Visualisierung von NoSQL-Transformationen unter der Verwendung von Sampling-Techniken

Stefan Braun, Johannes Schildgen, Stefan Deßloch

{s_braun10, schildgen, dessloch}@cs.uni-kl.de
AG Heterogene Informationssysteme, Fachbereich Informatik, TU Kaiserslautern

Zusammenfassung. Analysen auf NoSQL-Datenbanken sind oft langdauernd und die Ergebnisse für den Benutzer häufig schwer verständlich. Wir präsentieren eine Möglichkeit, Datenmengen aus Wide-Column Stores mittels der Transformationssprache NotaQL zu transformieren sowie zu aggregieren und die Ergebnisse in Form von Diagrammen dem Benutzer darzustellen. Dabei kommen Sampling-Techniken zum Einsatz, um die Berechnung auf Kosten der Genauigkeit zu beschleunigen. Das von uns verwendete iterative Samplingverfahren sorgt für eine kontinuierliche Verbesserung der Berechnungsgenauigkeit und bietet zudem Möglichkeiten zur Genauigkeitsabschätzung, die in Form von Konfidenzintervallen in den Diagrammen dargestellt werden kann.

1 Einführung

Big-Data-Analysen sind Prozesse, die aufgrund großer Datenvolumina oft viele Minuten oder Stunden in Anspruch nehmen. Zudem sind die Ergebnisse solcher Analysen für den Menschen oft schwierig zu interpretieren, da sowohl die Datenbasis als auch das Analyseergebnis im Wesentlichen unstrukturierte Daten oder die Aggregation vieler nummerischer Werte sind. Um dem Benutzer die Datenmengen adäquat zu visualisieren, sind oft einfache Werkzeuge wie Kreis-, Balken- oder Liniendiagramme eine gute Wahl. Sie geben einen Überblick über die Datenverteilung, Zusammenhänge und über zeitliche Verläufe.

Um die Analysen verteilt auf großen Rechenclustern auszuführen, wird oft auf Verarbeitungs-Frameworks wie MapReduce [6] (bzw. Hadoop [1]) oder Spark [19] zurückgegriffen. Die verteilte Speicherung übernimmt in diesen Fällen entweder ein verteiltes Dateisystem oder ein NoSQL-Datenbanksystem. Ersteres eignet sich besonders für die Analyse von Log-Dateien, letzteres für heterogene Datensammlungen ohne fixes Schema. Die verschiedenen Arten von NoSQL-Systemen [4] bieten unterschiedliche Datenmodelle, von Sammlungen einfacher Schlüssel-Wert-Paare (*Key-Value Stores*), über Tabellen mit flexiblen Spalten (*Wide-Column Stores*) bis hin zur Speicherung komplexer Dokumente (*Document Stores*). Diese drei Arten werden die Aggregat-orientierten Datenbanken

 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

genannt und haben gemeinsam, dass jeder Eintrag über eine eindeutige ID identifiziert wird. Über diese ID erfolgt auch die Partitionierung; ein impliziter Index darauf erlaubt effiziente Lese- und Schreibzugriffe. Da viele NoSQL-Datenbanksysteme keine komplexen Anfragen erlauben, werden oft Datentransformationen mittels MapReduce oder höheren Sprachen wie Pig [13], Hive [18], Phoenix [3] oder NotaQL [15] durchgeführt. Während Hive und Phoenix einen SQL-artigen Zugriff auf die Daten erlauben, bieten Pig und NotaQL weitere Transformationsmöglichkeiten, die aufgrund der Schema-Flexibilität in NoSQL-Datenbanken vonnöten sind.

Diese Arbeit beschäftigt sich mit der Visualisierung von Daten, die im Wide-Column Store *HBase* [2] gespeichert sind. Mithilfe von Transformationsskripten, die in der Sprache NotaQL formuliert werden, können Eingabedaten zunächst gefiltert, transformiert und aggregiert werden. Das Resultat wird in Form von Diagrammen dem Benutzer präsentiert. Da die Transformationen sehr lange dauern können und in Diagrammen oft keine hundertprozentige Genauigkeit erforderlich ist, schlagen wir die Verwendung von Sampling-Techniken vor. Durch das Ermitteln von Zufallsstichproben in den Eingabedaten wird die Berechnung beschleunigt, sodass der Benutzer bereits nach kurzer Zeit das Resultat in Form von Kreis-, Balken- oder Liniendiagrammen sehen kann. Der Hauptfokus dieser Arbeit liegt auf der Anwendung von iterativen Samplingprozessen bei NoSQL-Datentransformationen sowie dem Zusammenspiel von Sampling-, Visualisierungs- und Ungenauigkeitsbestimmungstechniken.

Im folgenden Kapitel stellen wir Sampling-Ansätze vor und erläutern, wie sich mit statistischen Methoden Abschätzungen über die Genauigkeit machen lassen. Weiterhin präsentieren wir die Sprache NotaQL, mit der Transformationen auf der HBase-Datenbank ausgeführt werden können. In Kapitel 3 stellen wir ein Visualisierungswerkzeug vor, welches mittels Sampling-Methoden NotaQL-Transformationen durchführt und in Form von Diagrammen visualisiert. Dort wird erläutert, wie mithilfe von Whiskers die Berechnungsgenauigkeit im Diagramm dargestellt werden kann und wie ein iterativer Transformationsprozess diese Genauigkeit kontinuierlich steigern lässt. Kapitel 4 beinhaltet Ergebnisse von Experimenten, die die Performanz des iterativen Samplingprozesses analysieren. Nach einer Vorstellung verwandter Arbeiten in Kapitel 5 folgt eine Zusammenfassung in Kapitel 6.

2 Grundlagen

Dieses Kapitel beinhaltet die mathematischen und technischen Grundlagen zur Ausführung von Sampling-basierten Tabellentransformationen, die für die Visualisierung genutzt werden. Wide-Column Stores bieten eine simple API, um komplette Tabellen zu scannen und bestimmte Zeilen anhand ihrer ID (*row-id*) abzurufen. Wegen des flexiblen Datenmodells kann keine Aussage über die Spaltennamen einer Tabelle gemacht werden. Aus diesen Gründen kommt im Rahmen dieser Arbeit kein SQL zum Einsatz, sondern die Transformationssprache NotaQL.

In Tabelle 1 wird der zeitliche Verlauf von Spritpreisen gespeichert. Die Tabelle besteht aus zwei sogenannten Spaltenfamilien, die beim Erstellen der Tabelle definiert werden. Die erste Spaltenfamilie „Spritpreise“ zeigt die Spritsorten, die eine Tankstelle anbietet, sowie deren Preise. Die zweite Spaltenfamilie „Informationen“ hingegen listet den Tankstellennamen sowie die Straße auf.

row_id	**Spritpreise**		**Informationen**	
	Diesel	SuperE10	Tankstelle	Straße
2014-11-07 12:32:00	1,319	1,489	FillItUp	Rue de Gaulle
	Diesel		Tankstelle	Straße
2014-11-07 21:30:00	1,409		FillItUp	Rotweg
	Diesel	SuperE10	Tankstelle	Straße
2014-11-08 04:30:00	1,409	1,509	FillItUp	Rue de Gaulle
	Diesel		Tankstelle	Straße
2014-11-08 05:30:00	1,389		FillItUp	Rotweg

Tabelle 1. Wide-Column Tabelle mit zwei Spaltenfamilien.

2.1 NotaQL

In [15] wird die Datentransformationssprache NotaQL vorgestellt. Sie ermöglicht das Ausdrücken vieler MapReduce-Algorithmen anhand von zwei oder drei Zeilen Code. Im Gegensatz zu Phoenix oder Hive kann direkt auf den Tabellen eines Wide-Column Stores gearbeitet werden, ohne dass vorher ein Tabellenschema definiert werden muss. NotaQL dient im Grunde zur Erstellung einer Vorschrift, wie eine *Output-Zelle* anhand des Inputs gebildet werden soll, wobei eine Zelle die Kombination aus einer row-id und einem Spaltennamen $(\_r, \_c)$ bildet. Da jede von ihnen einen atomaren Wert besitzt, repräsentiert das Tupel $(\_r, \_c, \_v)$ die Verknüpfung der Zelle mit ihrem Wert. Existieren mehrere Spaltenfamilien in der Tabelle, können die Namen der Spaltenfamilien als Präfix für den Spaltennamen verwendet werden, z.B. `Informationen:Straße` anstatt `Straße`.
Die Möglichkeit zur Selektion von Zeilen wird durch die *IN-FILTER*-Klausel gegeben. Sie definiert einen optionalen Filter am Anfang eines NotaQL-Skripts.

Folgendes ist ein NotaQL-Skript, welches angewandt auf Tabelle 1 zur Berechnung des Durchschnittspreises für die einzelnen Spritsorten der Tankstellenkette „FillItUp“ dient:

```
IN-FILTER: Informationen:Tankstelle = 'FillItUp',
OUT._r <- IN.Spritpreise:_c,
OUT.aggr:AVGPreis <- AVG(IN.Spritpreise:_v);
```

Das Skript ist wie folgt zu verstehen: Die erste Zeile führt eine Zeilenselektierung durch. Es werden nur Zeilen mit dem Wert „FillItUp“ in dem Spaltennamen „Tankstelle“ der Spaltenfamilie „Informationen“ ausgewählt. In der zweiten Zeile wird beschrieben, dass die Spaltennamen der Spaltenfamilie „Spritpreise“ (`IN.Spritpreise:_c`) die neuen row-ids (`OUT._r`) sind. Es wird also für jeden distinkten Spaltennamen in dieser Spaltenfamilie eine Zei-

le in der Ausgabetabelle erzeugt. Die dritte Zeile beschreibt nun die Anwendung der Aggregatfunktion *AVG* auf die Werte der Spaltenfamilie „Spritpreise“ (`AVG(IN.Spritpreise:_v)`); also eine Ermittlung des Durchschnittspreises je Sorte. Dieser Wert wird dann in der Spaltenfamilie „aggr“ unter dem Spaltennamen „AVGPreis“ (`OUT.aggr:AVGPreis`) abgelegt. Das Ergebnis der Transformation ist in Tabelle 2 zu sehen.

row_id	**aggr**
	AVGPreis
Diesel	1,3815
	AVGPreis
SuperE10	1,499

Tabelle 2. Transformationsergebnis

2.2 Sampling

Durch die Verwendung von *Sampling* lassen sich Datentransformationen auf Kosten der Berechnungsgenauigkeit um einen beliebigen Faktor beschleunigen. Sampling wird verwendet, um eine *Stichprobe*, also eine Teilmenge, aus einer *Grundgesamtheit* zu ziehen und anhand dieser Statistiken, wie Ergebnisse von Aggregatfunktionen, der Grundgesamtheit abzuschätzen. Je repräsentativer eine solche Teilmenge ist, also je mehr ihr prozentualer Aufbau dem der Grundgesamtheit gleicht, desto genauere Hochrechnungen und Schätzungen erlaubt sie. Es existieren verschiedene Techniken zum Ziehen von Stichproben [16]. Im Folgenden wird die *einfache Zufallsstichprobe* und das *iterative Sampling* kurz erläutert.

Einfache Zufallsstichprobe Auch bekannt als *Simple Random Sampling (SRS)*. Diese Technik kann mit und ohne Zurücklegen der Elemente durchgeführt werden. Wir betrachten letzteres, da durch das Vermeiden von Duplikaten in der Stichprobe im Allgemeinen eine höhere Genauigkeit erreicht wird. Jedes Element der Datenmenge hat die gleiche Wahrscheinlichkeit $p\%$, um in die Stichprobe aufgenommen zu werden. Laut dem *Gesetz der großen Zahlen* beträgt der Stichprobenumfang n somit $N \cdot p/100$, wobei N dem Umfang der Datenmenge entspricht. Die Vorteile dieser Technik liegen in der einfachen Umsetzbarkeit, und dass keine weitere Informationen, wie die Häufigkeitsverteilung des Merkmals, über die Datenmenge vorliegen müssen. Außerdem ist diese Technik *unbiased*, das heißt, dass kein Element bei der Auswahl bevorzugt wird, was zu einer Verzerrung der Häufigkeitsverteilung in der Stichprobe führen könnte.

Iteratives Sampling Der Samplingprozess besteht aus mehreren Iterationen von einfachen Zufallsstichproben. Es empfiehlt sich eine kleine Startgröße zu wählen, um bereits nach kurzer Zeit Ergebnisse zu sehen. Die iterative Ausführung mit immer größer werdender Stichprobengröße sorgt nicht nur für die kontinuierliche Verbesserung der Berechnungsgenauigkeit, sondern liefert auch die notwendigen Informationen, um ebendiese Genauigkeit mathematisch berechnen zu

können. Der Prozess kann abgebrochen werden, wenn für den Nutzer eine ausreichende Genauigkeit erzielt wurde. Alternativ ist der Prozess dann beendet, sobald die Stichprobengröße 100% der Grundmenge beträgt. In [12] wird erläutert, wie das Ergebnis der vorherigen Iteration für die darauf folgende wiederverwendet werden kann. Dieses Verfahren beschleunigt die Berechnung, erhöht jedoch die Ungenauigkeit, da es sich in diesem Fall um Sampling mit Zurücklegen handelt.

Hochrechnung Bei der Verwendung der Aggregatfunktionen `SUM` und `COUNT` ist das Ergebnis einer Berechnung nicht direkt aussagekräftig für die Grundgesamtheit, wenn die Berechnung nur eine Teilmenge betrachtet hatte. Das Ergebnis muss somit zuerst mit dem Faktor $100/p$ hochgerechnet werden, wobei p der Samplingwahrscheinlichkeit entspricht.

Konfidenzintervalle Durch die Verwendung von Sampling und beim Hochrechnen der Ergebnisse von Aggregatfunktionen ergibt sich eine gewissen Berechnungsungenauigkeit. Um diese Ungenauigkeit auszudrücken, werden *ci*%-*Konfidenzintervalle* verwendet. Dabei gilt, dass je größer *ci*% ist, desto größer ist auch das Intervall. Beispielsweise wird Tabelle 1 als Stichprobe betrachtet und anhand dieser Daten der Durchschnittspreis für Diesel geschätzt. Dieser Durchschnittswert, genannt *Stichprobenmittel* $\bar{x}$, ist in Tabelle 2 gelistet. Die Berechnung eines 50%-Konfidenzintervalles liefert nun das Intervall [1,3513, 1,4117], ein 95%-Konfidenzintervall liefert hingegen die etwas breiteren Grenzen [1,286, 1,477]. Die Interpretation ist wie folgt: Würde das Stichprobenziehen und Anwenden derselben Transformation immer wieder wiederholt werden und jedes mal ein *ci*%-Konfidenzintervall berechnet werden, so würden *ci*% der Intervalle den wahren Wert beinhalten. Wobei *ci* während den Wiederholungen immer den gleichen Wert hat.

Für die Berechnung der Intervalle wird neben dem Stichprobenmittel $\bar{x}$ auch die Varianz der Stichprobe, genannt *Stichprobenvarianz* s_n^2, benötigt. Dieser Wert ist ein Maß für die Streuung der Daten in der Stichprobe.

$$\text{Mittelwert:} \quad \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \text{Varianz:} \quad s_n^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2 \tag{1}$$

Dabei entspricht n dem Stichprobenumfang und x_i dem i-ten Element in der Stichprobe. Die Stichprobenvarianz weist weiterhin eine systematische, nichtzufällige Abweichung auf. Das Ausmaß dieser Abweichung wird *Bias* oder Verzerrung genannt. Zur Korrektur wird der Faktor $n/(n-1)$ verwendet. Dieser Faktor wird als *Bessel-Korrektur* bezeichnet. Daraus berechnet sich dann die *korrigierte Stichprobenvarianz* s^2.

$$s^2 = \frac{n}{n-1} s_n^2 \tag{2}$$

Die Quadratwurzel aus der korrigierten Stichprobenvarianz liefert die *Stichprobenstandardabweichung* σ. Diese ist ein Maß für die Streuung der Daten um das Stichprobenmittel.

$$\sigma = \sqrt{s^2} \tag{3}$$

Die Berechnung der Konfidenzintervalle erfolgt nun nach der *Chebyshev-Formel* [14]:

$$[\bar{x} - \sqrt{(1/\alpha)} \cdot \frac{\sigma}{\sqrt{n}}, \bar{x} + \sqrt{(1/\alpha)} \cdot \frac{\sigma}{\sqrt{n}}] \tag{4}$$

α dient dabei als Genauigkeitsregulator. Für ein 95%-Konfidenzintervall beträgt $\alpha = 1$ - $0.95 = 0.05$, für ein 90% entsprechend $\alpha = 1$ - $0.90 = 0.10$. Allgemein ausgedrückt:

$$\alpha = 1 - ci\% \tag{5}$$

Aktuell gilt die Formel 4 nur für die Aggregatfunktion *AVG*. Eine Erweiterung dieser für die Aggregatfunktion SUM ist möglich durch die Multiplikation der Grenzen mit der Anzahl N an Datensätzen in der Grundgesamtheit. N kann bei Verwendung der einfachen Zufallsstichprobe aus der Stichprobengröße n und der Sampling-Wahrscheinlichkeit p hergeleitet werden.

$$n = N \cdot \frac{p}{100} \implies N = n \cdot \frac{100}{p} \tag{6}$$

Die Chebyshev-Formel gilt als eine konservative Abschätzung, da sie meist Intervallgrenzen berechnet, die nicht nur das gewünschten Konfidenzniveau abdecken, sondern auch überschreiten. Es wird also meist ein zu großes Intervall berechnet. Jedoch kann es auch zu einer Unterschreitung des Niveaus kommen, wenn die geschätzte Standardabweichung stark von der wirklichen abweicht und diese darüber hinaus noch sehr groß ist. Der Vorteil dieser Formel ist die einfache Berechenbarkeit und die Anwendbarkeit auf jede beliebige Werteverteilung, solange das Stichprobenmittel sowie die Stichprobenvarianz berechenbar sind.

3 Visualisierung von Transformationsergebnissen

Das Ziel dieser Arbeit ist die Visualisierung von Daten aus einem Wide-Column Store, welche mit Hilfe von NotaQL-Skripten zuerst transformiert wurden. Der entsprechende Workflow wird in Abbildung 1 dargestellt. Gestrichelte Linien zeigen dabei Prozesse an, die nur im Zusammenhang mit einem iterativen Samplingprozess ausgeführt werden.

Zuerst erfolgt eine einfache Zufallsstichprobe mit der eingestellten Sampling-Wahrscheinlichkeit aus der Eingabetabelle. Die HBase API stellt hierfür die Klasse *RandomRowFilter* bereit. Die Daten werden anschließend gemäß der gegebenen NotaQL-Vorschrift transformiert [15] und - falls erforderlich - hochgerechnet. Außerdem werden die Ergebnisse für einen iterativen Samplingprozess, in Verbindung mit den Aggregatfunktionen, in einem *Uncertainity-Computer* abgelegt. Dieser dient unter anderem zur Berechnung von Konfidenzintervallen. Vor der

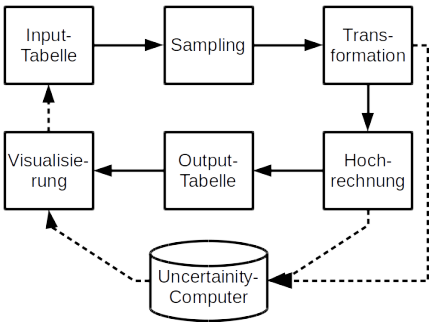

Abb. 1. Arbeitsworkflow

Visualisierung werden die Ergebnisse sortiert in der Ausgabetabelle abgelegt. Den letzten Schritt, die Erzeugung von Diagrammen, übernimmt die *Visualisierungskomponente*.

3.1 Uncertainity-Computer

Die Aggregatfunktionen AVG, COUNT, SUM, MAX, MIN werden in zwei Gruppen aufgeteilt. Gruppe eins besteht aus den beiden Funktionen MAX und MIN, deren Resultat ein bestimmtes Element der zu aggregierenden Werte darstellt. Die zweite Gruppe besteht aus den Funktionen AVG, COUNT, SUM, auf deren Resultat alle Werte einen direkten Einfluss haben.

MAX und MIN Die Funktionen der ersten Gruppe berechnen im Gegensatz zur zweiten Gruppe keinen Wert, sondern suchen den größten beziehungsweise kleinsten Wert aus einer Datenmenge heraus. Der gesuchte Wert ist also auf jeden Fall ein Bestandteil der Datenmenge. Werden diese Funktionen nun anhand einer Stichprobe ausgewertet, so kann man über den gefunden Wert nur sagen, dass der wahre Wert nicht kleiner (im Falle von MAX) oder größer (im Falle von MIN) ist. Weitere Aussagen anhand der Standardabweichung oder anderen Werten sind meist nicht aussagekräftig, da diese Funktionen die Ausreiser einer Verteilung als Ziel haben. Verringert sich jedoch ein mit der MAX-Funktion berechnetes Ergebnis in Iteration $i + 1$ gegenüber der vorherigen Iteration i, wird die vorherige Schätzung beibehalten, um eine Verschlechterung der Schätzwerte zu verhindern.

AVG, COUNT und SUM Die Funktionen der zweiten Gruppe berechnen anhand einer Datenmenge eine Statistik. Im vorherigen Kapitel wurden die Konfidenzintervalle bereits erklärt und gezeigt, wie diese für die Aggregatfunktionen AVG und SUM berechnet werden können. Um auch Konfidenzintervalle für die Aggregatfunktion COUNT angeben zu können, wird die Berechnung dieser Intervalle mit dem iterativen Samplingprozess verbunden. Zur Berechnung der Konfidenzintervalle verwaltet der Uncertainity-Computer für jede Output-Zelle die Anzahl der gespeicherten Werte sowie deren Summe und Quadratsumme. Daraus können bei Bedarf in konstanter Zeit die Konfidenzintervalle mit der Chebyshev-Formel (Gleichung 4) berechnet werden. Es werden somit mindestens zwei Werte, entsprechend mindestens zwei Samplingschritte, zur Berechnung benötigt, da sonst die Varianz 0 ist und somit kein Intervall berechnet werden kann. Dieses Berechnungsverfahren wird für alle Aggregatfunktionen dieser Gruppe verwendet.

3.2 Visualisierungskomponente

Diagramme wie Linien- oder Balkendiagramme dienen zur adäquaten Darstellung großer Datenmengen und bieten großen Modifikationsspielraum. So lassen

sich Ungenauigkeiten von Daten, also Konfidenzintervalle, mit Hilfe von Whiskers (Abbildung 2) oder Deviation Areas (Abbildung 3) darstellen. Erstere nutzen dafür einen kleinen vertikalen Balken und letztere heben das Konfidenzintervall farblich vom Hintergrund und anderen Linien ab. Während Whiskers auf beiden Diagrammtypen angewandt werden können, sind die Devation Areas nur für Liniendiagramme sinnvoll, da sie eine Interpretation des Zwischenraums erlauben. Zur Umsetzung der Diagramme haben wir die Java Bibliothek *JFreeChart*[1] verwendet und erweitert [17].

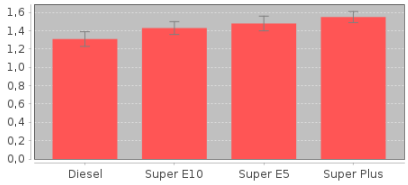

Abb. 2. Balkendiagramm mit einem Whisker pro Statistik.

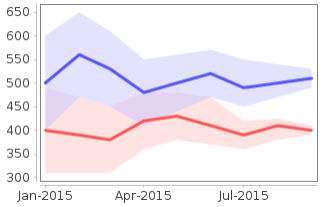

Abb. 3. Liniendiagramm mit einer Deviation Area pro Kurve.

4 Experimente

In diesem Kapitel demonstrieren wir anhand von Experimenten zum einen die Genauigkeit, die sich durch die Verwendung von iterativen Samplingtechniken erreichen lässt. Zum anderen analysieren wir die Performanz dieses Prozesses und vergleichen die Laufzeiten mit der einer vollständigen Berechnung. Dazu betrachten wir zuerst anhand eines Tests mit Startsamplinggröße 1%, die pro Iteration um 1% steigt, den Verlauf der Konfidenzintervallgrenzen und im darauffolgenden Test wird die Verwendung verschiedener Startsamplinggrößen mit entsprechender Erhöhung pro Iteration evaluiert.

Wie in Abbildung 4 zu sehen ist, wächst mit steigendem Iterationsdurchlauf, also auch steigender Sampling-Wahrscheinlichkeit, die Berechnungsgenauigkeit der verwendeten Aggregatfunktion. Dadurch sinkt nicht nur im Mittel die Abweichung zwischen berechnetem und wahrem Wert, sondern auch die berechnete Standardabweichung, wodurch die Chebyshev-Formel anwendbar ist. Der berechnete Wert konvergiert also gegen den wahren Wert und somit konvergieren auch die Intervallgrenzen gegen den wahren Wert. Die Abbildung stellt einen iterativen Samplingprozess mit Startwahrscheinlichkeit 1% auf der Tabelle 1 dar. Die verwendete Transformationsvorschrift sorgt für die Berechnung des Durchschnittspreises für Diesel über den kompletten Zeitraum und alle Tankstellen: `OUT._r <- 'Diesel', OUT.Preis <- AVG(IN.Diesel)`. Die grüne und die gelbe Linie stellen den Werteverlauf der oberen bzw. unteren Grenze des 95%-Konfidenzintervalls dar. Die blaue Linie markiert den tatsächlichen Durchschnittspreis und die rote Linie den Verlauf der berechneten Durchschnittspreise. Auffällig ist, dass manche berechneten Werte außerhalb des Konfidenzintervalls liegen. So zum Beispiel der Wert, der mit einer Sampling-Wahrscheinlichkeit von

[1] `http://www.jfree.org/jfreechart/index.html`

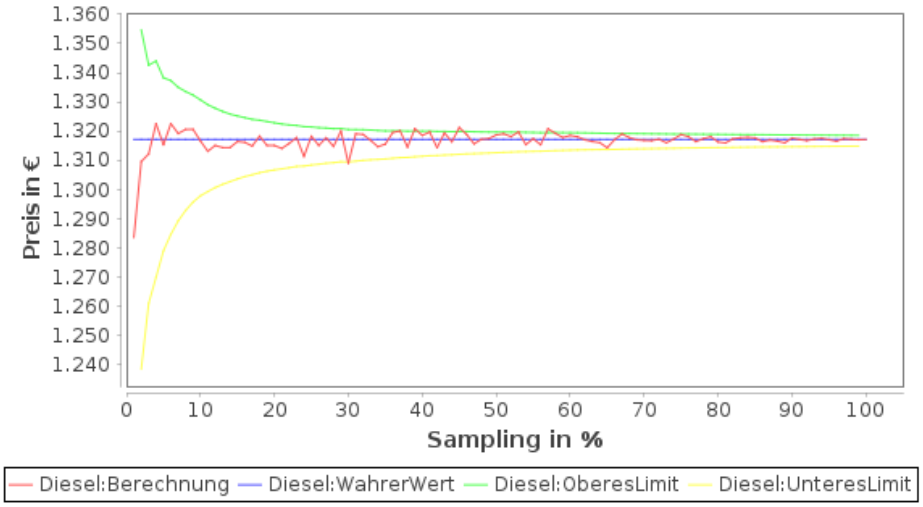

Abb. 4. Darstellung des Verlaufs der Werte des berechneten Dieseldurchschnittpreises (rot), des 95%-Konfidenzintervalls (grün, gelb) und des wahren Wertes (blau) nach Anwendung eines iterativen Samplingprozess auf Tabelle 1.

30% entstanden ist. Die Abweichung dieses Wertes zur Ideallinie ist, im Vergleich zu seinen Vor- und Nachgängern, wesentlich größer. Dies lässt sich dadurch erklären, dass die Stichprobe die Verteilung der Datenmenge ungenauer dargestellt hat. Es wurden also prozentual wesentlich mehr unterdurchschnittliche Werte in die Stichprobe eingelesen, als wirklich in der Datenmenge vorhanden sind. Es ist also eine zufällige Verzerrung aufgetreten. Obwohl der berechnete Wert außerhalb des Konfidenzintervalls liegt, enthält dieses in jedem Iterationsschritt dennoch den wahren Wert. Weiterhin ist zu erkennen, dass für den ersten Iterationsschritt und für das komplette Auslesen der Datenmenge, wie bereits erwähnt, kein Konfidenzintervall berechnet wurde.

Für das nächste Experiment wurde die folgende Transformationsvorschrift verwendet: `OUT._r <- IN._c, OUT.avg <- AVG(IN._v);` Diese führt für jeden Spaltennamen eine Durchschnittswertberechnung aus.

Eine HBase-Tabelle wurde von einem Datengenerator mit zehn Million Zeilen gefüllt, von denen jede drei Spalten mit Zufallszahlenwerten im Intervall $[0, 1500]$ haben. Zudem wurden weitere Testtabellen generiert und dabei die Anzahl der Zeilen und Spalten variiert. Alle Tests wurden auf einem Zweikern-Prozessor (je 2,1 GHz) sowie 3,9 GB RAM durchgeführt. Als Wide-Column Store wurde eine HBase Standalone Datenbank (Version 0.98.7) verwendet.

Abbildung 5 zeigt die Abhängigkeit der Berechnungsdauer von der Samplinggröße. Es fällt

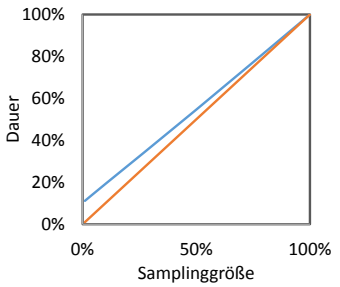

Abb. 5. Berechnungsdauer relativ zur vollständigen Berechnung (blaue Linie). Zum Vergleich die Optimallinie (orange).

auf, dass bereits eine 1% Samplingberechnung zehn Prozent der Dauer im Vergleich zur vollständigen Berechnung benötigt. Mit zunehmender Samplinggröße nähert sich der Zusammenhang nahezu der Optimallinie.

Im Folgenden wird der iterative Samplingprozess auf der gleichen Tabelle (10 Mio. Zeilen, 3 Spalten) genauer untersucht. Dazu wurden auf dieser Tabelle vier iterative Samplingprozesse mit den Startwahrscheinlichkeiten 1%, 5%, 10% und 20%, mit entsprechender prozentualen Erhöhung pro weiterem Iterationsschritt, durchgeführt (siehe Abbildung 6). Zum Vergleich wurde eine vollständige Transformation ohne die Verwendung von Sampling ausgeführt. Diese dauerte knapp zehn Minuten und ist als rote Linie in Abbildung 6 eingezeichnet. Ein iterativer Samplingprozess besitzt gegenüber der Verarbeitung ohne Sampling einen zeitlichen Vorteil, solange seine kumulierte Dauer geringer ist als die Dauer für die komplette Verarbeitung. Die Verwendung der Startwahrscheinlichkeit 1% liefert zwar am schnellsten die ersten Ergebnisse, jedoch sind diese recht ungenau und liefern somit nur eine grobe Approximation. Nach acht Schritten hat der Sampling-Prozess seinen Zeitvorteil verloren und nur 8% der Datenmenge im letzten Schritt verarbeitet. Mit steigender Startwahrscheinlichkeit erhöht sich die Dauer bis zum ersten Ergebnis, dafür steigt aber auch die maximal mögliche Anzahl an ausgelesenen Daten. So ist mit einer Startwahrscheinlichkeit von 20% die Ausführung von bis zu zwei Iterationsschritten sinnvoll und somit werden 40% der Datenmenge im zweiten Schritt ausgelesen. Wie in Abbildung 4 zu sehen ist, ist die Berechnungsgenauigkeit bereits bei einer Samplinggröße von 10 bis 15% für viele Anwendungen ausreichend. Diese Größe ist nach zwei bis drei Iterationen zu je 5% erreicht, was in der Hälfte der Zeit gegenüber einer vollständigen Berechnung ausgeführt werden kann. Die direkte Wahl einer größeren Samplinggröße würde zwar die gleichen Ergebnisse bereits nach kürzerer Zeit und nur einer Iteration liefern. Dafür kann dem Benutzer allerdings keine Information über die Berechnungsgenauigkeit mittels Whiskers und Deviation Areas gegeben werden.

Die Wahl der Startwahrscheinlichkeit hängt somit von der gewünschten Berechnungsdauer für das erste Ergebnis und der Approximationsgenauigkeit ab. Dabei bezieht sich die Approximationsgenauigkeit auf die maximale Sampling-Wahrscheinlichkeit und etwaige berechnete Konfidenzintervalle.

5 Verwandte Arbeiten

Sowohl Datenvisualisierungen als auch Sampling-Verfahren kommen oft zum Einsatz, wenn es um die Analyse großer Datenmengen geht. In [11] wird auf die Wichtigkeit hingewiesen, bei der Verwendung von Sampling darauf zu achten, dass die Stichproben die Charakteristika der Originaldaten widerspiegeln. Der in diesem Artikel vorgestellte Ansatz verwendet zufällige Stichproben anhand der Zeilen auf einer HBase-Tabelle. Gegenüber komplexeren Strukturen wie Graph-Datenbanken sowie Tabellen, die über Join-Pfäde verbunden werden müssen, können hier Datensätze weitestgehend unabhängig voneinander betrachtet werden, was eine höhere Genauigkeit zur Folge hat.

EARL [10] ist eine auf Hadoop-basierende Sampling-Bibliothek, die Bootstrapping [7–9] verwendet, um Aussagen über die Genauigkeit einer Berechnung

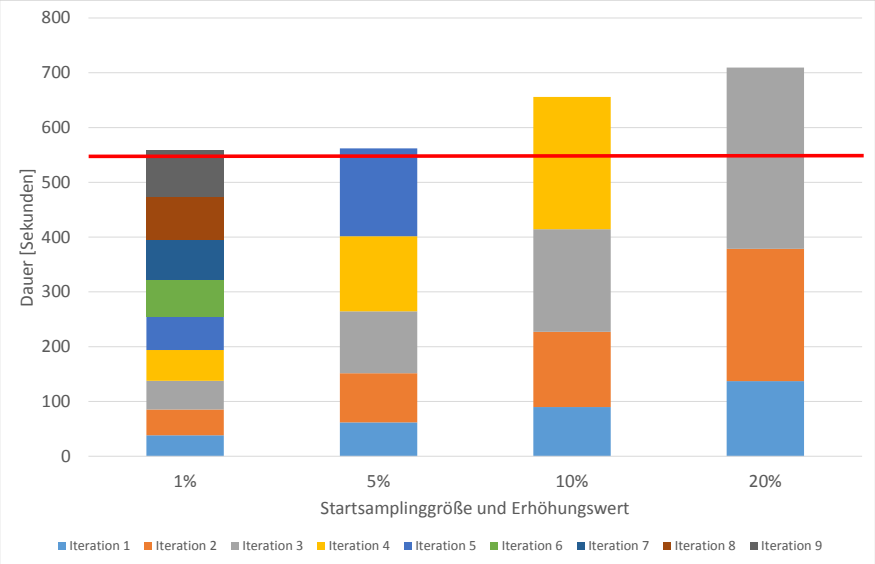

Abb. 6. Ausführung mehrerer Iterationen von Transformationen mit unterschiedlichen Samplinggrößen sowie einer vollständigen Berechnung (rote Linie).

zu machen. EARL führt dabei Transformationen kontinuierlich durch. Unser Ansatz dagegen ist iterativ, was den Vorteil hat, dass auch ohne Bootstrapping eine Genauigkeitschätzung möglich ist, nämlich über die Varianz der Ergebnisse verschiedener Iterationen. Beim Bootstrapping wird eine Stichprobe in Unterstichproben zerlegt, sodass die die aggregierten Resultate auf diesen kleineren Mengen basieren.

In [5] werden Techniken zur Visualisierung von SQL-Anfrageergebnissen vorgestellt. Der Autor verwendet verschiedene Darstellungsformen wie Whiskers und Deviation Areas, um die Berechnungsunsicherheit darzustellen.

6 Zusammenfassung

Wir haben gezeigt, dass bei der Ausführung von Tabellentransformationen Sampling-Techniken dazu beitragen können, früh erste Ergebnisse zu sehen. Eine Visualisierungssystem ist in der Lage, HBase-Tabellen als Linien-, Balken- oder Kreisdiagramm darzustellen sowie weitere Iterationen auf vergrößerten Stichproben im Hintergrund weiterlaufen zu lassen. Die Ergebnisse mehrerer Iterationen können für die Genauigkeitsberechnung verwendet werden, die mittels Whisters und Deviation Areas in die dargestellten Diagramme eingezeichnet werden können. Dadurch erhält der Benutzer nicht nur nach kurzer Zeit erste Ergebnisse einer Berechnung, sondern auch eine Information über deren Genauigkeit und die Möglichkeit, eine Berechnung vorzeitig abzubrechen, wenn die gewünschte Genauigkeit erreicht ist. Wir haben mittels Experimenten gezeigt, dass dieses Vorgehen deutlich schneller ist als eine vollständige Berechnung, und dass die Ungenauigkeit bereits bei geringen Stichprobengrößen so minimal ist, dass die

Diagramme, die dem Benutzer gezeigt werden, im Wesentlichen so aussehen, als würden sie auf der kompletten Datenbasis basieren.

Literatur

1. Apache Hadoop project. `http://hadoop.apache.org/`.
2. Apache HBase. `http://hbase.apache.org/`.
3. Apache Phoenix. `http://phoenix.apache.org/`.
4. Rick Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.
5. Danyel Fisher. Incremental, Approximate Database Queries and Uncertainty for Exploratory Visualization. In *IEEE Symposium on Large Data Analysis and Visualization*. IEEE, October 2011.
6. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
7. B. Efron. Bootstrap methods: Another look at the jackknife. *Ann. Statist.*, 7(1):1–26, 01 1979.
8. Michael I. Jordan. Divide-and-conquer and statistical inference for big data. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 4–4, New York, NY, USA, 2012. ACM.
9. Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael Jordan. The big data bootstrap. *arXiv preprint arXiv:1206.6415*, 2012.
10. Nikolay Laptev, Kai Zeng, and Carlo Zaniolo. Early accurate results for advanced analytics on mapreduce. *Proceedings of the VLDB Endowment*, 5(10):1028–1039, 2012.
11. Shuai Ma, Jia Li, Chunming Hu, Xuelian Lin, and Jinpeng Huai. Big graph search: challenges and techniques. *Frontiers of Computer Science*, pages 1–12, 2014.
12. Marc Schäfer, Johannes Schildgen, Stefan Deßloch. Sampling with Incremental MapReduce. *Workshop on Big Data in Science (BigDS), BTW*, 2015.
13. Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.
14. S. Acharaya, P. B. Gibbons, V. Poosala , S. Ramaswamy. Join Synopses for Approximate Query Answering. Technical report, Bell Laboratories, Murray Hill, New Jersey, 1999. Full version of the paper appearing in SIGMOD'99.
15. Johannes Schildgen and Stefan Deßloch. NotaQL Is Not a Query Language! It's for Data Transformation on Wide-Column Stores. In *British International Conference on Databases - BICOD 2015*, 7 2015.
16. Seymour Sudman. Applied sampling. *Academic Press New York*, 1976.
17. Stefan Braun. Visualisierung von NotaQL-Transformationen unter der Verwendung von Sampling-Techniken. Bachelorarbeit TU Kaiserslautern, 2015.
18. Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.
19. Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.