Unkonventionelle Anfrageverarbeitung im Big-Data-Umfeld

Alexander Ponomarenko

alexander.ponomarenko@haw-hamburg.de
Hochschule für Angewandte Wissenschaften Hamburg
Master Informatik
Hauptseminar im Wintersemester 2016

Zusammenfassung: In dieser Ausarbeitung wird aufgezeigt, welche Möglichkeiten es gibt, um Anfragen im Big-Data-Umfeld schnell zu beantworten. Es werden Grundlagen beleuchtet und Big-Data-Architekturen beschrieben. Die approximative Anfrageverarbeitung stellt eine besondere Form dar, um Anfragen schnell zu beantworten. Die Antworten sind jedoch nicht exakt. Der Fehler der Antworten hingegen ist bekannt. Anschließend werden Ziele für das Grund- und Hauptprojekt sowie für die Master-Arbeit benannt und die dabei möglichen Risiken dargestellt.

Schlüsselwörter: Big Data, NoSQL, Approximative Anfrageverarbeitung, Sampling, Uniform Samples, Stratified Samples, BlinkDB, SnappyData

1 Einleitung

Das gesamte Datenvolumen lag im Jahr 2012 bei etwa 2,8 Zettabyte. Im Jahr 2020 hingegen soll das gesamte Datenvolumen bereits im Bereich von 40 Zettabyte liegen. [5] Nimmt die Datenmenge zu, dauert ihre Auswertung auch länger. Mitunter kann eine Anfrage im Bereich von Minuten, Stunden oder sogar Tagen liegen, wenn die gesamte zugrunde liegende Datenmenge in der Größenordnung von mehreren Terabyte durchsucht wird. [29]

Dauert eine Anfrage im Terabyte-Bereich bereits so lange, wird eine Anfrage auf einen Datensatz im Bereich von Petabyte oder darüber hinaus noch viel länger dauern und die Antwort in der Zwischenzeit wertlos geworden sein. Denn während die Anfrage ausgeführt wird, kann sich die Datenbasis ändern und die Antwort wäre unter Umständen anschließend faktisch falsch. Daher sind Systeme, die kurze Antwortzeiten trotz sehr großer Datenmengen gewährleisten, von großem Interesse.

Mögliche Anwendungsfälle, in denen kurze Antwortzeiten gefordert sind, sind u.a. folgende:

1. *Wertpapierhandel*
   Die Entscheidungen darüber, ob gewisse Wertpapiere ge- oder verkauft werden sollen, müssen im Bereich von Millisekunden getroffen werden. Eine minuten- oder stundenlange Auswertung ist nicht möglich.

2. *Sensoren im IoT-Umfeld*
   Im *Internet of Things*-Umfeld sammeln zahlreiche Sensoren sehr viele Daten, die sehr schnell durchsucht und ausgewertet werden müssen. Angenommen auf einer Ölbohrplattform liefern Sensoren ungewöhnliche Werte, die eine sofortige Reparatur erfordern, muss hierauf schnell reagiert werden, um z.B. eine drohende Explosion zu vermeiden.
3. *Streaming-Dienste*
   Möchten Kunden eines Video-Streaming-Anbieters beispielsweise Filme zu einem bestimmten Genre angezeigt bekommen, so muss die Suche sofort Ergebnisse liefern und nicht erst die gesamte Datenbasis mehrere Minuten lang durchsuchen. Denn das würde sich negativ auf die Akzeptanz der Kunden auswirken und sie würden einen solchen Streaming-Anbieter nicht nutzen.

In dieser Ausarbeitung werden Konzepte vorgestellt, die sehr kurze Antwortzeiten für Anfragen auf sehr großen Datenmengen ermöglichen. Zunächst werden die Begriffe *Big Data* und *NoSQL* erläutert. Anschließend werden *Anforderungen an Big-Data-Systeme* definiert und *Big-Data-Architekturen* vorgestellt, die diese Anforderungen erfüllen können. Daraufhin wird das Konzept der *approximativen Anfrageverarbeitung* vorgestellt und mit *BlinkDB* und *SnappyData* werden zwei exemplarische Umsetzungen für die Anfrageverarbeitung im Big-Data-Umfeld gezeigt. Zuletzt werden die Ziele und der aktuelle Stand des Grundprojekts sowie Ziele für das Hauptprojekt und die Master-Arbeit genannt.

2 Big Data und NoSQL

In diesem Abschnitt werden die Begriffe *Big Data* und *NoSQL* geklärt.

2.1 Big Data

Es ist nicht klar, wann der Begriff *Big Data* entstand und was damit genau gemeint war. Jedoch hat sich die Definition von Gartner aus dem Jahr 2011 [14] als Konsens herausgebildet. So wird Big Data mittels des *3-V-Modells* definiert, welches die drei Dimensionen (1) Volume, (2) Velocity und (3) Variety enthält. [24] Es gibt weitere Modelle wie das 4-V-Modell [3] oder das 5-V-Modell [27]. Jedoch liegt all diesen das 3-V-Modell zugrunde und hat sich als bewährte Definition durchgesetzt.

Mit *Volume* ist das Ansteigen des Datenvolumens gemeint. *Velocity* bezieht sich einerseits auf die Geschwindigkeit, mit der Daten erzeugt und andererseits mit der sie schnell weiterverarbeitet werden. Unter *Variety* ist die Unterschiedlichkeit der Daten gemeint. Diese sind unstrukturiert und unterschiedlich voneinander und stellen für traditionelle Datenbanksysteme eine große Herausforderung dar. [24]

2.2 NoSQL

Relationale Datenbankmanagementsysteme (RDBMS) wurden früher für nahezu alle Speicherzwecke verwendet, selbst wenn das zugrunde liegende Daten-Modell nicht zum relationalen Modell passte, wie beispielsweise der Object-Relational Impedance Mismatch oder das Speichern von Graphen in Tabellen. Um damit umzugehen, werden Mapping-Frameworks und komplexe Algorithmen eingesetzt, was wiederum zu einer erhöhten Komplexität führt. Zudem bieten RDBMS viele Features an, die für einfache Aufgaben wie Logging nicht benötigt werden. [23]

Dadurch, dass sehr große Datenmengen im Tera- und Petabytes-Bereich aufkamen, müssen massive Schreib- und Leseoperationen durchgeführt werden, die auf Clustern mit einer geringen Latenz laufen. Für die Verteilung der Daten sind RDBMS nicht gut geeignet. Die Gründe hierfür sind die vollständige Unterstützung von ACID und das normalisierte Datenmodell. [23]

Hochverfügbarkeit ist ebenso wichtig für ein Datenbanksystem. So muss ein solches System einfach replizierbar sein und mit ausgefallenen Knoten zurechtkommen. Damit die maximale Kapazität einzelner Server nicht überschritten wird, müssen Anfragen verteilt werden. Da RDBMS ihren Fukus tendenziell auf Konsistenz und weniger auf Hochverfügbarkeit legen können sie diese Anforderungen schlecht bewerkstelligen. [23] Das von Brewer aufgestellte CAP-Theorem ist der Grund hierfür. Demnach sind in einem verteilten Datenbanksystem nur zwei der drei Dimensionen gleichzeitig erreichbar: Konsistenz, Hochverfügbarkeit, Partitionstoleranz. [12] Als Pendant zu ACID wurde BASE entwickelt. Hierbei steht die Hochverfügbarkeit im Fokus und nicht die Konsistenz. [32]

Daher entstanden neue Datenbanksysteme, die als NoSQL-Datenbanken tituliert wurden. Hierbei gibt es vier Untergruppen von Systemen: (1) Key Value Stores, Document Stores, Column Family Stores, Graph Databases. [23] NoSQL-Datenbanken nutzen u.a. das Map-Reduce-Verfahren [16] oder Multiversion Concurrency Control [17].

3 Anforderungen an Big-Data-Systeme

Die Workflows vieler Big-Data-Systeme beinhalten kontinuierliches Stream Processing, Online Transaction Processing (OLTP) und Online Analytical Processing (OLAP). Diese Systeme müssen parallel sehr schnelle Streams konsumieren um Warnungen in Echtzeit auszulösen, diese in eine schreib-optimierte Datenbank speichern und analytische Anfragen ausführen um schnell tiefe Einsichten in die Daten zu gewinnen. [11]

Eine etwas ausführlichere Auflistung von Anforderungen an ein Big-Data-System liefern Marz und Warren in [28]. Ein Big-Data-System soll (1) belastbar und fehlertolerant sein, (2) Lesen und Aktualisieren mit geringen Latenzen ermöglichen, (3) skalierbar, (4) allgemeingültig und (5) erweiterbar sein, (6) die Möglichkeit Ad-hoc-Abfragen auszuführen bieten, (7) einen minimalen Wartungsaufwand haben und (8) eine Fehlerbehebung beinhalten.

3.1 Big-Data-Architekturen

Um die Anforderungen im Big-Data-Umfeld zu gewährleisten haben sich drei maßgebliche Architekturen herausgebildet. Diese werden im folgenden beschrieben.

3.1.1 Lambda-Architektur

Der Begriff *Lambda-Architektur* wurde durch Nathan Marz eingebracht für eine generische, skalierbare und fehlertolerante Datenverarbeitungsarchitektur. Diese basiert auf seinen Erfahrungen bezüglich verteilter Datenverarbeitungssysteme, die er während seiner Arbeit bei Backtype und Twitter gesammelt hat. [22] Narz und Warren beschreiben die Lambda-Architektur ausführlich in [28]. Dabei gehen sie genau auf die einzelnen Layer ein und zeigen Möglichkeiten für die Realisierung durch Produkte aus der Praxis.

Abbildung 1(a) zeigt die Lambda-Architektur. Diese besteht aus den folgenden Komponenten:

(1) Eintreffende Daten werden an Batch- und Speed-Layer zur Verarbeitung gegeben. (2) Der Batch-Layer verwaltet den unveränderlichen Master-Datensatz mit Rohdaten, an den Daten nur hinzugefügt werden können und berechnet den Batch-View. (3) Der Serving-Layer indiziert den Batch-View, um Ad-hoc-Anfragen mit geringen Latenzen zu ermöglichen. (4) Alle Abfragen können durch das Zusammenführen von Batch- und Real-Time-View beantwortet werden.

3.1.2 Kappa-Architektur

In der Lambda-Architektur muss Code für zwei den Batch- und Speed-Layer geschrieben und gewartet werden. Diese Layer sind komplex und dies stellt eine Herausforderung dar sowie das Debugging ebenfalls.

Das Problem ließe sich durch ein vor Batch- und Speed-Layer vorgeschaltetes Framework reduzieren, in dem der Code nur ein Mal geschrieben wird. Realistisch ist ein solches Framework jedoch nicht, da es die Vereinigungsmenge beider Layer abbilden würde und die Produkte in der Praxis zu unterschiedlich sind als dass sich eine solche Menge bilden ließe. [26]

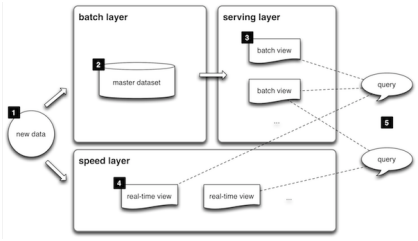
Daher wird von Jay Kreps die *Kappa-Architektur* vorgeschlagen, dessen Kernkomponente ein zentrales, massiv skalierbares Logsystem ist. In dieser werden alle Ereignisse gespeichert und sind unveränderlich. Jedes neue Ereignis wird an das Log-Ende hinzugefügt, woraus eine strikte zeitliche Abfolge resultiert. Verschiedene Anwendungen verarbeiten die Log-Daten, wobei jede Anwendung die Kontrolle über zuletzt gelesene und noch zu lesende Datensätze hat. Um eine hohe Performance zu gewährleisten, ist ein hoher Durchsatz beim Schreiben und Lesen des Logsystems wichtig. Durch diese Architektur werden Inkonsistenzen zwischen Batch- und Speed-Layer vermieden. Abbildung 1(c) zeigt die Kappa-Architektur. [26], [20]

3.1.3 Liquid-Architektur

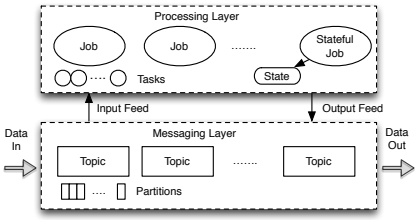
Da die Kappa-Architektur einen höheren Storage benötigt und die Anwendungen auf alte Daten zugreifen, solange das System die Daten neu berechnet, bildete sich die *Liquid-Architektur* heraus. In Abbildung 1(b) ist diese zu sehen.

Diese Architektur besteht aus Messaging- und Processing-Layer. Im Processing-Layer werden Jobs gemäß eines Stateful-Stream-Verarbeitungsmodells für verschiedene Backend-Systeme ausgeführt. Ressource-Isolation, Ergebnisse mit niedrigen Latenzen und eine inkrementelle Datenverarbeitung werden ermöglicht. Der Messaging-Layer speichert Daten mit hoher Verfügbarkeit, die sich bei Bedarf zurücksetzen lassen (Rewindability).

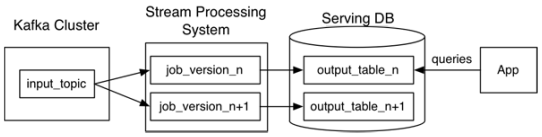
Die Layer kommunizieren über Feeds (Topics in einem Publish-Subscribe-Modell), sodass jedes Backend-System nur die Daten konsumiert, die es benötigt und von anderen Daten keine Kenntnis haben muss. [21], [15]



(a) Lambda-Architektur [22]



(b) Liquid-Architektur [21]



(c) Kappa-Architektur [26]

Abb. 1. Big-Data-Architekturen

3.2 Bewertung der Architekturen

Die im Abschnitt §3 genannten Anforderungen erfüllen die Architekturen nahezu komplett. Sie sind allesamt belastbar und fehlertolerant (1), skalierbar (3), allgemeingültig (4), erweiterbar (5), bieten Ad-Hoc-Abfragen (6) und eine Fehlerbehandlung (8).

Das Lesen und Aktualisieren mit geringen Latenzen (2) ist in der Lambda-Architektur für den Batch-Layer nicht komplett gegeben. Die Daten des Batch-Layers müssen zunächst im Serving-Layer aufbereitet werden, ehe das Lesen mit geringen Latenzen möglich ist. Der Wartungsaufwand (7) ist in der Lambda-Architektur ebenfalls deutlich größer als in der Kappa- bzw. Liquid-Architektur,

da der Code in zwei unterschiedlichen komplexen Systemen angepasst werden muss, sobald Änderungen notwendig sind.

Wartungsaufwand kann auch durch die Komplexität des Systems resultieren. Komplexität wurde nicht als einzelne Anforderung genannt, ist jedoch für jedes System wichtig. Je weniger komplex ein System ist, desto einfacher ist es zu verstehen und somit auch zu warten. Bei allen drei Architekturen ist Gefahr da, dass das System zu komplex wird. Obwohl die Liquid-Architektur die wenigsten Layer hat, kann sie komplexer als die Lambda- oder Kappa-Architektur sein, denn einzelne Topics können neben Rohdaten auch durch Jobs (mehrfach) aufbereitete Daten enthalten. Ebenso lassen sich in der Lambda- und Kappa-Architektur Fälle finden, die das System zu komplex werden lassen. Diese Problematik sollte bereits beim Entwurf eines Systems bedacht und durch Konventionen und Dokumentation verringert werden.

4 Approximative Anfrageverarbeitung

Die im Abschnitt §3.1 beschriebenen Architekturen beinhalten Methoden, um schnelle Antworten auf Anfragen zu ermöglichen. Bei der Lambda-Architektur ist hierfür beispielsweise der Speed-Layer verantwortlich. Dieser basiert jedoch nur auf den aktuellsten Daten und stellt nicht die einzige Herangehensweise dar, wie sich schnelle Antworten auf Anfragen realisieren lassen. Einen etwas anderen Ansatz bietet die approximative Anfrageverarbeitung, die in diesem Abschnitt beschrieben wird.

Wenn sehr große Datenmengen anfallen, brauchen auch NoSQL-Systeme sehr lange, um auf Anfragen zu antworten. Beispielsweise braucht eine Anfrage auf einen 7,5 Terabyte großen Datensatz mit Hive/Hadoop auf einem Cluster von 100 Knoten ca. 1,5 Stunden. Eine solch lange Laufzeit einer Anfrage ist bei weitem nicht mehr als Ad-hoc-Anfrage zu bezeichnen. [9]

Hierfür bietet sich die approximative Anfrageverarbeitung an, da sie es ermöglicht Anfragen sehr schnell zu beantworten. Die Antworten sind nicht genau, aber der Fehler ist bekannt und die Antwortzeiten sind kurz. So ist für viele Anwendungsfälle eine Antwortgenauigkeit von 99% und eine Antwortzeit von wenigen Sekunden besser geeignet als eine 100% genaue Antwort, die länger als eine Stunde dauert. [9]

4.1 Einsatzgebiete approximativer Anfrageverarbeitung

Die approximative Anfrageverarbeitung bietet in den folgenden Szenarien einen Vorteil: [31], [30]

1. *Interaktive Anfragen*
   Die Fülle an Datensätzen hat auf Daten beruhende Entscheidungen sehr prominent gemacht. Jedoch bilden zahlreiche vorhandene Datenverarbeitungs-Tools den Flaschenhals in Entscheidungs-Systemen, was auf den starken Zuwachs der Datenmengen beruht. Durch die approximative Anfrageverarbeitung sind weiterhin interaktive Anfragen mit kurzen Antwortzeiten möglich.

2. *Perfekte Entscheidungen trotz nicht perfekter Antworten*
   Bei A/B-Tests, Testen von Hypothesen Root-Cause-Analysen, explorativen Analysen, Feature Selection und bei der Visualisierung von Daten im Big-Data-Umfeld sind genaue Antworten nicht notwendig, da auch eine Annäherung an die exakte Antwort bereits eine ausreichende Sicherheit gibt. Daher ist es auch nicht erforderlich eine Anfrage auf dem gesamten Datensatz durchzuführen.
3. *Unvollständige Datenbasis*
   Datensätze sind meist bereits mit einem Fehler behaftet. Die Gründe hierfür können u.a. folgende sein: Menschliche Fehler, fehlende Werte, Rauschen in den Daten, Sensorfehler, fehlerhafte Datenextraktionen, Datenkonvertierungen oder Rundungsfehler. Somit sind selbst Antworten auf Basis der gesamten Datenmenge nicht exakt, sondern approximativ.
4. *Predictive Analytics*
   Soll durch Predictive Analytics eine Vorhersage getroffen werden, werden Maschine-Learning-Algorithmen eingesetzt, um dies zu bewerkstelligen. Diese werden mittels eines Trainingsdatensatzes trainiert. Wird der Trainingsdatensatz reduziert, so verschlechtert sich der Fehler in der Praxis kaum. Der gesamte Trainingsdatensatz ist also nicht erforderlich, um einen Machine-Learning-Algorithmus zu trainieren.

4.2 Samples

Um bei Entscheidungsfindungen zu helfen, werden in Anwendungen beispielsweise Online Analytical Processing (OLAP) oder Data Mining eingesetzt. Dabei werden Aggregationsanfragen auf sehr großen Datenmengen ausgeführt, die häufig teuer und ressourcenintensiv sind. Um die Skalierbarkeit dieser Anwendungen zu ermöglichen, können ungefähre aber schnelle Antworten eine Möglichkeit sein. [13]

Die Grundidee bei Samples ist, die Auswertung nur auf einem Teil der Daten auszuführen. Samples werden vorberechnet und anschließend für Anfragen genutzt. Die Ausführungsgeschwindigkeit der Anfragen ist geringer, da diese nicht auf der gesamten Datenmenge ausgeführt werden. Jedoch sind die Antworten ungenau. Der in ihnen enthaltene Fehler ist bekannt. [6]

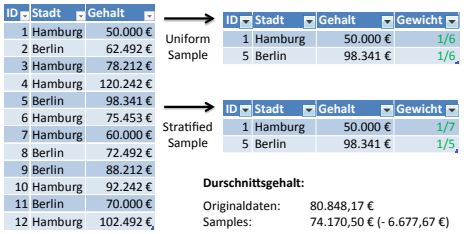


Abb. 2. Uniform und Stratified Samples [6]

4.2.1 Uniform und Stratified Samples

Bei den sog. *Uniform Samples* werden zufällige Einträge aus der Originaltabelle anhand einer Sampling-Rate entnommen. Die Werte der Einträge spielen keine Rolle. Bei *Stratified Samples* werden die Einträge nicht zufällig ausgewählt.

Ziel dabei ist es, dass keine Teilmengen verloren gehen. Somit werden kleine Teilmengen überrepräsentiert und gewichtet. Hierdurch wird sichergestellt, dass Anfragen über alle Teilmengen beantwortet werden können. [9]

Abbildung 2 zeigt ein Beispiel für Uniform und Stratified Samples. Diese werden in Bezug auf die Spalte Stadt erstellt. Bei den Uniform Samples hat jeder Eintrag dasselbe Gewicht, während bei den Stratified Samples die Einträge nach der Häufigkeit ihres Vorkommens in der Originaltabelle gewichtet sind.

4.2.2 Sample Management

Samples können dadurch erstellt werden, dass zukünftige Anfragen im Vergleich zu vergangenen ähnlich sind. Es gibt unterschiedliche Modelle, um die Ähnlichkeit zukünftiger Anfragen zu bestimmen: [9]

1. *Vorhersagbare Anfragen*
   Zukünftige Anfragen sind identisch zu den vergangenen. Ändert sich die Anfrage, funktioniert dieses Vorhersagemodell nicht mehr.
2. *Vorhersagbare Anfrage-Prädikate*
   Über die Zeit gesehen bleibt die Häufigkeit von Gruppierungs- und Filterprädikaten identisch. Haben beispielsweise 5% vergangener Anfragen das Prädikat WHERE city='Hamburg' und kein weiteres, so werden 5% zukünftiger Anfragen dasselbe Prädikat haben. Das Modell ist flexibler als Modell 1, jedoch hat es genaue Werte in den Prädikaten. Ändern sich diese Werte, greift dieses Modell ebenfalls nicht mehr.
3. *Vorhersagbare Spalten einer Anfrage*
   Über die Zeit gesehen bleibt die Häufigkeit der Spalten einer Anfrage identisch, die exakten Werte in den Gruppierungs- und Filterprädikaten sind jedoch nicht vorhersagbar. Wenn beispielsweise 5% vergangener Anfragen nach der Spalte city gruppiert oder gefiltert wurden, werden 5% zukünftiger Anfragen nach derselben Spalte gruppiert oder gefiltert. Dadurch, dass nur die Mengen der Spalten gleich bleiben, die Werte der Prädikate jedoch nicht beachtet werden, bietet dieses Modell eine sehr hohe Flexibilität.
4. *Unvorhersagbare Anfragen*
   Wenn die Anfragen nicht vorhersagbar sind und Optimierungen nur auf einer Anfrage erfolgen, ist das Modell nicht geeignet für das Erstellen von Samples.

4.3 Fehlerratenbestimmung

Die Antwort einer auf Samples ausgeführten Anfrage ist ungenau. Eine genaue Antwort ist nur möglich, wenn diese auf der gesamten Datenbasis ausgeführt wird. Und auch nur dann, wenn in der Datenbasis keine Messfehler vorhanden sind. Um die Antwort-Qualität zu erkennen, ist die Kenntnis des in ihr enthaltenen Fehlers sehr wertvoll.

Um dieses Problem zu lösen, gibt es statistische Ansätze, die den Fehler einer Antwort schätzen. Je besser die statistischen Methoden dieser Ansätze sind, desto exakter ist der geschätzte Fehler der Antwort.

So wurde *Bootstrap* 1979 von Efron vorgestellt [19] und 1993 von Efron und Tibshirani weiterentwickelt [18]. Im Jahr 2013 haben Kleiner et al. gezeigt, dass dieser Ansatz unzuverlässig ist [25]. 2014 wurde von Zeng et al. der Ansatz *Analytical Bootstrap* entwickelt, der Bootstrap erweitert [33].

5 Exemplarische Umsetzungen der Prinzipien

Im Folgenden werden zwei Umsetzungen vorgestellt, die die Prinzipien der Anfrageverarbeitung im Big-Data-Umfeld implementieren.

5.1 BlinkDB

BlinkDB ist ein massiv paralleles Framework, mit dem approximative Anfragen auf sehr großen Datenmengen durchgeführt werden können. BlinkDB kann dabei mit derselben Art von Anfragen umgehen, wie der Hive/HDFS-Stack. Die Antworten werden in einer sehr kurzen Zeit geliefert und haben eine garantierte Fehlerquote. Mit BlinkDB lassen sich Daten in der Größenordnung von Petabytes verarbeiten und BlinkDB lässt sich auf mehrere Tausend Knoten skalieren. [7]

Die Anfragen in BlinkDB sind SQL-artig und werden auf vorberechneten Samples ausgeführt. In der Anfrage lässt sich definieren, in welcher Zeit die Antwort geliefert werden soll oder welche Fehlerquote diese haben darf. [9]

BlinkDB verwendet Stratified Samples, die über die Zeit gesehen angelegt werden. Hierfür ist eine dynamische Auswahlstrategie verantwortlich, die eine passend große Datenmenge auf der Basis von Zeit- bzw. Genauigkeitsanforderungen der Anfrage anlegt. [7]

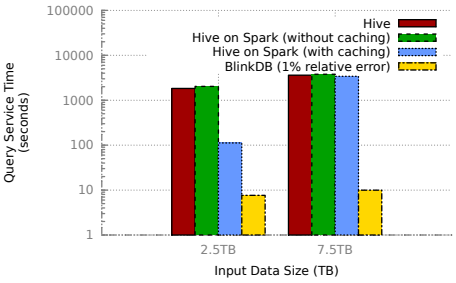


Abb. 3. BlinkDB / kein Sampling [9]

Für die Auswahl der Samples wird das Modell der *vorhersagbaren Spalten einer Anfrage* verwendet wie in Abschnitt §4.2.2 beschrieben. Die Gruppe der vorhandenen Spalte einer Anfrage wird auch *Query Column Set* oder kurz *QCS* genannt. Die Sammlung der Stratified Samples wird auf der Basis der QCSs und ihrer historischen Häufigkeit unterhalb einer Speichergrenze angelegt, die der Benutzer konfiguriert. Die Samples werden so ausgewählt, dass Anfragen mit identischen QCSs effizient und Anfragen mit ähnlichen QCSs gut beantwortet werden können. Praxisversuche haben gezeigt, dass dieses Modell gut funktioniert und für zukünftige Anfragen gut geeignet ist. [9]

BlinkDB verwendet den von Kleiner et. al [25] entwickelten Ansatz zur Schätzung des Fehlers in der Antwort. Da dieser jedoch nicht schnell genug ist für ein System

der approximativen Anfrageverarbeitung, wurde dieser von BlinkDB erweitert. [8]

Wie in Abb. 3 zu sehen, liefert BlinkDB Antworten in einer sehr kurzen Zeit. So wurden Anfragen in Produktivumgebungen von Facebook und Conviva beispielsweise statt in 0,5 Stunden in unter 10 Sekunden bzw. statt in 1,5 Stunden in 10 Sekunden beantwortet.

Das Projekt BlinkDB ist nicht mehr aktuell. Der letzte Commit ist am 06.02.2014 erfolgt, also vor mehr als drei Jahren. Als Basis wird Apache Spark 0.9 genannt. [1] Aktuell ist derzeit die Version 2.1.0. [2] Somit ist selbst ein Einsatz für Untersuchungszwecke schwierig zu realisieren.

5.2 SnappyData

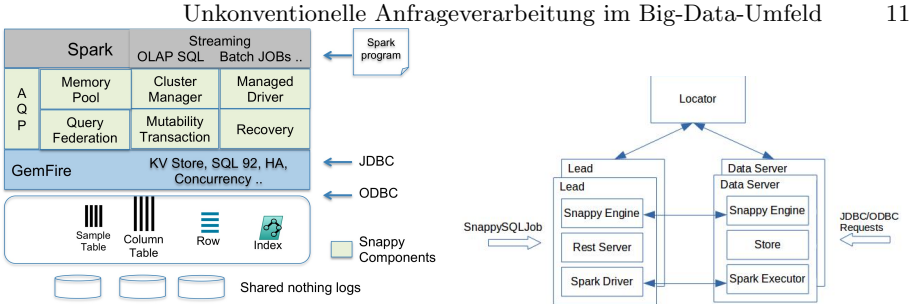
In der Praxis ist es häufig so, dass mehrere Produkte miteinander kombiniert werden, um beispielsweise die Lambda-Architektur aufzubauen (siehe Abschnitt §3.1.1). Eine solche Architektur hat jedoch Nachteile:

1. *Hohe Komplexität*
   Entwickler müssen mehrere APIs, Datenmodelle, Konfigurationen und sich ändernde Optionen für mehrere Produkte beherrschen. Das Betreiben einer solchen Architektur ist operativ nicht einfach. Und das Debugging ist schwierig, da mehrere Logs unterschiedlicher Produkte ausgewertet werden müssen, um den Fehler zu finden.
2. *Niedrige Performance*
   Die Daten werden zwischen mehreren non-colocated Clustern ausgetauscht, was mehrere Netzwerk-Hops und multiple Kopien der Daten bedeutet. Außerdem müssen die Daten ggf. transformiert werden, wenn die Datenmodelle untereinander nicht kompatibel sind.
3. *Verschwendete Ressourcen*
   Dadurch, dass die Daten dupliziert werden, wird mehr Netzwerk-Bandbreite, mehr CPU und mehr Speicher benötigt.
4. *Interaktive Analysen ungenügend*
   Anfragen, die über verteilte Cluster laufen, dauern zu lange (über zehn Sekunden bis hin zu mehreren Minuten).

SnappyData möchte diese Nachteile beseitigen und vereint die Lambda-Architektur in einem Produkt. [11] SnappyData verwendet dabei *Apache Spark*[1] und *GemFire*[2] als Kernkomponenten und bildet damit ein *Single Unified HA Cluster*, das *OLTP*, *OLAP* und *Streaming für Real-Time-Analysen* ermöglicht. [10]

Abbildung 4(a) zeigt einen Überblick über die SnappyData-Komponenten. So ist unter anderem ein auf BlinkDB aufbauendes Modul für die approximative Anfrageverarbeitung enthalten, um so schnelle interaktive Anfragen zu ermöglichen. Hierfür wird eine Sample-Tabelle vorgehalten, um dies zu gewährleisten.

[1] http://spark.apache.org/
[2] https://pivotal.io/pivotal-gemfire



(a) SnappyData Überblick [11] (b) SnappyData Architektur [11]

In Abbildung 4(b) ist die Verteilung der Komponenten von SnappyData zu sehen. Der *Locator* bietet einen Discovery Service an mit dessen Hilfe neue Knoten einander kennenlernen können. Der *Lead* agiert analog zum *Spark Driver* in Apache Spark und stellt einen SparkContext zur Verfügung. Außerdem führt er SQL-Abfragen aus. Der *Data Server* speichert die Daten. Ebenfalls führt dieser Abfragen entweder direkt aus oder leitet diese zu dem Lead weiter, der diese weiterverarbeitet.

6 Master-Arbeit und Projekte

Das Ziel des Master-Themas ist die experimentelle Untersuchung von ausgewählten Anwendungsszenarien der approximativen Anfrageverarbeitung bezüglich des Kompromisses der Qualitätsziele verschiedener Szenarien. Hierfür sollen im Grund- und Hauptprojekt die Grundlagen in Form einer Experimentierplattform geschaffen und in der Master-Arbeit die theoretischen Grundlagen untersucht werden.

6.1 Grundprojekt

6.1.1 Ziele

Im Grundprojekt wird eine Experimentierplattform mit SnappyData aufgebaut, um Performance-Analysen durchführen zu können. Die Plattform wird auf dem Big-Data-Cluster in der HAW im Raum 10.88 aufgebaut. Wenn diese Experimentierplattform fertiggestellt ist, soll sie sich auch auf bei einem Cloud-Computing-Anbieter wie beispielsweise Amazon Web Services[3] deployen lassen.

Die Komponenten *Locator*, *Lead* und *Data Server* sollen dabei jeweils in Docker-Containern[4] laufen. Dies stellt sicher, dass ein Deployment bei Cloud-Computing-Anbietern einfacher möglich ist.

Die Performance-Analysen sollen durch geeignete Daten erfolgen. Hierfür könnten beispielsweise die Daten der „NYC Taxi & Limousine Commision“[5] herangezogen werden. Denkbar sind jedoch auch weitere Benchmark-Daten wie sie beispielsweise „TPC“[6] bereitstellt.

[3] https://aws.amazon.com
[4] https://www.docker.com
[5] http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
[6] http://www.tpc.org/

Das Hauptmerkmal der Untersuchungen soll dabei auf dem Modul für die approximative Anfrageverarbeitung liegen. Die Ergebnisse der Performance-Analysen sollen miteinander verglichen werden. Es soll geprüft werden inwieweit sich die Ergebnisse zwischen der Ausführung auf der gesamten Datenbasis und der Ausführung auf Samples unterscheiden im Hinblick auf die Genauigkeit und Ausführungsgeschwindigkeit.

6.1.2 Risiken

Das Grundprojekt ist teilweise mit Risiken behaftet. So könnten unter anderem folgende Risiken eintreten:

1. *Docker-Container und Netzwerk*
   Durch das Zusammenspiel aus den Docker-Containern, die die SnappyData-Komponenten enthalten und dem Netzwerk könnte eine kleine Herausforderung darstellen. Denn Docker-Container haben dynamisch zugewiesene IP-Adressen. Das bedeutet, dass keine feste IP-Adressen für die Konfiguration von SnappyData-Komponenten verwendet werden können. Die Docker-Schicht bringt also zunächst eine weitere Komplexitätsebene mit sich.
2. *Keine weite Verbreitung von SnappyData*
   SnappyData hat noch keine allzu große Verbreitung. Sollte es bei der Installation oder beim Betrieb zu Problemen kommen, könnte die Fehlersuche länger dauern als beispielsweise bei der Inbetriebnahme oder beim Betrieb von Apache Spark. Begründet ist dies damit, dass Spark eine große Verbreitung hat und es zahlreiche Foren-Einträge gibt, die sich mit Installations- und Betriebsproblemen auseinandersetzen. Jedoch verfügt SnappyData über eine ausführliche Dokumentation und bietet über mehrere Kanäle wie Slack, Github, Stackoverflow Support[7].

6.1.3 Aktueller Stand

Die SnappyData-Komponenten befinden sich derzeit in drei einzelnen Docker-Containern, die auf einem Rechner gestartet werden. Derzeit müssen noch die IP-Adressen der Docker-Container jedes Mal manuell in der SnappyData-Konfiguration eingetragen werden, was zunächst nur dazu dient, den Durchstich zu erzielen. Dieser ist jedoch noch nicht erreicht, da das SnappyData-Cluster noch nicht verteilt startet. Es lässt sich nur innerhalb eines Docker-Containers starten.

6.2 Hauptprojekt

6.2.1 Ziele

Im Hauptprojekt soll die Experimentierplattform genutzt werden, um weitere Anwendungsszenarien durchzuspielen und die Eignung der approximativen Anfrageverarbeitung weiter zu überprüfen. Es soll mindestens ein weiterer Datensatz untersucht werden. Idealerweise zeigen die Untersuchungen, dass die Abweichungen der approximativen Anfrageverarbeitung zu stark von den tatsächlichen

[7] http://www.snappydata.io/community

Ergebnissen abweichen. Das würde aufzeigen, dass die approximative Anfrageverarbeitung nicht für jegliche Szenarien geeignet ist.

Die Experimentierplattform soll außerdem bei einem Cloud-Computing-Anbieter deployt werden, um so die Plattform noch weiter skalieren zu können im Gegensatz zum HAW-Cluster. Hierbei soll untersucht werden, welche Datenmengen noch komplett durchsucht werden können ohne auf die approximative Anfrageverarbeitung zurückgreifen zu müssen.

6.2.2 Risiken

Das Hauptprojekt beinhaltet weitere Risiken. Dies könnten unter anderem folgende sein:

1. *Passender Datensatz*
   Um zu zeigen, dass die approximative Anfrageverarbeitung nicht für alle Szenarien geeignet ist, muss ein passender Datensatz gefunden werden, der dies zeigt. Das Finden eines solchen Datensatzes, der frei zugänglich ist, könnte eine Hürde darstellen.
2. *Kosten Cloud-Computing-Anbieter*
   Die Kosten der Cloud-Computing-Anbieter könnten ein Risiko darstellen, da keine unbegrenzten finanziellen Mittel zur Verfügung stehen. Ggf. hilft hierbei das Programm „AWS im Bildungswesen“, das die HAW nutzen darf.
3. *Deployment Cloud-Computing-Anbieter*
   Das Deployment bei dem Cloud-Computing-Anbieter setzt eine Einarbeitungsphase voraus, in der die Grundlagen für das Deployment geschaffen werden.

6.3 Master-Arbeit

6.3.1 Ziele

In der Master-Arbeit sollen die durchgeführten Experimente des Grund- und Hauptprojekts auf theoretischer Ebene untersucht und bewertet werden. Möglicherweise lassen sich Verbesserungsvorschläge erarbeiten, die einen besseren Kompromiss der Qualitätsziele der Szenarien ermöglichen. Die Verbesserungsvorschläge ließen sich sogar ggf. in die Experimentierplattform implementieren und nochmals überprüfen, denn der Sourcecode von SnappyData ist Open Source. [4]

6.3.2 Risiken

Als Risiken für die Master-Arbeit sind mindestens folgende zu benennen:

1. *Statistische Modelle*
   Die der approximativen Anfrageverarbeitung zugrunde liegenden statistischen Modelle müssen verstanden werden, um eine Bewertung dieser vorzunehmen. Hierfür sind mindestens Grundlagen der Statistik erforderlich, die erarbeitet werden müssten.

2. *Modifizieren des Sourcecodes*
   Die Modifikation des Sourcecodes von SnappyData könnte eine Hürde darstellen, da hierfür die Kenntnis der einzelnen Komponenten und ihrer Abhängigkeiten untereinander erforderlich ist, um so gezielt die entsprechende Stellen anpassen zu können.

Literatur

1. Blinkdb: Sub-second approximate queries on very large data. https://github.com/sameeragarwal/blinkdb, letzter Zugriff: 05.03.2017.
2. Blinkdb: Sub-second approximate queries on very large data. http://spark.apache.org/, letzter Zugriff: 05.03.2017.
3. The four v's of big data. http://www.ibmbigdatahub.com/infographic/four-vs-big-data, letzter Zugriff: 21.02.2017.
4. Snappydata, the spark database. stream - transact - analyze - predict all in one cluster. http://www.snappydata.io/, letzter Zugriff: 17.02.2017.
5. 2.800.000.000.000.000.000.000 byte: Das digitale universum schwillt an, Dezember 2012. http://www.spiegel.de/netzwelt/web/das-internet-der-dinge-erzeugt-2-8-zettabyte-daten-a-872280.html, letzter Zugriff: 17.02.2017.
6. Agarwal, S. BlinkDB: Qureying Petabytes of Data in Seconds using Sampling, 2014.
7. Agarwal, S., Iyer, A. P., Panda, A., Madden, S., Mozafari, B., and Stoica, I. Blink and it's done: interactive queries on very large data. *Proceedings of the VLDB Endowment 5*, 12 (2012), 1902–1905.
8. Agarwal, S., Milner, H., and Kleiner, A. Knowing when you're wrong: building fast and reliable approximate query processing systems. *Proceedings of the 2014 ACM SIGMOD international conference on Management of data* (2014), 481–492.
9. Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., and Stoica, I. BlinkDB: queries with bounded errors and bounded response times on very large data. *Proceedings of the 8th ACM European Conference on Computer Systems - EuroSys '13* (2013), 29.
10. Barzan Mozafari, Jags Ramnarayan, S. M. Y. M. S. C. H. B. K. B. Snappydata: A unified cluster for streaming, transactions and interactice analytics, 2017. http://cidrdb.org/cidr2017/slides/p28-mozafari-cidr17-slides.pdf, letzter Zugriff: 17.02.2017.
11. Barzan Mozafari, Jags Ramnarayan, S. M. Y. M. S. C. H. B. K. B. Snappydata: A unified cluster for streaming, transactions and interactive analytics. *CIDR 2017* (2017). http://cidrdb.org/cidr2017/papers/p28-mozafari-cidr17.pdf, letzter Zugriff: 17.02.2017.
12. Brewer, E. Eric brewer's cap-theorem keynote, Juli 2000. http://www.cs.berkeley.edu/ brewer/cs262b-2004/PODC-keynote.pdf, letzter Zugriff: 21.02.2017.
13. Chaudhuri, S., Das, G., and Narasayya, V. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems 32*, 2 (2007), 9–es.
14. Christy Pettey, L. G. Gartner says solving 'big data' challenge involves more than just managing volumes of data, 2011. http://www.gartner.com/newsroom/id/1731916, letzter Zugriff: 21.02.2017.

15. Colyer, A. Liquid: Unifying nearline and offline big data integration. https://blog.acolyer.org/2015/02/04/liquid-unifying-nearline-and-offline-big-data-integration/, letzter Zugriff: 21.02.2017.
16. Dean, J., and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Commun. ACM 51*, 1 (2008), 107–113. http://doi.acm.org/10.1145/1327452.1327492, letzter Zugriff: 21.02.2017.
17. Edlich, S., Friedland, A., Hampe, J., and Brauer, B. *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. Hanser Fachbuchverlag, 2010.
18. Efron, B. D. o. S. S. U., and Tibshirani, Robert J. (Department of Preventative Medicine and Biostatistics and Department of Statistics, U. o. T. An Introduction to the Bootstrap. *Chapman & Hall* (1993).
19. Efron, B. S. U. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics 7* (1979), 1–26.
20. Fasel, D. Big Data für Smart Cities. *Informatik-Spektrum 40*, 1 (2017), 14–24. http://link.springer.com/10.1007/s00287-016-1001-6, letzter Zugriff: 21.02.2017.
21. Fernandez, R. C., Pietzuch, P., Kreps, J., Narkhede, N., Rao, J., Koshy, J., Lin, D., Riccomini, C., and Wang, G. Liquid: Unifying Nearline and Offline Big Data Integration. *Cidr* (2015).
22. Hausenblas, M., and Bijnens, N. Lambda architecture, 2016. http://lambda-architecture.net/, letzter Zugriff: 20.02.2017.
23. Hecht, R., and Jablonski, S. NoSQL evaluation: A use case oriented survey. *Proceedings - 2011 International Conference on Cloud and Service Computing, CSC 2011* (2011), 336–341.
24. Klein, D., Tran-Gia, P., and Hartmann, M. Big data. *Informatik-Spektrum 36*, 3 (2013), 319–323.
25. Kleiner, A., Talwalkar, a., and Agarwal, S. A general bootstrap performance diagnostic. *Proceedings of the 19th ...* (2013), 419–427.
26. Kreps, J. Questioning the lambda architecture, 2014. https://www.oreilly.com/ideas/questioning-the-lambda-architecture, letzter Zugriff: 20.02.2017.
27. Marr, B. Why only one of the 5 vs of big data really matters, 2015. http://www.ibmbigdatahub.com/blog/why-only-one-5-vs-big-data-really-matters, letzter Zugriff: 21.02.2017.
28. Marz, N., and Warren, J. *Big Data: Entwicklung und Programmierung von Systemen für große Datenmengen und Einsatz der Lambda-Architektur*. mitp Professional. mitp-Verlag, 2016.
29. Mozafari, B. Blinkdb: A massively parallel query engine for big data, August 2013. Verfügbar unter: `http://istc-bigdata.org/index.php/blinkdb-a-massively-parallel-query-engine-for-big-data/`, letzter Zugriff: 11.06.2015.
30. Mozafari, B. When should approximate query processing be used?, April 2015. http://www.snappydata.io/blog/when-should-approximate-query-processing-be-used, letzter Zugriff: 09.01.2017.
31. Mozafari, B., and Niu, N. A Handbook for Building an Approximate Query Engine. *IEEE Data Engineering Bulletin 38*, 3 (2015), 3–29.
32. Pritchett, D. Base: an Acid Alternative. *Queue 6*, 3 (2008), 48–55.
33. Zeng, K., Gao, S., Mozafari, B., and Zaniolo, C. The analytical bootstrap. *Sigmod* (2014), 277–288.