means noch keine optimalen Ausführungspläne.

[1] Die explizite Klammerung der Unteranfrage ist nötig, da wir dort beliebige Anfragen erlauben; einfache Kommatrennung führt zu Mehrdeutigkeiten in der Grammatik.

[2] Siehe `https://github.com/ldbc/ldbc_snb_datagen`.

[3] MATLAB führt die Algorithmen sequentiell aus. Jedoch wäre die Laufzeit auch bei perfekter Skalierung über die sechs verfügbaren CPU-Kerne noch immer unterlegen.

Tabelle 1. Datensätze zur Evaluierung der gewählten Verfahren

Datensatz	**Datenmodell**	**# Tupel**	**# Dimensionen**	**Größe in GB**
Syn 1	Vektordaten	50 000	50	0,01
Syn 2	Vektordaten	15 000 000	4	0,22
Syn 3	Vektordaten	15 000 000	50	2,79
		# Knoten	**# Kanten**	
LDBC SF 1	Graph/Kantenliste	10 993	451 522	0,03
LDBC SF 10	Graph/Kantenliste	72 949	4 641 430	0,26

Tabelle 2. Laufzeiten in Sekunden, OOM = Out of Memory

	Datensatz	**Spark**	**MATLAB**	**HyPer Op**	**HyPer SQL**
K-Means,	Syn 1	0,287	1,504	0,110	1,040
$k = 3$,	Syn 2	5,347	582,139	4,643	87,588
3 Iterationen	Syn 3	51,632	OOM	19,496	369,438
PageRank,	LDBC SF 1	2,329	4,506	0,16	0,30
$d = 0.85$,	LDBC SF 10	72,391	OOM	1,69	4,76
$e = 0.0001$					

Zusammenfassung Wir haben eine vierschichtige Integration von Data Mining in unser relationales Hauptspeicherdatenbanksystem HyPer vorgestellt. Wichtige Algorithmen sind hochoptimiert im Datenbankkern integriert und können direkt per SQL aufgerufen werden, wo sie auch Laien leicht zugänglich sind. Zusätzliche Algorithmen können durch unsere Spracherweiterung direkt in SQL spezifiziert werden und nutzen somit HyPers effiziente Codegenerierung und Laufzeitumgebung. Unsere Testergebnisse zeigen, dass Data Mining auf Vektor- und Graphdaten in HyPer performanter ist und besser skaliert als in vergleichbaren state-of-the-art Datenanalysesystemen. Zeitaufwändige ETL-Zyklen entfallen.

Literatur

1. N. Aggarwal, A. Kumar, H. Khatter, and V. Aggarwal. Analysis the effect of data mining techniques on database. *Advances in Engineering Software*, 47(1), 2012.
2. D. Cai. Litekmeans: the fastest matlab implementation of kmeans. *Available at: http://www.zjucadcg.cn/dengcai/Data/Clustering.html*, 2011.
3. F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA database – an architecture overview. *IEEE Data Eng. Bull.*, 35, 2012.
4. A. Kemper and T. Neumann. HyPer: A hybrid OLTP & OLAP main memory database system based on virtual memory snapshots. In *ICDE*, April 2011.
5. MATLAB. *Version 8.5 (R2015a)*. The MathWorks Inc., 2015.
6. P. Tamayo et al. Oracle data mining. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 1315–1329. Springer US, 2005.
7. Xindongi Wu et al. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.
8. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *HotCloud'10*. USENIX Association, 2010.