Transparente Datenbankunterstützung für Analysen auf Big Data

Dennis Marten
Universität Rostock, Institut für Informatik
18051 Rostock
dm@informatik.uni-rostock.de

Andreas Heuer
Universität Rostock, Institut für Informatik
18051 Rostock
ah@informatik.uni-rostock.de

Zusammenfassung

Es ist kein Geheimnis, dass in den letzten Jahren in der Entwicklung und Forschung unterschiedlichster Bereiche ein enormer Anstieg an genutzten Datenmengen zu beobachten ist. Grund hierfür sind hauptsächlich neue technische Möglichkeiten, wie beispielsweise bessere oder günstigere Sensortechnik. Diese unzähligen Datenmengen bieten den Entwicklern und Forschern oftmals unfassbare Möglichkeiten, führen aber auch teilweise zu neuen Problemen. So werden oftmals Programme für Auswertungen genutzt, die nicht für enorm große Datensätze entwickelt wurden und demnach entweder deutlich zu langsam sind oder nicht funktionsfähig. Ein gutes Beispiel hierfür ist das open-source Programm R [14], welches für sein umfangreiches Repertoire statistischer Methoden, wie beispielsweise Machine-Learning-Algorithmen, bekannt und beliebt ist. R's Performance jedoch hängt maßgeblich mit der Datengröße zusammen. Steigt diese über die Größe des Hauptspeichers, fällt die Rechengeschwindigkeit drastisch ab. Aus diesem Grund stellen wir in diesem Artikel unseren Framework für eine transparente Datenbank- und Parallelisierungsunterstützung von R vor. Hierbei legen wir großen Wert auf die Nutzerfreundlichkeit, d.h. in diesem Fall, dass der Nutzer seine gewohnte Entwicklungsumgebung (etwa sein R-Skript) nicht zu ändern braucht und trotzdem die Vorteile paralleler Berechnung, sowie die In- und Output-Fähigkeiten von Datenbanksystemen ausnutzen kann.

ACM Klassifikation

H.2.4 [**Database management**]: Systems; H.2.8 [**Database management**]: Database applications

Schlagworte

Performance

Stichworte

Big Data, R, Datenbanksysteme, Parallel Computing, Machine-Learning

27^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 26.05.2015 - 29.05.2015, Magdeburg, Germany.

1. EINLEITUNG

Durch das Voranschreiten der Technik ist in den letzten Jahren ein enormer Anstieg genutzter Datenmengen in Entwicklung und Forschung zu beobachten. Dies eröffnete einige Möglichkeiten und Herausforderungen, die teilweise durch konventionelle Ansätze nicht mehr realisierbar sind. Dies hat in den letzten Jahren zu einem Hype in der Entwicklung neuer Umgebungen oder Programmen geführt, die gezielt auf große Datenverarbeitung ausgelegt sind. Ein gutes Beispiel hierfür ist das MapReduce Framework [7] mit seinem wohl populärsten Stellvertreter Hadoop [2]. Der Einschlag dieser Umgebung war enorm und führte zu zahlreichen weiteren Projekten, die auf ihr aufbauten. Jedoch muss auch hier anerkannt werden, dass nicht jedes „Big Data Problem" mit MapReduce gelöst werden kann, bzw. nicht jeder Entwickler die Expertise hat es in MapReduce zu lösen. Letzteres ist ein Punkt den wir in diesem Artikel betonen wollen. Die gewohnten Entwicklungsumgebungen, das sind beispielsweise Programme für statistische Auswertungen, sind in den wenigsten Fällen solche, die für große Datenverarbeitung ausgelegt sind. Wir beziehen uns in diesem Artikel auf die open-source Programmiersprache R, welches für ihr umfangreiches Repertoire statistischer Methoden bekannt ist und eine äußerst große und stetig wachsende Community besitzt. Das Problem, dem viele R-Entwickler begegnen ist das Einbrechen der Performance bei der Behandlung von Daten, welche die Hauptspeichergröße überschreiten. Es gibt natürlich Wege diesem Problem entgegenzutreten, wie beispielsweise die Hardware zu verbessern, auf Performance-verbessernde R-Erweiterungen zurückzugreifen oder ganz auf ein neues Programm oder Tool umzusteigen. Alle diese Lösungen sind, falls sie ihr eigentliches Ziel überhaupt erreichen, mit einigen Nachteilen, wie materiellen Kosten oder erhöhtem Zeitaufwand durch Einarbeitung in neue Tools und Sprachen, verbunden. Dies stellt den Einstiegspunkt des hier vorgestellten Projektes dar: Ein Framework das R mit Datenbank- und Parallelisierungstechnologie ausstattet. Hierbei wird der R-Code analysiert und in SQL- und parallelisierbare Segmente zerlegt. Einer der Hauptaspekte liegt hierbei auf der transparenten Umsetzung, das heißt, dass der Nutzer nur sein R-Skript benötigt und keine Kenntnisse von SQL oder von Parallelisierungtechniken haben muss.

Der Rest des Artikels ist folgendermaßen gegliedert: In Abschnitt 2 werden Forschungsprojekte und Entwicklungen vorgestellt die entweder R und Datenbanksysteme oder R und paralleles Rechnen vereinigen. In Abschnitt 3 stellen wir einige Aspekte unseres eigenen Projektes vor, wobei die Architektur und Schlüsselcharakteristiken und die Wahl eines

geeigneten Datenbanksystems diskutiert werden. Im letzten Abschnitt beschreiben wir künftige Teilprojekte, die wir in Angriff nehmen werden.

2. STAND DER FORSCHUNG

Das Verbinden von R mit Datenbank- oder Parallelisierungstechnologie ist sicherlich keine neue Idee. Es gibt zahlreiche verschiedene Projekte, die genannt werden sollten.

Die R-Community selbst ist sehr lebendig und hat in den letzten Jahren verschiedene Erweiterungen („Pakete“) hervorgebracht. Datenbankverbindungen mittels allgemeinem JDBC-Treiber oder für spezielle Datenbanksysteme wie MySQL, PostgreSQL und ähnliche sind alle verfügbar und einfach handhabbar. Allerdings eröffnen diese Anbindungen oftmals nur eine Schnittstelle um SQL-Anfragen in R durchzuführen. Eine deswegen sehr interessante Erweiterung ist das dplyr-Paket [6], welches eine automatische Übersetzung einiger, simpler R-Methoden in SQL ermöglicht.

MonetDB [8] nutzt diese, um ein automatisches Prefiltering mittels SQL zu ermöglichen, welches die Datenmenge die R verarbeiten muss deutlich reduzieren kann. Zusätzlich bietet MonetDB eine Möglichkeit R-Methoden in seine SQL-Anfragen einzubetten. In dieser Verbindung zwischen dem Datenbanksystem und R wird zusätzlich ein geteilter Hauptspeicheransatz verfolgt, welcher beiden Schnittstellen eine schnelle und günstige Kommunikation erlaubt.

Ein weiteres Datenbankprojekt das hervorzuheben ist, ist RIOT-DB [17], welches MySQL und R verbindet. Hier werden Variablen zwischen R und MySQL mittels *Mapping* verbunden, welche mittels überladener Operatoren in R bearbeitet werden. Das Projekt konzentriert sich dabei hauptsächlich auf die Optimierung von Matrix-Operationen.

Einen anderen Ansatz schlägt paradigm4's SciDB [13] vor. SciDB ist ein Datenbanksystem, welches im Gegensatz zu typischen spalten- oder zeilenorientierten Datenbanksystemen auf einem Array-Datenmodell basiert. Dieses Modell soll sich bei wissenschaftlichen Berechnungen äußerst günstig auf die Rechenzeit auswirken. SciDB unterstützt, ähnlich wie RIOT-DB, eine durch *Mapping* realisierte Verbindung der Schnittstellen und eine, durch überladene Operatoren in R realisierte, Verarbeitung.

Als Zwischenfazit lässt sich sagen, dass mehrere Projekte für Datenbankenverbindungen mit R bereits existieren, welche eine gute Basis für weitere Forschung bieten. Ein Punkt der hierbei meist vernachlässigt wird ist die Übersetzung von komplexeren R-Funktionen, welche ein Schlüsselaspekt unseres Framework sein wird. Zudem werden für einige dieser Projekte SQL-Kenntnisse des Nutzers vorausgesetzt oder zumindest das manuelle Aufbauen von Verbindungen und Nutzen neuer vorgefertigter Methoden.

Wie auf der Datenbankseite, existieren auch für paralleles Rechnen zahlreiche R-Pakete die den Nutzern etwa Multi-Core-Verarbeitung (siehe etwa das pR-Paket [9]) oder sogar eine Schnittstelle zu Hadoop [2] ermöglicht.

Als eine Alternative zu R sollte IBM's SystemML [1, 10] genannt werden. Dieses arbeitet mit einer R-ähnlichen Sprache und führt MapReduce-Strategien für zahlreiche Operationen ein. Besonders an SystemML ist der Fokus auf die numerische Stabilität von Algorithmen, welche gerade bei einer großer Anzahl von algebraischen Operationen oftmals leidet und so Rundungsfehlereinflüsse deutlich verstärken kann.

Weitere nennenswerte Verbindungen von R und MapReduce sind RHIPE [5], Ricardo [15] und Cumulon [3]. Schließlich sei der kommerzielle Vertreter Revolution Analytics' „Revolution R“ mit seinem ScaleR-Paket genannt. Dieses bietet eine breite Reichweite an automatischen Übersetzungen von R-Funktionen für parallele Plattformen, wie beispielsweise Hadoop-Cluster.

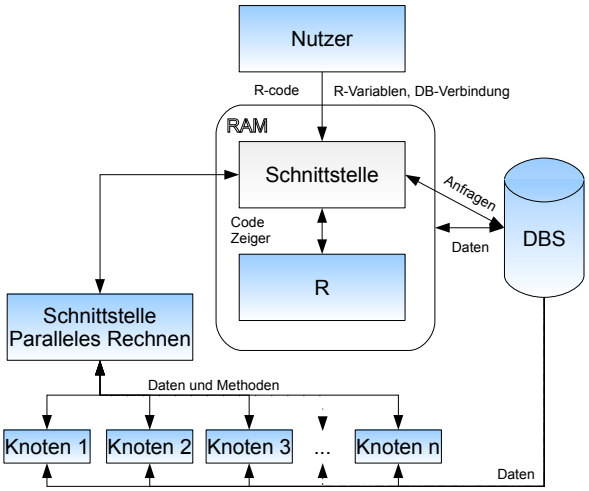

Abbildung 1: Architektur des vorgestellten Frameworks.

Unser Projekt wird sich von diesen Entwicklungen zum einen durch das Nutzen von Datenbank- *und* Parallel Computing-Technologie und zum anderen durch den Fokus des *automatischen Übersetzens* von bereits existierenden, komplexen R-Methoden unterscheiden. Damit soll erreicht werden, dass Nutzer ohne Änderungen ihrer R-Skripte parallele Verarbeitung nutzen können. Die einzig nötige Anpassung sollte eine Zuordnung von Datenbank- und R-Variablen sein.

3. ARCHITEKTUR

In diesem Abschnitt stellen wir unsere Architektur vor, die in Abbildung 1 dargestellt ist. Das Augenmerk dieser, ist die eigens implementierte Schnittstelle, die verschiedene Funktionen inne hat. Sie ist mit R, der Parallelisierungsschnittstelle und dem Datenbanksystem verbunden und ist demnach zur Verwaltung der Kommunikation, welche in den nächsten Unterabschnitten näher beschrieben wird, verantwortlich. Es sollte außerdem hervorgehoben werden, dass zudem keine der anderen vier Hauptschnittstellen mit einer anderen verbunden ist. Weiterhin wird hier ein Ansatz des geteilten Hauptspeichers zwischen R und der Schnittstelle verfolgt, welcher eine günstige Kommunikation ermöglicht um beispielsweise Zwischenergebnisse zu bearbeiten. Ursprünglich war ein klassischer MapReduce-Ansatz vorgesehen, wobei jedoch das Senden der Daten zur Schnittstelle und der darauffolgenden erneuten Aufteilung zu unnötigen Kosten geführt hätte. Daher werden die Knoten zur parallelen Berechnung selbst mit der Datenbank verbunden um unnötiges Senden von Daten zu meiden und so optimal die Vorteile aller drei Welten nutzen zu können.

Im Folgenden werden Einblicke in den Ablaufprozess gegeben und Vor- und Nachteile des Ansatzes diskutiert.

3.1 Konzeptuelle Charakteristiken

In diesem Abschnitt werden verschiedene zu implementierende Charakteristiken der Zielumgebung erläutert.

3.1.1 Input

Eines unser Hauptbedenken gilt der Nutzerfreundlichkeit, da ein Einarbeiten in neue Programme oder Techniken für Nutzer sehr zeitintensiv und frustrierend sein kann. Demnach sollte nur das gewohnte R-Skript übergeben werden und zusätzlich eine Tabelle, die die Beziehung der Variablen im R-Skript und der Datenbank beschreibt.

3.1.2 Wahl der Plattform

Es ist sicherlich nicht sinnvoll jede Methode direkt auf einem Cluster ausführen zu lassen. Viele verschiedene Faktoren spielen bei der Wahl, ob eine Funktion auf einem Cluster, lokal parallel oder vielleicht nur auf der Datenbank selbst ausgeführt werden sollte, eine Rolle. Schlüsselfaktoren sind etwa die Clustergröße mit Hauptspeicher und Prozessordaten, die Netzwerkgeschwindigkeit, die zu verarbeitende Datengröße, der genutzte Algorithmus und welche Teilprobleme überhaupt parallelisierbar sind. Ein hilfreicher Ansatz ist das Aufstellen von Kostenfunktionen, die diese Aspekte mit einbeziehen und bei der Wahl Hilfe leisten können. Da bei den meisten der geplanten Methoden keine iterativen Probleme mit Abbruchbedingung enthalten sind, ist die Herleitung in vielen Fällen unkompliziert.

Der R-Code wird anfangs durchlaufen und ein Ablaufplan erstellt, welcher einzelne Methoden einzelnen Plattformen zuordnet. Bei der Erstellung des Plans werden komplexere (unterstützte) R-Methoden in Kombinationen einfacher, algebraischer Operationen und Aggregationsfunktionen aufgespaltet. Für diese grundlegenden Operationen können dann einfache Parallelisierungsstrategien eingesetzt werden, wie beispielsweise bei algebraischen oder distributiven Aggregationsfunktionen. Ist ein Ablaufplan erstellt, liegen in günstigen Fällen keine Datenabhängigkeiten vor, sodass die Datenbank, der Cluster und der Hauptknoten (etwa R) gleichzeitig arbeiten können und so der Leerlauf einzelner Instanzen minimiert wird.

3.1.3 Geschachtelte Anfragen und lazy-SQL-Auswertung

Ein weiterer, wichtiger Ansatz den wir verfolgen ist das *lazy* Auswerten von SQL-Anweisungen, was unweigerlich zur Verarbeitung geschachtelter SQL-Anfragen führt. Diese Methodik hat zur Folge, dass der Hauptspeicher von eventuell unnötigen Daten verschont bleibt. Eines der erstaunlich häufig vorkommenden Beispiele aus der Praxis ist das Einlesen von CSV-Dateien (Comma-Separated-Values), welche keine vorgefilterten Read ermöglichen. Im konventionellen R-Code würde zwangsläufig die gesamte Datei eingelesen, auch Daten die vielleicht niemals benötigt würden.

Ein anderes Beispiel sind Sequenzen von Berechnungen in denen nach und nach Daten gefiltert werden. Können hier Anfragen an die Datenbank verzögert und miteinander verbunden werden, entstehen geschachtelte SQL-Anfragen, deren Ergebnismenge mitunter deutlich geringer sein kann. Dadurch wird unnötiges Senden von Daten und deren Aufnahme in den Hauptspeicher vermindert. Ferner ist davon auszugehen, dass die Datenbankoptimierer mit solchen, typischerweise einfachen, Schachtelungen problemlos umgehen können und diese intern entschachteln.

3.2 Architekturdiskussion

In diesem Abschnitt wird das Potenzial der Architektur näher erläutert. Dazu werden einige Schlüsselfragen gezielt beantwortet.

3.2.1 Warum nicht ein paralleles Datenbanksystem benutzen?

Parallele Datenbanksysteme werden seit Jahrzehnten entwickelt und gepflegt. Sie enthalten demnach viele hochentwickelte Techniken für paralleles Rechnen. In [12] wird gezeigt, dass solche Systeme es vermögen auch MapReduce-Programme teilweise deutlich in den Schatten zu stellen. Es wird aber auch darauf eingegangen, dass solche Systeme typischerweise sehr teuer und äußerst schwierig zu installieren, beziehungsweise optimal zu konfigurieren sind.

Einer der Hauptkritikpunkte an parallelen Datenbanksysteme ist die starke Abhängigkeit der Performance bezüglich der Aufteilung der Daten. Das Problem hierbei ist, dass verschiedene Algorithmen von verschiedenen Partitionierungen profitieren. Jedoch ist es im Vorfeld oftmals nicht absehbar welche Methoden auf den Daten angewendet werden oder es ist bekannt, aber es werden viele verschiedene Methoden genutzt. Dies macht eine günstige Verteilung in vielen Fällen nicht möglich.

Ein weiterer Aspekt paralleler Datenbanksystemen ist, dass diese typischerweise Inter- anstatt Intraoperatorparallelrechnung unterstützen, welche momentan nicht in unserem Fokus liegt.

Außerdem gilt es ebenfalls zu bedenken, dass parallele Datenbanksysteme nicht für komplexere, nicht (auf einfache Weise) SQL-darstellbare Methoden geeignet sind.

3.2.2 Warum nicht MapReduce mit R benutzen?

Datenbanksysteme zu integrieren hat wesentliche Vorteile, die solch eine Umgebung bereichern können. Hier seien einige genannt:

- Die Verwaltungsstärke eines DBMS; dies beinhaltet die Reduktion von Redundanz, paralleles Arbeiten auf aktualisierten Daten, die In- und Output-Fähigkeiten des DBMS, ...
- Die Verbesserung des Dateizugriffes der Clusterknoten
- Vorfilterung von Dateien durch Selektion oder Berechnung von Methoden auf der Datenbank ...

Einer der Gründe warum Datenbanken MapReduce-Umgebungen drastisch übertreffen *können* ist die Analyse der Daten beim Einleseprozess. Eine strukturierte Datenanordnung, inklusive Indexstrukturen für schnellere Berechnungen, sind zwei der Hauptvorteile. Es gibt natürlich viele Wege MapReduce bezüglich dieser Aspekte zu optimieren [7], was jedoch sehr viel Zeit in Anspruch nimmt und für ein transparentes System nicht unbedingt vorteilhaft ist.

Ob parallele Datenbanksysteme oder MapReduce genutzt wird; viele Probleme lassen sich durch spezifische Systeme und spezifische Konfigurationen sehr schnell lösen. Aber die entscheidende Frage ist, inwiefern es sich lohnt für jedes solcher Probleme die „optimale“ Konfiguration zu suchen,

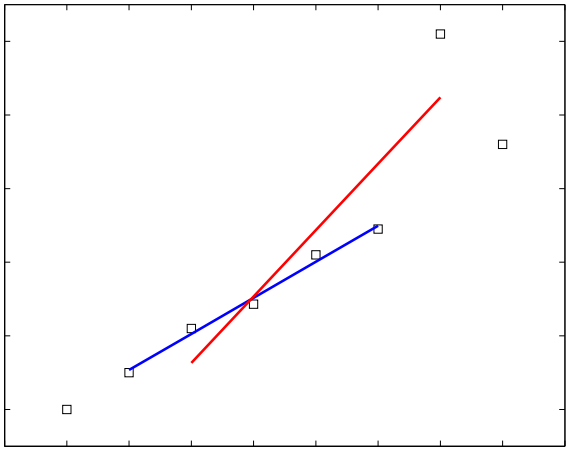

Abbildung 2: Beispielhafte Darstellung der Anstiegsänderung der linearen Regressionsgeraden bei Ausreissern.

anstatt möglicherweise lieber eine etwas schlechtere Performance in Kauf zu nehmen, dafür jedoch kein Expertenwissen und viel Zeit investieren zu müssen.

Als Fazit sollte gesagt werden, dass es nicht immer sinnvoll ist das Rad neu erfinden zu wollen, sondern durchaus auch das Nutzen jahrzehntelanger Forschung und Expertise von Datenbanksystemen sinnvoll ist.

3.3 Wahl des Datenbanksystems

In diesem Abschnitt wird die Suche und Auswahl eines geeigneten Datenbanksystems näher erläutert. Prinzipiell ist jedes gängige Datenbanksystem, welches SQL unterstützt, eine mögliche Wahl für solch eine Aufgabe. Jedoch sind sicherlich gravierende Performance-Unterschiede zwischen einzelnen Systemen zu erwarten.

Zur Entscheidungshilfe wurden einige Experimente in unserer Forschungsgruppe durchgeführt. Zunächst galt es jedoch eine adäquate Methode zu finden, die (zumindest teilweise) auf SQL darstellbar ist, eine gewisse Komplexität aufweist und außerdem eine signifikante Bedeutung für wissenschaftliches Rechnen besitzt. Im Rahmen des Graduiertenkollegs *MuSAMA* (Multimodal Smart Appliance Ensembles for Mobile Applications) [`www.musama.de`] fand diesbezüglich ein informativer Austausch statt. Der Forschungsschwerpunkt des Kollegs liegt in der Untersuchung und Entwicklung von Modellen und Methoden für Assistenzsysteme. Hierfür wurden beispielsweise Experimente durchgeführt, die Bewegungsdaten mittels xsens-motion-tracking für typische Bewegungsmuster in einer Küche aufgenommen haben. Mit diesen Daten sollen durch gezieltes Anwenden von Machine-Learning-Algorithmen, Rückschlüsse auf momentane Handlungen des Testsubjektes gemacht werden. Eine der Kern-Methoden in solchen statistischen Analysen ist die *lineare Regression*. Diese tritt bei der Erstellung linearer Modelle in statistischen Rechnungen häufig auf, besitzt aber auch andere wichtige Einsatzfelder. Ein weiteres Einsatzgebiet ist beispielsweise die Bestimmung von Ausreissern in Datensätzen. Hierbei werden Regressionsgeraden über kleinen Fenstern berechnet. Geht das Verschieben des Fensters um einen Datenpunkt mit einer signifikanten Änderung des Anstiegs der Regressionsgerade einher, wurde im Falle von glatten Daten wahrscheinlich ein Ausreisser gefunden (siehe Abbildung 2).

Sei nun zur formalen Beschreibung der linearen Regression $x, y \in \mathbb{R}^n$ Sensordaten, welche voneinander in etwa linear abhängig sind, das heißt

$$y \approx \alpha x + \beta$$

mit $\alpha, \beta \in \mathbb{R}$. Die gesuchten Koeffizienten werden dann durch die Minimierung der Quadrate der Residuen berechnet:

$$\left(\sum_{i=1}^{n} (y_i - (\alpha x_i + \beta))^2 \right) \to \min_{\alpha,\beta \in \mathbb{R}}$$

Die Koeffizienten sind dann durch

$$\alpha = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$\beta = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

beschrieben und können in SQL durch

```
SELECT ((COUNT(x) * SUM(x*y)) - (SUM(x)*SUM(y)))/
   (COUNT(x) * SUM(x*x)-SUM(x) * SUM(x)) AS alpha,
   AVG(y) - (COUNT(x) * SUM(x*y) - SUM(x)*SUM(y))/
   (COUNT(x) * SUM(x*x)-SUM(x)*SUM(x)) * AVG(x) as
   beta
FROM data
```

oder mit analytischen Funktionen durch

```
SELECT COVAR_POP(x,y)/COVAR_POP(x,x) as beta,
   AVG(y) - COVAR_POP(x,y)/COVAR_POP(x,x) *
   AVG(x) as alpha
FROM data
```

berechnet werden.

Die iterative Berechnung von Regressionsgeraden in kleinen Fenstern ist deutlich schwieriger, kann aber in SQL:1999 mittels `WITH RECURSIVE` oder `OVER` in SQL:2003 berechnet werden. Der im Experiment genutzte Code kann im Anhang gefunden werden. Dessen Komplexität ist ein gutes Beispiel dafür, weshalb ein automatisches *Parsen* für Entwickler deutlich attraktiver ist, als nur die Möglichkeit SQL in R zu nutzen.

Die vorgestellte Methode wurde nach dem Aufstellen des Konzepts in SQL auf den Plattformen PostgreSQL 9.4.1, DB2 v10.1.0.0 (BLU) und MonetDB v1.7 implementiert und deren Performance untersucht. Hierbei wurde für jedes System ein virtueller Server mit jeweils 2.8 GHz Octacore CPU und 4 GB DDR2 Arbeitsspeicher und dem Betriebssystem CentOS 6.6 genutzt. Gearbeitet wurde auf einer Tabelle mit 4000 Tupeln und 666 Spalten, sowie einer festen Fenstergröße von 5.

Aus Gründen der Platzbeschränkung kann hier keine detaillierte Auswertung vorgenommen werden. Eines der Kernergebnisse ist jedoch in Abbildung 3 dargestellt. Hierbei wurden die bereits erwähnten Systeme und zusätzlich MonetDB's Embedded R, welches das Aufrufen von R in SQL ermöglicht, verglichen. Da MonetDB momentan keine Rekursion unterstützt, wurden in diesem Experiment einzelne Tupel per Hand gruppiert. Alle Ergebnisse liegen jedoch trotzdem weit unter R's Performance, selbst wenn das Einlesen der Daten in R nicht mit eingerechnet wird. Unterschieden wurde hier zudem die Berechnung mittels einfacher

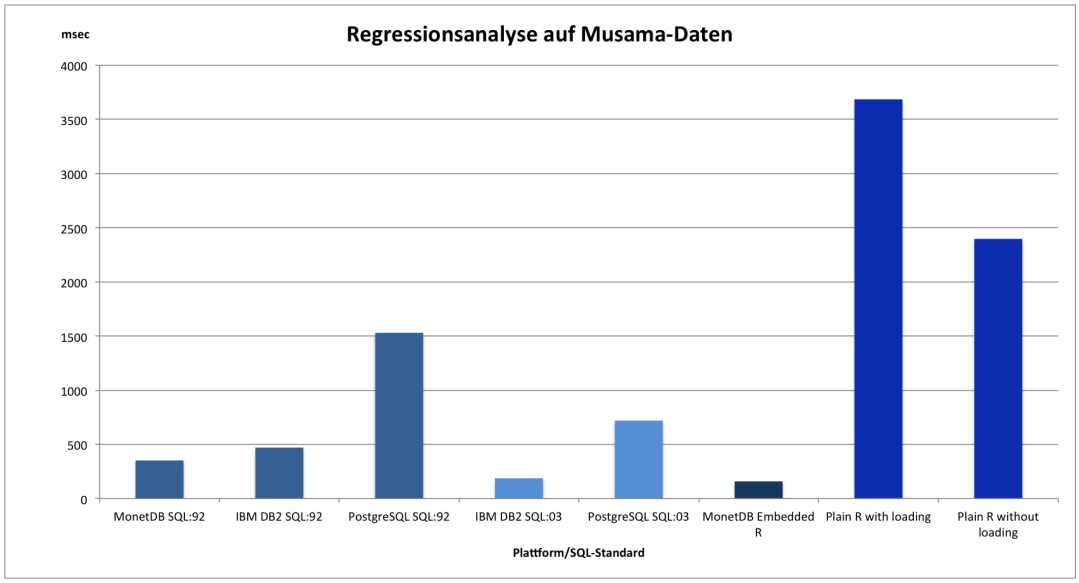

Abbildung 3: Zeitliche Auswertung der Regressionsanalyse ohne Rekursion. Verglichen wurden verschiedene Datenbanksysteme und verschiedene SQL-Standards.

algebraischer Operatoren unter SQL-92 und der Berechnung mittels der Funktionen `regr_slope` und `regr_intercept` in SQL:2003. Letztere führen, wie zu erwarten war, zu einer zusätzlichen Performanceverbesserung.

Unter der zusätzlichen Betrachtung unser anderen Experimente kann gesagt werden, dass MonetDB die Kontrahenten PostgreSQL und DB2 (BLU) deutlich schlug. Da PostgreSQL eine zeilenorientierte Architektur besitzt, war deren vergleichsweise schlechte Performance im Rahmen unserer Erwartung.

Die schwache Leistung von IBM's DB2 BLU in anderen Experimenten stieß zunächst auf Verwunderung. Weitere Untersuchungen zeigten jedoch, dass einzelne Operationen momentan noch nicht auf die spaltenorientierte Architektur angepasst worden sind und dadurch zusätzliche Konvertierungen in die zeilenorientierte Darstellung die Rechengeschwindigkeit drastisch reduzierten. Aus diesem Grund ist zu diesem Zeitpunkt ein Vergleich mit MonetDB nicht fair.

Wie bereits erwähnt stellte sich heraus, dass spaltenorientierte Architekturen für wissenschaftliche Berechnungen deutlich günstiger sind. Als Vertreter dieser bietet MonetDB jedoch noch zusätzlich ausgeprägte Techniken (siehe etwa [11, 16]) und zeigt in vielen Benchmarks beeindruckende Ergebnisse. Dies macht es zu unserem Datenbanksystem der Wahl.

Abschließend sei erwähnt, dass die oben beschriebene Methode zur Berechnung der Regressionskoeffizienten eine nicht stabile Variante ist und weiterhin auch nicht in dieser Form in R genutzt wird.

Typischerweise wird eine QR-Matrixfaktorisierung der zugehörigen Designmatrix $A \in \mathbb{R}^{n \times 2}$ erstellt um eine stabile Berechnung der Koeffizienten zu ermöglichen. Dafür wird das zugehörige lineare Gleichungssystem

$$Rw = Q^T b$$

gelöst, wobei $w \in \mathbb{R}^2$ die gesuchten Koeffizienten beinhaltet und $b \in \mathbb{R}^n$ die Sensorantworten.

Der von uns durchgeführte Ansatz kann sich jedoch für kleine Teildatensätze, wie sie bei *Rolling-Window-Methoden* eingesetzt werden, als durchaus günstig erweisen. Trotzdem ist es wichtig stabile Methoden bereitzustellen, etwa durch Einpflegen von Matrixfaktorisierungen und andere Möglichkeiten, wie die Kahan-Summation, da gerade bei der Verarbeitung großer Datensätze Rundungsfehler verheerende Einflüsse haben können.

4. AUSBLICK

Da unser Projekt noch am Anfang steht, ist noch ein Großteil an Forschungspunkten offen. Der unmittelbar nächste Schritt ist das gezielte Aufstellen und Auswerten von Experimenten, die verschiedenste Methoden auf R, der Datenbank und dem Cluster berechnen lassen. Dabei konzentrieren wir uns auf komplexere Methoden wie das Aufstellen linearer Modelle (`lm`). Dafür müssen weiterhin Matrixfaktorisierungen und Methoden wie die Kahan-Summation auf SQL implementiert werden. Für solche Methoden werden dann Teilablaufpläne entwickelt und Strategien, diese mit anderen Plänen zu verbinden und optimieren. Hierfür wird das Aufstellen von Kosten- und Schätzfunktion ein weiterer wichtiger Aspekt unserer Forschung sein.

Schließlich werden die Parallelisierungsmethoden auf Schleifen ausgeweitet um fundamentale, statistische Ansätze, wie beispielsweise *leave-one-out Kreuzvalidierung*, zu unterstützen. Solche Ansätze sind besonders günstig für Datenbankunterstützungen, da sie häufig auf dieselben Daten zugreifen und damit für das mühsame Einspeisen der Daten entschädigen.

Das große Ziel unserer Forschung ist es, unsere Umgebung mit einer weiteren Schicht für die Datensicherheit (siehe [4]) zu kombinieren. Diese wird direkt vor der Datenbank platziert um mögliche Verletzungen der Datensicherheit von SQL-Anfragen durch Veränderung dieser zu meiden.

5. SCHLUSSWORT

In dem vorliegenden Artikel wurde unser Projekt, welches

das Statistik-Programm R, Clusterparallelisierung und Datenbanktechnolgie vereinigt, vorgestellt. Ein entscheidender Aspekt ist hierbei die gezielte Abarbeitung bestimmter R-Methoden in SQL, wobei dieser Prozess automatisch, ohne manuelles Einwirken des Nutzers, geschehen soll. Für diese Arbeit wurden Vor- und Nachteile diskutiert, aufkommende Probleme beschrieben und ein Ausblick für unsere zukünftigen Teilprojekte gegeben.

6. DANKSAGUNG

Dennis Marten wird durch die Deutsche Forschungsgemeinschaft (DFG) im Rahmen des Graduiertenkollegs 1424 (Multimodal Smart Appliance Ensembles for Mobile Applications - MuSAMA) gefördert.

Für Ausarbeitungen in den oben beschriebenen Experimenten gilt Jan Svacina, Dennis Weu, Stefan Lüdtke, Pia Wilsdorf, Felix Köppl und Steffen Sachse unser Dank.

7. LITERATUR

[1] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, S. Vaithyanathan. SystemML: Declarative Machine Learning on MapReduce. In *2011 IEEE 27th International Conference on Data Engineering*.

[2] Apache. Hadoop. `http://hadoop.apache.org`.

[3] B. Huang, S. Babu, J. Yang. Cumulon: optimizing statistical data analysis in the cloud. In *2013 ACM SIGMOD/PODS*.

[4] H. Grunert. Distributed denial of privacy. In *Jahrestagung der Gesellschaft für Informatik (INFORMATIK 2014)*.

[5] S. Guha. *Computing Environment for the statistical analysis of large and complex data*. PhD thesis, Purdue University, 2010.

[6] H. Wickham, R. Francois, RStudio. dplyr: A grammar of data manipulation, 2015.

[7] J. Dean, S. Ghemawat. MapReduce: A flexible data processing tool. *Communications of the ACM*, 53(1):66–71, January 2010.

[8] J. Lajus, H. Mühleisen. Efficient data management and statistics with zero-copy integration. In *International Conference on Scientific and Statistical Database Management*, volume 24, 2014.

[9] J. Li, X. Ma, S. Yoginath, G. Kora, N. F. Samatova. Transparent runtime parallelization of the R script language. *Journal of Parallel and Distributed Computing*, 2011.

[10] M. Boehm, S. Tatikonda, B. Reinwald, P. Sen, Y. Tian, D. Burdick, S. Vaithyanathan. Hybrid Parallelization Strategies for Large-Scale Machine learning in systemml. *Proceedings of the VLDB Endowment*, 2014.

[11] M. G. Ivanova, M. L. Kersten, N. J. Nes, R. A. P. Goncalves. An architecture for recycling intermediates in a column-store. *ACM Trans. Database Syst.*, 35(4):24, 2010.

[12] M. Stonebraker, D. Abadi, D. J. Dewitt, S. Madden, E. Paulson, A. Pavlo, A. Rasin. MapReduce and parallel DBMSs: Friends or foes? *Communications of the ACM*, 53(1):66–71, January 2010.

[13] Paradigm4. SciDB. `http://www.scidb.org`.

[14] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014.

[15] S. Das, Y. Sismanis, K. S. Beyer, R. Gemulla, P. J. Haas, J. McPherson. Ricardo: Integrating R and Hadoop. In *2010 ACM SIGMOD*.

[16] S. Idreos, M. L. Kersten, S. Manegold. Self-organizing tuple reconstruction in column-stores. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 297–308, 2009.

[17] Y. Zhang, H. Herodotou, J. Yang. RIOT: I/O-efficient numerical computing without SQL. In *4th Biennial Conference on Innovative Data Systems Research*, 2009.

APPENDIX

Im Folgenden ist der SQL-Code eines *Rolling-Window-Ansatzes* zur Berechnung von linearen Regressionen dargestellt. Hierbei wurde das iterierende Fenster durch `WITH RECURSIVE` umgesetzt. Die Fenstergröße wurde im Quelltext auf 5 gesetzt. Je nach Kompatibilität kann der Anstieg mittels SQL:2003-Kommandos (`regr_slope(y,x)`) oder wie bereits erklärt, ausprogrammiert berechnet werden.

```
WITH RECURSIVE retest (n, koeff, anstieg) AS
(   SELECT 0,
    0::double precision,
    0::double precision
    UNION ALL
    SELECT retest.n+1,
    (
        SELECT koeff
        FROM
        (
            SELECT regr_intercept(y,x) AS koeff
            FROM
            (
                SELECT n AS grpe, time -99999 AS x,
                Segment_RightUpperLeg_Position_Z AS y
                FROM test
                LIMIT 5 OFFSET n
            ) AS sublv12a
            GROUP BY grpe
        ) AS sublvl1a
    ),
    (   SELECT anstieg
        FROM
        (
            SELECT regr_slope(y, x) AS anstieg
            FROM
            (
                SELECT n AS grpe, time -99999 AS x,
                Segment_RightUpperLeg_Position_Z AS y
                FROM test
                LIMIT 5 OFFSET n
            ) AS sublv12b
            GROUP BY grpe
        ) AS sublvl1b
    ) FROM retest
    WHERE n < (SELECT COUNT(time) FROM test)
    ) SELECT * FROM retest OFFSET 1;
```