Data Mart (Star Schema) Offload nach Hadoop

Carsten Herbe
Metafinanz-Informationssysteme GmbH
München

Schlüsselworte
Data Mart, Hadoop, HDFS, Hive, Impala, Parquet, Kompression, Snappy, Star Schema, Performance

Einleitung

Heutzutage dienen viele Data Marts als Datenbasis für Standardberichte, Ad-hoc-Analysen und zunehmend auch komplexeren Analysen wie z.B. Data Mining. Dabei sind spielen neben aktuellen Daten auch Historien eine wichtige Rolle. Gerade aus Compliance oder Governance Gründen sind Datenhistorien der letzten 10 Jahre keine Seltenheit.
Oft sind die aktuellen System-Ressourcen bis zum Anschlag ausgelastet und Hardware-Erweiterungen sind aufwendig bzw. ziehen auch erhebliche Kosten nach sich (Lizenzen).
Auf der anderen Seite findet Hadoop immer mehr Einzug in die Rechenzentren. Und damit hat man eine Möglichkeit, bestimmte (historische) Daten von einem RDBMS in einen Hadoop Cluster zu verlagern.
In diesem Dokument wird beschrieben, welche Schritte notwendig sind, welche technologischen Komponenten benötigt werden und was die Auswirkungen auf die Performance sind.

Exkurs Hadoop

Hadoop besteht im Kern aus zwei Komponenten:

- dem verteilten Filesystem HDFS, welches auf einem Cluster aus Standard-Servern liegt und
- dem MapReduce-Framework zur parallelen Verarbeitung dieser Daten.

Rund um diesen Kern ist ein ganzes Ökosystem von Tools entstanden. Manche von ihnen bieten auch die Möglichkeit, Daten mit SQL auszuwerten. Zwei von ihnen, Hive und Impala – werden später noch vorgestellt.
Trotzdem bleibt HDFS ein Filesystem in dem beliebige Daten in beliebigen Formaten gespeichert werden können. Im Gegensatz zu relationalen Datenbanken (Schema-on-Write) wird die Struktur erst beim Zugriff definiert (Schema-on-Read). Weiter können einmal geschriebene Daten in HDFS nicht mehr geändert werden.
Die Stärken von Hadoop sind die kostengünstige Speicherung beliebig strukturierter und nicht strukturierter Daten sowie die parallele Verarbeitung von riesigen Datenmengen. Hadoop Cluster skalieren linear in Bezug auf Speicherkapazität und Performance. 10% mehr Knoten bringen also 10% mehr Speicherkapazität und auch 10% mehr Performance.

Architektur eines Data Mart Offloads

Data Warehouse Architekturen ähneln im Prinzip den Bereichen Quelle, Staging, Core und Data Marts (s. Abb. 1). Aus denen in der Einleitung beschriebenen Gründen, kopieren wir die Daten eines Data Marts aus dem RDBMS in den Hadoop Cluster. Die bestehenden ETL-Prozesse zur Befüllung des Data Marts in der Datenbank bleiben unverändert. Nach der initialen Befüllung in Hadoop könnte man – je nach Anwendungsszenario – z.B. alle Daten die älter als z.B. das laufende Jahr sind, aus der Datenbank löschen und dort nur Daten des aktuellen Jahres vorhalten. Der Data Mart in der Datenbank wird kleiner und die Backup-Dauer wird ggfs. reduziert.

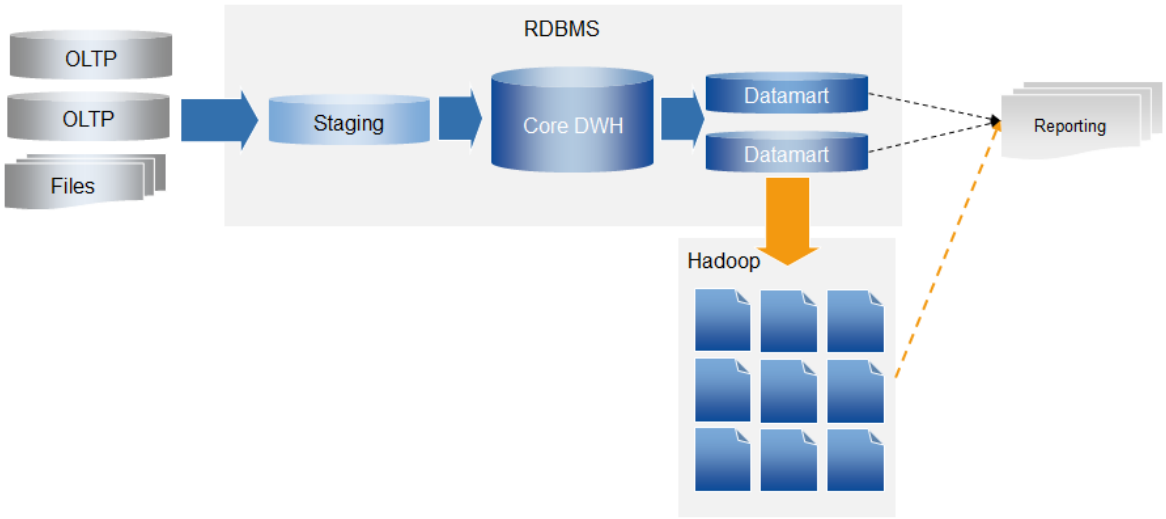

Abb. 1: Data Warehouse Architektur mit Data Mart Offload nach Hadoop

Sollen historische Daten ausgewertet werden, stehen diese in Hadoop zur Verfügung. Was für eine Anbindung des BI-Tools an Hadoop zu beachten ist, wird später noch beschrieben.

Einen Überblick über die unterschiedlichen Schichten des Data Marts in Hadoop gibt Abb. 2

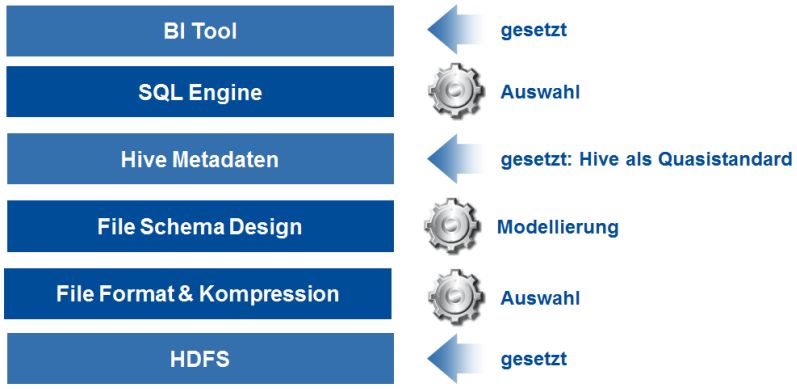

Abb. 2: Schichtenmodell

Einige Schichten sind fest gesetzt, so das HDFS Filesystem, die Verwaltung der relationalen Metadaten mit Hive und das BI-Tool ist natürlich die schon bestehende Lösung aus der relationalen Welt. Beim physikalischen und logischen Schema sowie bei der SQL Engine gibt es durchaus verschieden Möglichkeiten, auf die wir später noch eingehen werden.

Die folgenden Schritte sind für einen kompletten Data Mart Offload notwendig:

1. Aufbau Hadoop Infrastruktur
2. Logisches und physikalisches „Schema-Design“ in HDFS
3. Implementierung des Offload-Prozesses
4. Anbindung BI Tool

Diese Schritte werden im Folgenden nun beschrieben, mit Schwerpunkt auf die Schritte 2 und 3.

Hadoop Infrastruktur

Den Aufbau einer Hadoop Infrastruktur hier komplett zu beschreiben, würde den Rahmen sprengen. Prinzipiell sind die folgenden vier Punkte in Abhängigkeit der geplanten Verwendung des Hadoop Clusters zu klären

1. Hardware: Sizing und Anzahl der Nodes mit Fokus auf Speicherkapazität vs Rechenleistung.
2. Auswahl der benötigten Tools aus dem Hadoop Ökosystem
3. Auswahl einer Hadoop Distribution (oder Appliance)
4. Aufbau Betrieb: Team & Prozesse

Dabei haben die Punkte 1-3 gegenseitige Abhängigkeiten. Bestimmte Tools benötigen z.B. mehr Arbeitsspeicher pro Knoten und nicht jeder Distributor bringt Support für jedes Tool mit.

Man sollte sich hier die Zeit nehmen, eine vernünftige Strategie zu entwickeln.

Schema Design

Wie im Schichtenmodell (s. Abb. 2) schon angedeutet, muss man sich für ein physikalisches Design (File-Format und Kompressions-Codec) und ein logisches Design entscheiden.

Die Tabelle in Abb. 3 gibt einen Überblick über Kompressions-Codecs, die typischerweise in Hadoop verwendet werden.

Codes	Algorithmus	Fokus
BZIP2	Burrows-Wheeler	Ratio
ZLIB	DEFLATE	Ratio
GZIP	DEFLATE	Ratio
LZO	Variante von LZ77	Speed
Snappy	LZ77	Speed
LZ4	Variante von LZ77	Speed

Abb. 3: Kompressions-Codecs

Einige Codecs (BZIP2, ZLIB, GZIP) legen mehr Fokus auf eine hohe Komprimierung. Andere Codecs legen mehr Fokus auf Geschwindigkeit (LZO, LZ4, Snappy). Welches das geeignetste Codes ist, hängt also von der Verwendung des Data Marts ab. Liegt der Fokus mehr auf günstiger Archivierung oder performanter Auswertung.

Bei der Wahl eines File-Formats gibt es zwei unterschiedliche Arten. Die zeilenorientierten Formate speichern die Daten Datensatz für Datensatz.

- Textformat (CSV/TSV)

- Sequence Files
- Avro Files

CSV Dateien sind vielleicht praktisch zum Testen, da man den Inhalt einfach überprüfen kann, aber recht ineffektiv in Bezug auf Speicherbedarf und damit natürlich auf I/O Performance. Komprimierung ist auch nicht so einfach. Hadoop ist nämlich darauf ausgelegt, große Dateien effektiv zu verarbeiten. Dies geschieht dadurch, dass die Blöcke (typischerweise 128 MB pro Block) parallel auf verschiedenen Knoten (bzw. Platten und CPUs) verarbeitet werden. Dafür muss Hadoop natürlich wissen, wo ein Datensatz anfängt und aufhört. Von den vorgestellten Kompressions-Codecs bringt dies nur BZIP2 und mit Einschränkungen LZO mit.

Binäre Formate wie Sequence- oder Avro-Files sind hier im Handling deutlich einfacher. Sie lassen sich Datensatz- oder auch Block-weise mit unterschiedlichen Codes komprimieren. Dazu beinhalten sie auch noch Metadaten, die die Verarbeitung mit bestimmten Tools vereinfacht. Aber nicht jedes Tool aus dem Hadoop Ökosystem bringt Unterstützung für jedes File Format mit.

Neben den zeilenorientieren Formaten gibt es auch noch spaltenorientierte Formate:

- RCFile
- ORC File
- Parquet

Diese speichern die Daten pro Block spaltenweise. Dies ermöglicht eine effizientere Komprimierung und im Falle von ORC und Parquet sogar eine I/O-Reduzierung bei analytischen Queries durch einen sogenannten „Predicate-push-down“.

Beim logischen Schema Design gibt es zwei unterschiedliche Ansätze: das Star Schema und das Result Set Schema.

Im Star Schema Design wird die relationale Struktur eins zu eins nach Hadoop übernommen. Anpassungen bezüglich der Datenstruktur sind also weder beim Offload-Prozess noch im BI-Tool notwendig. Der Speicherbedarf ist ebenfalls recht günstig.

Warum also über eine Alternative nachdenken? Joins in Hadoop sind relativ teuer bzgl. Laufzeit und I/O.. Zwar unterscheiden sich hier die SQL Engines, eine Optimierung mit Indizes wie in der relationalen Datenbank ist aber nicht möglich.

In dem Result Set Design wird das komplette Star Schema komplett denormalisiert und eine einzige breite Tabelle erzeugt. Hier kommt der brachiale Parallelisierungsansatz von MapReduce voll zur Geltung. Allerdings auf Kosten des Speicherbedarfs.

Ein hybrider Ansatz ist natürlich auch möglich. Gerade kleine Dimensionstabellen könnten zusammengefasst oder direkt in die Faktentabelle denormalisiert werden.

Eine von der RDBMS abweichende Struktur erfordert natürlich entsprechende Transformationen beim Laden und auch bei der Konfiguration der Zugriffsschicht im BI-Tool.

Ähnlich wie in der Datenbank lassen sich auch in Hadoop die Daten partitionieren.

```
CREATE TABLE dim_kunde (
    id     INT,
    name   STRING,
```

```
    stadt   STRING,
    ...
)
PARTITION BY (
    land    STRING,
    region  STRING
);
```

Dies geschieht durch eine Aufteilung in entsprechende Unterverzeichnisse:

```
/dim_kunde/land=DE/region=N
/dim_kunde/land=DE/region=W
/dim_kunde/land=DE/region=O
/dim_kunde/land=DE/region=S
/dim_kunde/land=FR/region=N
...
```

Bei Dimensionen bietet sich eine Partitionierung nach fachlichen Schlüsseln an, um die Abfrageperformance zu optimieren.

Faktentabellen sollten nach der Zeit partitioniert werden. Dies grenzt nicht nur die Datenmenge bei den meisten Abfragen erheblich ein, sondern erleichtert auch den Offload-Prozess.

Hive Metadaten & SQL Engine

In Bezug auf das Zusammenbringen von SQL und Hadoop spielt Hive eine zentrale Rolle. Denn Hive bringt mit HiveQL nicht nur praktisch SQL auf den Hadoop Cluster, sondern mit HCatalog auch eine Metadatenschicht. In dieser werden per CREATE TABLE relationale Metadaten erzeugt. Aber Achtung: es wird nur ein Schema-on-Read definiert. Ob die Daten der Definition entsprechen, obliegt der Verantwortung des Anwenders. Auch gibt es keine Constraints.

Diese Metadatenschicht hat sich als Quasistandard für relationale Metadaten etabliert und entsprechend von anderen SQL Engines wie z.B. Impala aber auch von Tools wie z.B. Pig oder Sqoop verwendet. Und zwar können die Metadaten nicht nur gelesen, sondern auch direkt erzeugt werden.

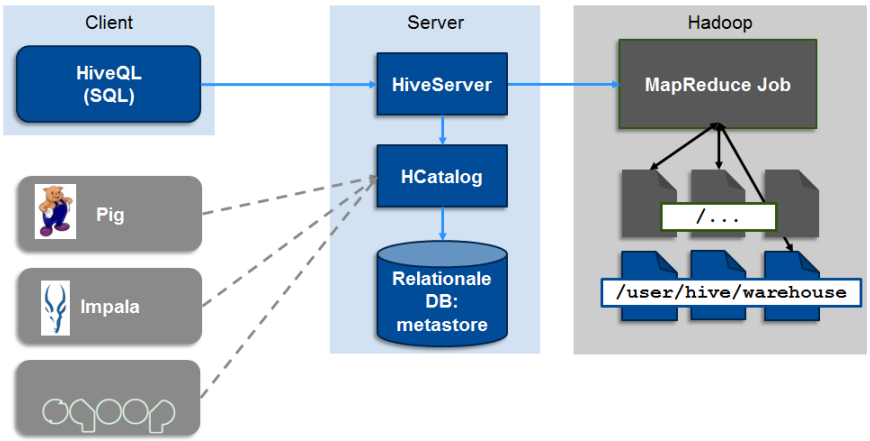

Abb. 4: Hive Architektur

Bei einer Hive-Abfrage wird das HiveQL (also SQL) ein einen oder mehrere MapReduce Jobs übersetzt. Dies bedeutet, dass es immer einen gewissen Overhead beim Verteilen des MapReduce-Codes auf die verschiedenen Knoten und vor allem beim Speichern der Zwischenergebnisse gibt. Jeder MapReduce-Job speichert sein Ergebnis nämlich wieder in HDFS. Der Folge-Job muss diese Daten dann erst wieder lesen usw.

Für die Verarbeitung von großen Datenmengen hat sich Hive aber durchaus bewährt, ist aber eher Batch-orientiert. Für Ad-hoc Analysen wirkt sich der Overhead sehr negativ aus.

Diese Lücke versuchen andere SQL Engines zu schließen. Tools wie Impala setzen auf eine in-memory-Verarbeitung ohne Zwischenspeicherung in HDFS. Dies erfordert natürlich eine entsprechende RAM-Ausstattung der einzelnen Knoten sowie eine zusätzliche Installation der Software auf jedem Knoten (s. Abb. 5).

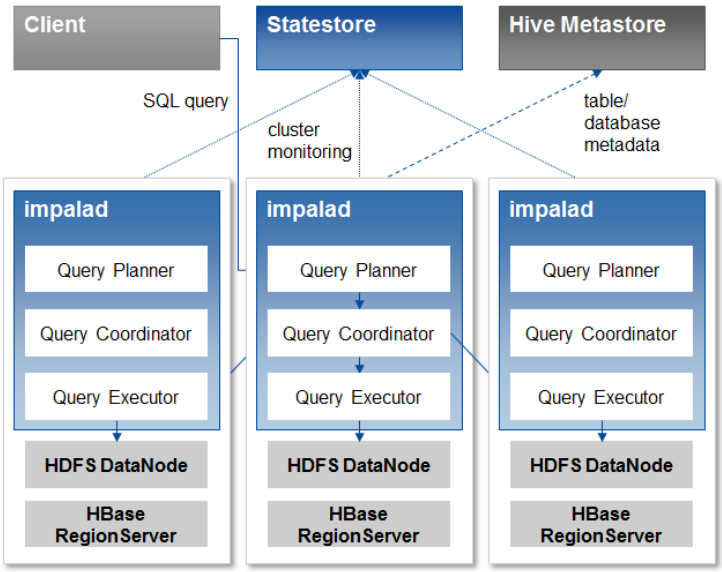

Abb. 5: Impala Architektur

Reicht der Speicher nicht aus, kann die Abfrage nicht beendet werden und das Query bricht ab.

Offload Prozess

Für den Datentransfer aus dem RDBMS nach HDFS gibt es verschiedene Wege. Wenn das eingesetzte ETL-Tool Hadoop als Ziel unterstützt, bietet es sich natürlich an, den Transfer mit dem ETL-Tool zu implementieren. So ist der gesamte ETL-Prozess ohne Technologiebruch umgesetzt.

Aber auch das Hadoop Ökosystem bringt mit Sqoop ein Tool mit, um Daten zwischen RDBMS und HDFS auszutauschen. Sqoop funktioniert als Kommandozeilen-Tool.

```
sqoop import
    --connect jdbc:oracle:thin:@//dbserver:1521/orcl
    --username scott
    --table dm_controlling.dim_kunde
    --where 'rownum=1'
```

```
--hive-import
```

Es lassen sich Tabellen komplett, in Teilen oder auch Ergebnisse von SQL Abfragen nach HDFS importieren.

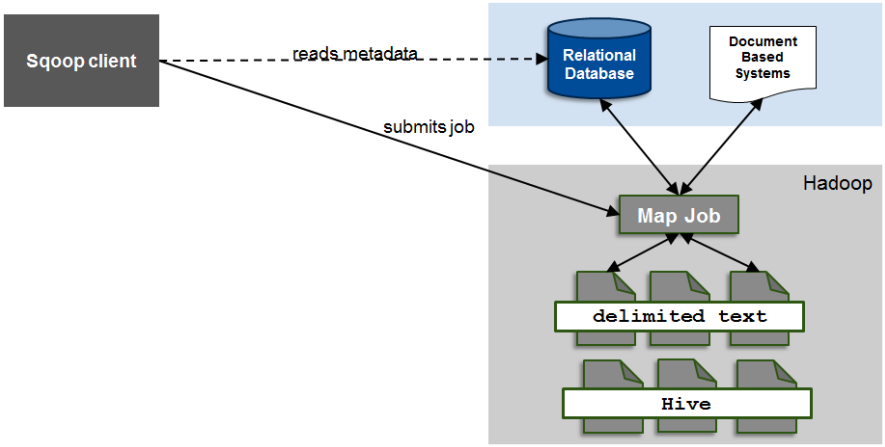

Abb. 6: Sqoop Architektur

Mit Sqoop können bei Bedarf auch die entsprechenden Hive-Metadaten erzeugt werden.

Wenn aus irgendwelchen Gründen die online-Verbindung zwischen RDBMS und Hadoop Cluster nicht möglich ist, bietet sich als letzter Ausweg der Datei-Transfer an. Hier werden die Daten aus der Datenbank in eine lokale Datei entladen. Diese wird dann auf einen Rechner, der Zugriff auf den Cluster hat, kopiert. Im letzten Schritt werden die Daten dann nach HDFS geschrieben. Entweder per File-Copy oder mit Flume, einem Tool zur Verarbeitung von Dateien und Events. Dieser Weg ist nicht nur am langsamsten, sondern auch am fehleranfälligsten.

Anbindung BI-Tool

Möchte man den Data Mart mit aktuellen Daten im RDBMS nutzen und mit kompletter Historie in Hadoop, benötigt man zwei verschiedene Connections. Eventuell weichen auch noch die physikalischen Modelle voneinander ab. Die logischen Modelle sollten dann wieder identisch sein. Hier hat man also schon einen doppelten Pflegeaufwand. Gleiches gilt dann auch für die Reports. Idealerweise benötigt man für die historischen Daten andere Reports. Oder man nutzt einen Data Mart nur für Ad-hoc Queries.

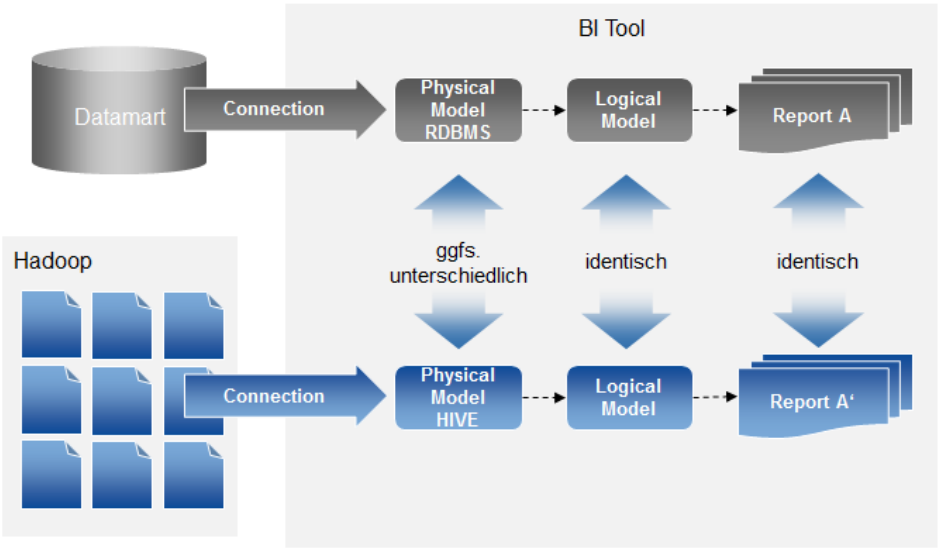

Abb. 7: Schichten im BI-Tool: Connection, Physical Model, Logical Model und Report

Diese Problematik tritt natürlich nicht auf, wenn vom BI-Tool aus nur Zugriffe auf den Hadoop Data Mart notwendig sind. In manchen Anwendungsfällen kann auch der Zugriff per SQL für einen ausgewählten Nutzerkreis eine sinnvolle Alternative sein.

Performance

Welches ist nun aber aus Performance-Gesichtspunkten die beste Kombination aus SQL-Engine, File Format, Compression Codec und Schema Design?

Diese Fragen wurden im Rahmen einer Zusammenarbeit mit der Technischen Universität Dresden untersucht. Die genauen Beschreibungen der Experimente und Ergebnisse findet man detailliert in der Masterarbeit „Real-time SQL on Hadoop “ von Xiaojing Zhao (August 2014). Aus dieser stammen auch die Abbildungen 8 – 10.

Alle Tests wurden auf einem Cluster mit 15 Data Nodes mit jeweils 96GB RAM durchgeführt. Als Hadoop Distribution kam CDH4 mit Hive (0.10) und Impala (1.2.4) zum Einsatz. Als Datenmodell kam ein Star Schema mit einer Faktentabelle und neun Dimensionstabellen aus dem TPC-DS Benchmark zum Einsatz.

Beim Laden der Faktentabelle sind einerseits die Ladeperformance und andererseits der benötigte Speicherplatz relevant. Wie man in Abbildung 8 erkennt, liefert Snappy die schnellste Ladezeit und zusammen mit Parquet auch eine recht gute Platzersparnis (60%). Mit GZIP lässt sich auf Kosten der Ladezeit eine noch etwas bessere Platzersparnis (70%) erzielen.

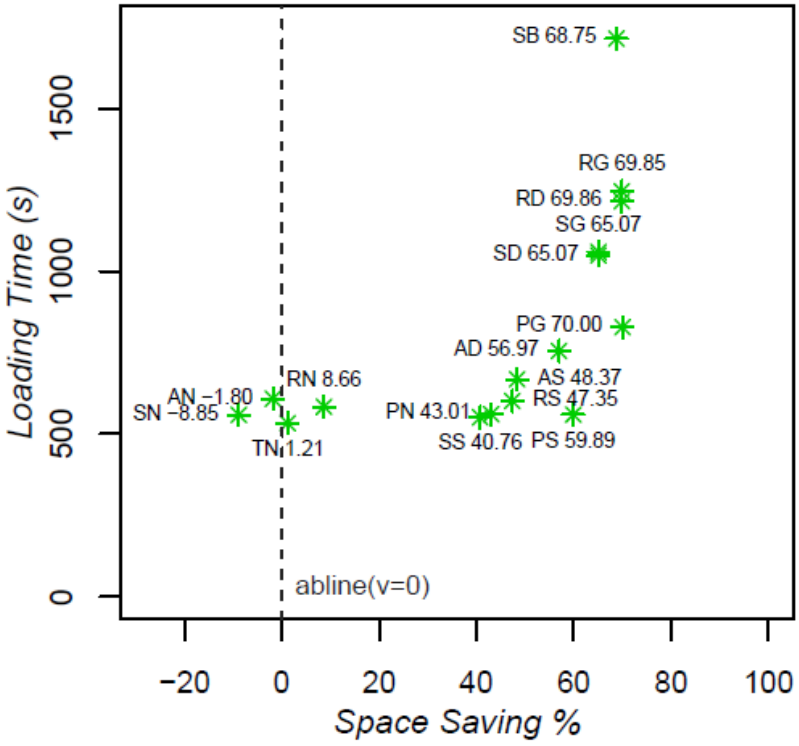

Abb. 8: Load Performance Faktentabelle mit 400GB

Oft dürfte die Abfrage-Performance aber wichtiger als die Ladeperformance sein. In der Abbildung 9 wird die Abfragezeit mit Hive untersucht. Auf der x-Achse findet man die Zeit für das Result Set Schema und auf der y-Achse die Zeit für das Star Schema. Unabhängig vom verwendeten File-Format und Compression Codec ist die Performance mit dem Result Set Schema etwas besser als mit dem Star Schema. Allerdings auf Kosten eines deutlich erhöhten Speicherbedarfs.

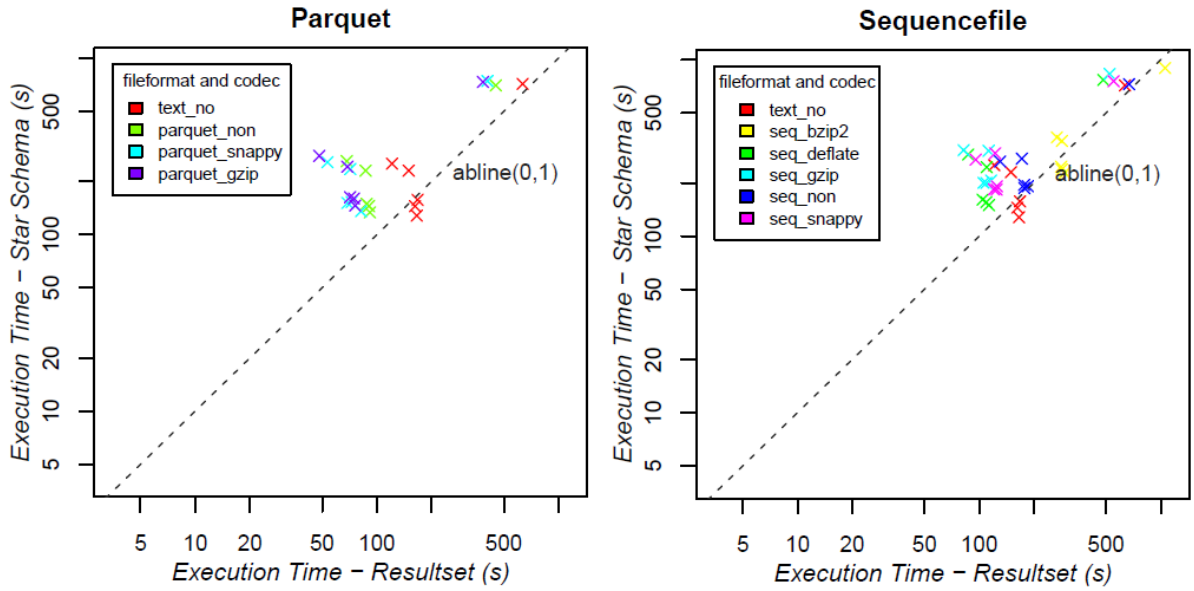

Abb. 9: Abfrage-Performance Hive

Abbildung 10 zeigt die Abfrage-Performance mit Impala in gleicher Weise wie Abbildung 9 sie mit Hive. Hier ist die Performance auf dem Star Schema besser als auf dem Result Set Schema, mit Ausnahme von ein paar kurzläufigen Queries.

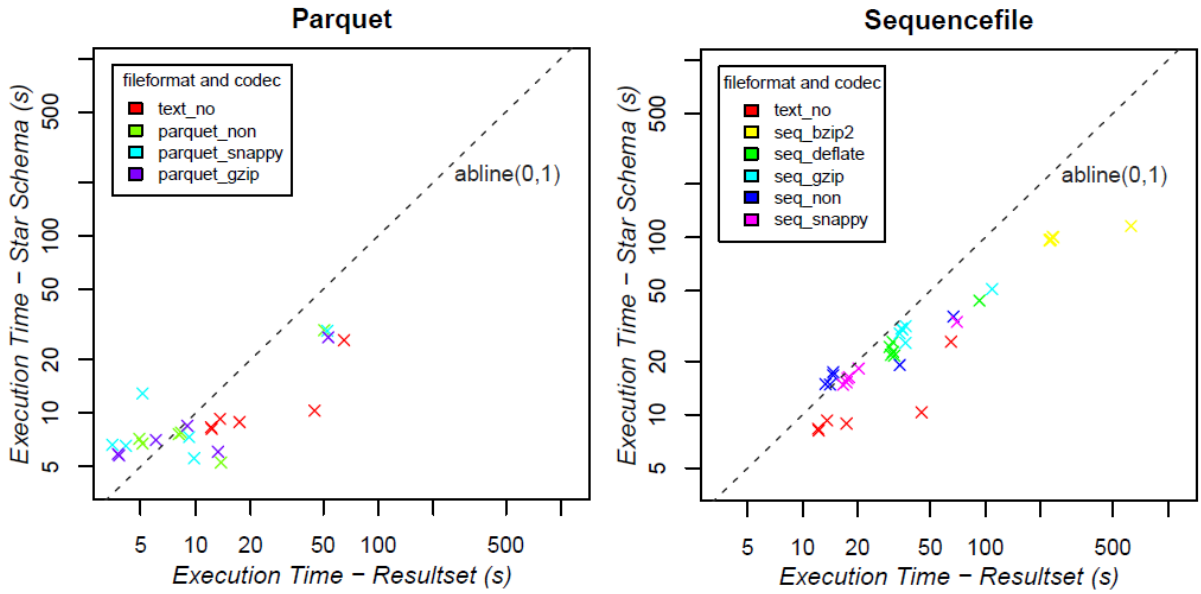

Abb. 10: Abfrage-Performance Impala

Insgesamt ist zu beobachten, dass die Performance von Impala deutlich besser als die von Hive ist. Voraussetzung ist allerdings genug Arbeitsspeicher auf den Data Nodes. Denn reicht der Speicher nicht aus, bricht das Query ab. Hive dagegen läuft viel robuster.

Ein Star Schema zusammen mit Snappy-komprimierten Parquet Files bildet eine recht gute Wahl für einen Data Mart in Hadoop Die SQL-Engine lässt sich ja recht einfach je nach Bedarf wechseln.

Zu beachten ist, dass alle Performance-Aussagen natürlich nur für die verwendeten Versionen von Hive (0.10) und Impala (1.2.4) gültig sind. Hier tut sich recht viel, und gerade Hive 0.13 verspricht drastische Performance-Verbesserungen.

Fazit

Es wurde gezeigt, welche Schritte notwendig sind um einen Data Mart Offload nach Hadoop umzusetzen. Die notwendigen Technologien wurden vorgestellt und die Vor- und Nachteile unterschiedlicher Varianten wurden beleuchtet.

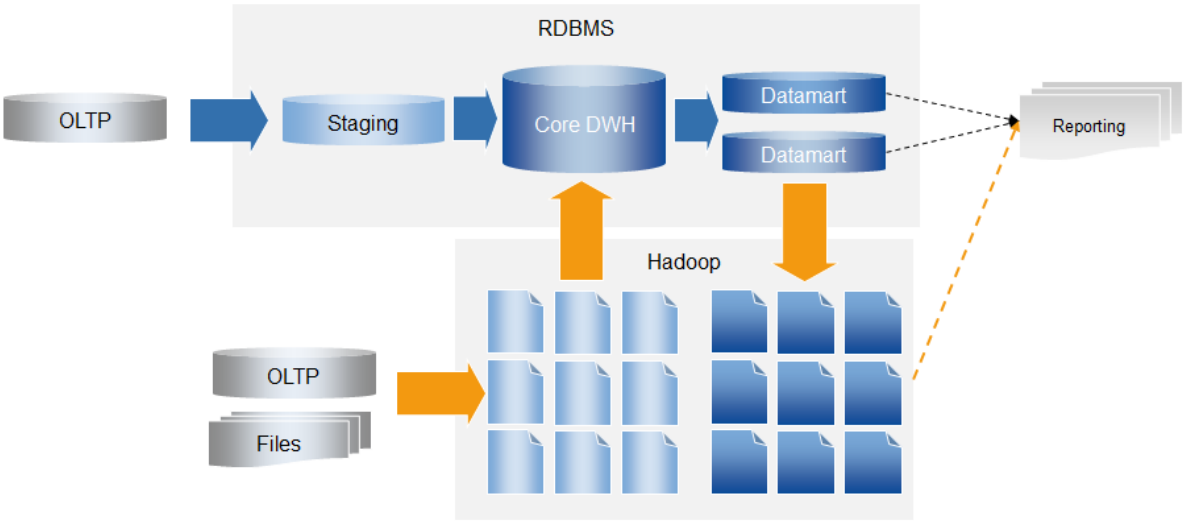

Abb. 1: DWH Architektur mit ETL Offload

Ein weiteres mögliches Einsatzszenario wäre auch der Offload der ETL Prozesse. Gerade im Staging Bereich kann der Einsatz von Hadoop mit seiner Stärke der parallelen Verarbeitung das RDBMS entsprechend entlasten. Alle hier vorgestellten Technologien und Modellierungsaspekte lassen sich entsprechend übertragen.

Kontaktadresse:
Carsten Herbe
metafinanz Informationssysteme GmbH
Leopoldstr. 146
D-80804 München

Telefon:	+49 (0) 89 360531-5039
Fax:	+49 (0) 89 360531-5015
E-Mail	carsten.herbe@metafinanz.de
Internet:	www.metafinanz.de