**UNIVERSITÄT LEIPZIG**

Institut für Informatik

# LARGE-SCALE DATENANALYSE

Seminararbeit

Leipzig, 30.01.2012

vorgelegt von:

Thanh Nghia Lam

Betreuer:

Diplom. Lars Kolb

# Inhaltsverzeichnis

# 1 EINLEITUNG

Analyse großer Datenmengen ist immer wichtigere Arbeit für viele Unternehmen seit meheren Jahren. Diese Arbeit kann heute noch wichtiger und schwieger sein. Da die Größe der Daten zur Zeit sehr schnell wachsen, die jeden Tag gesammelt und analysiert werden müssen. Es ist nicht ungewöhnlich heutzutage Daten in Petabyte zu sehen. Um die Zeit und Kosten bei großer Datenverarbeitung zu sparen und überhaupt erst die Bearbeitung von Petabytes an Daten in akzeptabler Zeit zu ermöglichen, werden spezielle Mechanismen und Algorithmen entwickelt. Deren Einsatzbereiche sind sehr verschieden und zahlreich z.B Kundendaten in großen Unternehmen, verschieden wissenschaflichen Analysen, Internet-Videoportal, Verwaltung der Datenbanken und natürlich Suchmaschinen. Schlüssel dafür ist die parallele Datenverarbeitung. Die gesamte Menge an Daten wird verteilt und an mehreren Rechnern parallel bearbeitet. Eine Lösung dafür ist die traditionelle Technologie Mehrrechner Datenbank für relationale Datenbank. z.B „*Massively Parallel Processing*“ (MPP) oder „*shared-nothing MPP*“.

Diese Systeme sind für die Speicherung und Abfragen von relationalen Daten optimiert. Aber bisher ist es schwer für diese Systeme, bis zu Tausende von Knoten zu skalieren. Einer der Gründe dafür ist ein Problem bei Skalierbarkeit oder bei Rechnerausfall. Außerdem ist es schwierig für diese Systeme bei Verarbeitung von nicht relationalen Daten.

Aus den vorgestellten Gründen wird ein Paradigma, sogenannt MapReduce System, erstellt. MapReduce ist ein neues Framework für verteiltes System (seine Open Source Implementierung ist Apache Hadoop) und wurde weitgehend wegen seiner beeindruckenden Skalierbarkeit und Flexibilität angenommen, um strukturierte sowie unstruktuierte Daten zu verarbeiten. MapReduce behandelt verschiedene Arten von Ausfällen sehr gut und kann deshalb leicht auf mehrere zehntausend Knoten skalieren. Es ist auch flexibler, jede Art von Daten zu verarbeiten. Andererseits, da es ein universelles System ist, fehlt noch eine deklarative Abfrageschnittstelle. Anwender müssen Codes schreiben, um auf die Daten zuzugreifen. Dieser Ansatz erfordert natürlich viel Aufwand und technische Fähigkeiten. Gleichzeitig können redundante Codes entstehen. Die Optimierung der Datenzugriffsschichten wird oft vernachlässigt. In dieser Arbeit wird ein System **Cheetah** vorgestellt, das zwei vorteilhafte Arten von Data Warehouse und MapReduce umfasst. Außerdem bietet das System noch eine SQL-ähnliche Anfragesprache und verschiedene Varianten zur Anfrage-Optimierung an.

Als andere Lösung im Vergleich mit MapReduce ist **Google Dremel** hier diskutiert. Während MapReduce komplett und komplizietere Aufgabe in zwei Hauptphasen: Map- und Reduce-Phase bearbeiten kann, ist Dremel für ein skalierbar, interaktiv Ad-Hoc Anfragessystem zur Analyse von schreibgeschützten verschachtelten Daten geeignet. Dremel kann aggregierte Anfragen über Trillionen Zeilen von Tabellen in Sekunden bearbeiten. Das System kann auch auf Tausende von CPUs und Petabyte an Daten skalierbar sein und hat tausende von Nutzern bei Google.

In Rahmen dieser Arbeit werden die beiden Systeme Cheetah und Dremel vorgestellt. Zunächst können einige technologische Aspekte in zwei Systemen verdeutlicht werden. Abschließend wird ein kurzes Fazit über die Anwendungen bzw. Ansätze erstellt.

# 2 GOOGLE DREMEL

Dremel ist seit 2006 in der Fertigung entwickelt und hat tausende von Benutzer innerhalb Google. Dremel ist ein interessanter Schritt von Google, das ein interaktives Ad-hoc-Anfrage-System für die Analysen von schreibgeschützten, verschachtelten Daten bietet. Es wurde erstellt, weil MapReduce zuviel Latenz für eine schnelle interaktive Anfrage oder Analyse hat.

Durch die Kombination von Multi-Level-Ausführungsbäume und spaltenorientierten Datenanordnung ist es lauffähig, Aggregation über Billionen zeilenorientierten Tabellen in Sekunden abzufragen. Das System skaliert Tausende von CPUs und Petabyte Daten. Ein Überblick über Architektur und Implementierung von Dremel werden hier beschrieben und erklärt, wie es das auf MapReduce basierten Computing ergänzt.

Dremel ist nicht als Ersatz für MapReduce gedacht, sondern wird häufig im Zusammenhang mit MapReduce verwendet, um Ausgänge des MR-Pipelines oder rascher Prototypen größer Berechnungen zu analysieren.

## 2.1 Architektur

Dremel basiert auf den Ideen von Web-Suche und parallel DBMS. Seine Architektur leiht das Konzept eines Dienstbaumes, der in verteilte Suchmaschinen verwendet ist. Genau wie ein Web-Suchabfrage, geht eine Abfrage in den Baum von oben nach unten und wird auf jeder Stufe neu geschrieben. Das Ergebnis der Abfrage wird rekursiv durch aggregierte Antworten zusammengestellt, welches aus den unteren Ebenen des Baumes erhalten wird. Ebenfalls bietet Dremel eine hochrangige, SQL-ähnliche Sprache, um Ad-hoc-Abfragen auszudrücken. Im Gegensatz zu Schichten wie **Pig** und **Hive**, führt Dremel seine Abfragen nativ ohne Umsetzung in MapReduce Jobs aus.

Dremel verwendet eine zerlegte spaltenorientierte Speicherdarstellung, die dem Dremel ermöglicht, weniger Daten aus sekundären Speicher zu lesen und CPU-Kosten durch günstige Datenkompression zu reduzieren.

Dremels Speicherung ist auf dem Google File System (GFS) bzw. BigTable aufgebaut, welche eine verteilte gemeinsame Speicherschicht anbietet. GFS verwendet Replikationstechnik, um die Daten trotz fehlerhafter Hardware zu erhalten und schnelle Reaktionszeiten zu erreichen. Ein leistungsstarke Speicherschicht ist kritisch für eine *in-situ* Datenverwaltung. Es erlaubt, den Zugriff auf die Daten ohne eine zeitaufwendige Ladephase. Außerdem können Daten in dem GFS Dateisystem bequem mit Standard-Werkzeugen bearbeitet werden, z.B Datenübertragung zu einem anderen Cluster, Änderung von Zugriffsrechten oder Identifizierung der Untergruppe von Daten für die Analyse auf Dateinamen.

## 2.2 Datenmodell

Datenmodell, das im Rahmen verteilter Systeme nähmlich "*Protocol Buffers*" entstanden ist, wird es weit verbreitet bei Google verwendet und ist verfügbar als Opensource Implementierung. Das Datenmodell basiert auf *strong-typed* verschachtelte Sätze. Die abstrakte Syntax sieht wie folgende Formel aus:

$$\tau = \mathbf{dom} \mid \langle A_1 : \tau[*|?], \ldots, A_n : \tau[*|?] \rangle$$

Γ ist ein elementarer Typ oder ein Datensatztype. Elementare Typen in **dom** bestehen aus *Integers*, *Floating-point Number, String* usw. . Datensätze erhalten ein oder mehrere Felder. Feld *i* in einem Datensatz hat einen Namen $A_i$ und eine fakultative Vielfalt von Beschriftungen. Wiederholte Felder (*repeated*) (*) können mehrfach in einem Datensatz auftreten. Sie werden als Wertlisten interpretiert, d.h die Reihenfolge der vorkommenden Felder ist signifikant in einem Datensatz. Fakultative Felder (*optional*) können aus dem Datensatz fehlen. Andernfalls würde ein Feld erfordert (*required*), so muss das Feld genau einmal im Datensatz vorkommen. Das folgende Beispiel wird das deutlich machen.

```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward; }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; }
    optional string Url; }}
```

```
DocId: 10          r1
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

## 2.3 Verschachtete spaltenorientierte Speicherung

Dremel nutzt eine Datensdarstellung in Spalte-gestreifte Speicherung. Es bringt mehrere Vorteile: Daten wenig von dem sekundären Speicher auszulesen, z.B wenn ein Aggregat über viele Zeilen, aber nur wenige Spalten gebildet werden muss, da man dann im Gegensatz zum zeilenorientierten System nur diese und nicht alle Spalten lesen muss; oder wenn eine Spalte gleichzeitig für alle Zeilen der Tabellen einen neuen Wert erhält, da man die Spaltendaten effizient schreiben kann und die Daten der anderen Spalten nicht berücksichtigen muss [1]. Dabei reduziert es CPU-Kosten durch günstige Komprimierung. Für eine Analyse der relationalen Daten ist das spaltenorientierte Speicherformat mehr verwendet worden. Aber sie sind bisherig noch nicht unter dem verschachtelten Datenmodell eingebracht. Aus dem Grund wird ein neuartig spaltenorientiertes Speicherformat für Anwendungen belieb-iger verschachtelter Daten in mehreren Datenverarbeitungs-Werkzeugen bei Google z.B MapReduce, Sawzall und FlumeJava entwickelt. Dafür werden Algorithmen für Zerlegung verschachtelter Daten-sätze in Spalten und Rekonstruierung zerlegter Datensätzen vorgestellt.

Wie in *Abbildung **1*** dargestellt, ist *r1* beispielweise ein verschachter Datensatz, von dem aller Werte eines bestimmten Feldes nacheinander abzuspeichern sind, um einen effizienten Retrieval zu verbessern. Es genügt nicht, die Struktur von Datensätzen zu übermitteln, wenn nur die Datenwerte in *r1* in Betracht kommen, z.B wenn zwei Werte von wiederholten Feldern vorgegeben sind, so ist es unbekannt, bei welchem Feld bzw. auf welcher Stufe der Wert sich wiederholt hat. d.h es ist möglich, dass die Werte aus zwei verschiedenen Datensätzen entstehen oder sie sich im gleichen Datensatz befinden. Ebenfalls, wenn ein fakultatives Feld vorgegeben ist, ist es unbestimmt, welche einschließenden Datensätze genau definiert wurden.

Aus diesen Gründen werden die Konzepte von *Wiederholungsstufe* und *Definitionsstufe* sinnvoll definiert.

### 2.3.1 Wiederholungsstufe und Definitionsstufe

***Wiederholungsstufe***:

Um das Konzept klar zu verstehen, ist das Feld *Code* in der ***Abbildung 1*** betrachtet. Das Feld entsteht ingesamt dreimal im Datensatz *r1*, nämlich *„en-us“* und *„en“* treten in ersten *Name* auf, während *„en-gb“* sich in dritten *Name* befindet. Um die Positionen von Auftreten zu unterschieden, sollte ein Wert einer wiederholte Stufe an jedem Datenwert eingefügt sein .d.h bei der Wiederholungsstufe zeigt *„bei welchem wiederholten Feld in dem Feldpfad ist der Wert wiederholt?“*. Der Feldpfad *Name.Language.Code* erhält zwei wiederholte Felder (*Name* Gruppe *und Language* Gruppe) nach dem vorgegeben Schema. Der Datensatz *r1* wird von oben nach unten nachgeprüft. Bevor der Wert *„en-us“* auftritt, findet es noch kein wiederholtes Feld z.B *Name* oder *Language*, so ist die Wiederholungsstufe gleich 0. Wenn *„en“* danach angesehen wird, hat das Feld *Language* einmal wiederholt,

so ist die Wiederholungstufe gleich 2. Wenn der Wert *„en-gb“* anschließend auftritt, ist dann seine Wiederholungstufe gleich 1, weil das Feld *Name* bei dem Auftritt wiederholt.

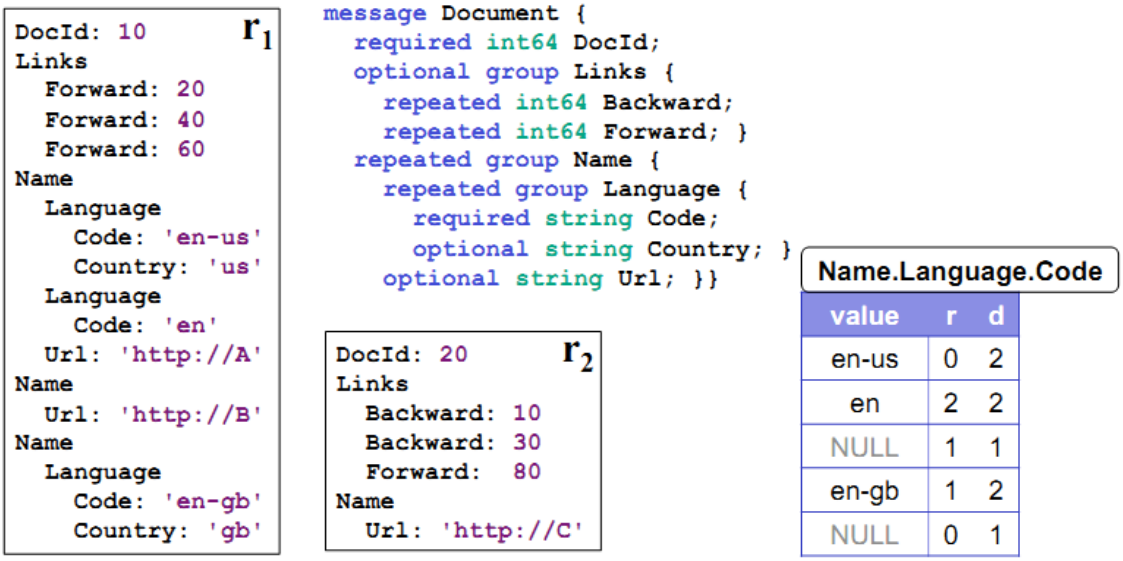


*Abbildung 1: beispielweise verschachtelte Datensätzen und ihre Schema*

Der zweite *Name* in *r1* wird dafür bemerkt, dass er keine *Code*-Werte enthält. Um die richtige Position von dem Wert „*en-gb*“ im dritten *Name* zu übermitteln, sollt ein Wert „*NULL*“ zwischen „*en*“ und „*en-gb*“ eingefügt werden.

Da *Code* Feld in *Language* erforderlich ist, so wenn es in *r1* fehlt, würde es darüber impliziert, dass das *Language* auch nicht definiert wird. Im Allgemein erfordert eine Bestimmung einer Stufe, die von seiner verschachtelten Datensätzen abhängig sind, zusätzliche Information.

***Definitionsstufe***

Jeder Feldwert in einem Pfad *p*, besonders jeder *NULL* hat eine Definitionsstufe, die dazu spezifiziert, *„wie viele Felder in p wegen wiederholter oder fakultativer Definition nicht definierbar sind, aber tatsächlich präsentiert werden müssen“*. d.h es wird für Definitionsstufe nur das wiederholte Feld und fakultative Feld interessant sein.

z.B. *r1* hat kein Feld *Backward* unter dem *Link*. Aber Feld *Links* ist definiert bei Stufe 1. Um die Information (Stufe = 1) zu erhalten, wird ein *NULL* Wert mit der Definitionsstufe 1 auf die Spalte *Links.Backward* eingefügt. In ähnlicher Weise führt das fehlende Auftreten von *Name.Language.Country* in *r2* eine Definitionsstufe 1 bzw. sein fehlendes Auftreten in *r1* haben die Definitionsstufe 2 (es tritt innerhalb unter dem *Name.Language* auf) und die Definitionsstufe 1 (innerhalb von *Name*).

***Encoding***

Jede Spalte ist als eine Menge von Blocken abgespeichert. Jeder Block enthält die Wiederholungs- und Definitionsstufe und die komprimierten Feldwerte.

*NULL*-Werte sind nicht explizit gespeichert, als sie durch die Definitionsstufen bestimmt werden: jede Definitionsstufe bezeichnet eine *NULL*, wenn der Wert der Definitionsstufe kleiner als die Anzahl der wiederholten und fakultativen Felder in einem Feldpfad ist.

Z.B. die 2. Zeile in der Spalte *Name.Language.Country* in ***Abbildung 2*** beschreibt davon, dass *NULL* Wert jetzt die Wiederholungsstufe 2 und Definitionsstufe 2 hat. Da die Anzahl der wiederholten und fakultativen Felder im Pfad *Name.Language.Country* gleich 3 ist (*repeated:Name; repeated:Language und optional:Country*), bestimmt es implizit durch die Stufen den Wert gleich *NULL*. Analog, bei 3.Zeile (1< 3) und 5.Zeile (1<3).

Außerdem sind Definitionsstufen nicht für die Werte gespeichert, die immer definiert werden. In ähnlicher Art und Weise können die Wiederholungsstufen nur bei Notwendigkeit abgespeichert

werden. z.B Definitionsstufe 0 impliziert ewig Wiederholungsstufe 0, so dass diese weggelassen werden kann. In der ***Abbildung 2*** sind keine Stufen für *DocId* gespeichert.

### 2.3.2 Zerlegung von Datensätzen in Spalten

Wie oben wird eine Kodierung des Datensatzes in einem Spaltenformat präsentiert. Die nächste angehende Herausforderung ist, wie spaltenorientierte Daten mit Wiederholungs- und Definitionsstufe effizient produziert werden. Der Basisalgorithmus zur Berechnung der Wiederholungs- und Definitionsstufe befindet sich im Anhang A in [3]. Der Algorithmus läuft rekursiv in die Datensätzen und berechnet damit die Stufen für jeden Feldwert, auch wenn Feldwerte fehlen.

**DocId**

| value | r | d |
|---|---|---|
| 10 | 0 | 0 |
| 20 | 0 | 0 |

**Name.Url**

| value | r | d |
|---|---|---|
| http://A | 0 | 2 |
| http://B | 1 | 2 |
| NULL | 1 | 1 |
| http://C | 0 | 2 |

**Links.Forward**

| value | r | d |
|---|---|---|
| 20 | 0 | 2 |
| 40 | 1 | 2 |
| 60 | 1 | 2 |
| 80 | 0 | 2 |

**Links.Backward**

| value | r | d |
|---|---|---|
| NULL | 0 | 1 |
| 10 | 0 | 2 |
| 30 | 1 | 2 |

**Name.Language.Code**

| value | r | d |
|---|---|---|
| en-us | 0 | 2 |
| en | 2 | 2 |
| NULL | 1 | 1 |
| en-gb | 1 | 2 |
| NULL | 0 | 1 |

**Name.Language.Country**

| value | r | d |
|---|---|---|
| us | 0 | 3 |
| NULL | 2 | 2 |
| NULL | 1 | 1 |
| gb | 1 | 3 |
| NULL | 0 | 1 |

*Abbildung 2: Spaltenorientierte Darstellung von zwei Datensätzen zzgl.ihre Wiederholungs- und Definitionsstufe*

Zahlreiche bei Google verwendete Datensätze sind spärlich, es ist nicht ungewöhnlich, dass ein Schema mit Tausenden von Feldern aber nur Hunderte von denen, in einem vorgegebenen Datensatz verwendet werden müssen. Daher versucht es, fehlende Felder so einfach wie möglich zu verarbeiten. Um gestreifte Spalten zu produzieren, wird ein Baum des Feld-Schreiberes erzeugt, dessen Struktur mit den hierarischen Felder im Schema übereinstimmt.

Die Grundidee ist es, dass der Feld-Schreiber sich nur aktualisiert, wenn er seine eigenen Daten besitzt. Er versucht nicht, übergeordneten Zustand nach unten in den Baum zu propagieren, wenn es nicht unbedingt nötig ist. Um dies zu erreichen, erbt Kind-Schreiber die Stufen von seinen Eltern. Ein Kind-Schreiber synchronisiert mit seiner Eltern-Stufe, sobald ein neuer Wert hinzugefügt wird. Die *Abbildung **2*** zeigt die alle Resultaten nach Aufteilung von Datensätzen *r1* und *r2* in Spalten.

### 2.3.3 Rekonstruierung von zerlegten Datensätzen

Rekonstruierung der in spaltenorientierte Daten zerlegten Datensätze ist effizient kritisch für zeilenorientierte Datenverarbeitungs-Werkzeuge (z.B. MapReduce). Würde eine Teilmenge von Felder vorgegeben, ist es Ziel dafür, die Ursprungdaten zu rekonstruieren, obwohl die Teilmenge nur ausgewählte Felder enthält, während alle anderen Felder weggelassen sind.

Die zentrale Idee ist, eine endliche Zustandsmaschine (FSM) zu definieren, die die Feldwerte und Stufen für jedes Feld liest und die Werte sequentiell an die ausgegebenen Datensätzen anfügt. Ein FSM-Zustand entspricht einem Feld-Leser für jedes ausgewählt Feld. Zustandsübergänge werden mit Wiederholungstufen gekennzeichnet. Sobald ein Leser einen Wert holt, wird die nächste Wiederholungstufe entscheiden, welcher nächster Leser betrachtet wird. Die FSM läuft von Anfang bis Ende einmal für jeden Datensatz durch.

Die *Abbildung **3*** zeigt eine kleine Zustandsmaschine als ein korrektes Beispiel. Der Startzustand ist *DocId*. Wenn ein *DocId*-Wert gelesen wird, geht die FSM weiter zu *Links.Backward* über. Nachdem

alle wiederholte *Backward*-Werte abgelassen wurde, springt die FSM zu *Links.Forward* usw. Die Details des Algorithmus zur Rekonstruierung von Datensätze wird in Anhang B in [3] angehängt.

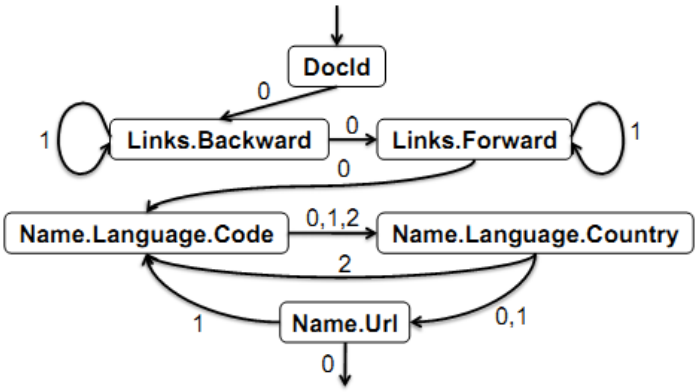


*Abbildung 3: Automat für die vollständigen Rekonstruierung der Ursprungdaten*

Um die Frage „wie die FSM Übergänge gebaut werden“ zu skizzieren, ist angenommen: Sei l die nächste Wiederholungstufe, die durch das aktuelle Feld-Leser für Feld *f* zurückgegeben würde. Beginnend bei Feld *f* im Baumschema, ist es gefunden, dass sein Vorfahren auf Stufe 1 wiederholt und das erste Feld-Blatt *n* innerhalb dieser Vorfahren ausgewählt wird. Dies gibt eine FSM-Übergang (*f;l*)->*n* an. z.B sei *l* = 1 die nächste Wiederholungsstufe durch Lesen von Feld *f* = *Name.Language.-Country*. Das Vorfahren von *f* mit der Wiederholungsstufe 1 ist *Name*, deren erstes Feld-Blatt *n=Name.Url* ist.

Die Details des Algorithmus zum FSM-Aufbau ist in Anhang C in [3] dargestellt. Wenn nur ein Teil der Felder abgefragt werden muss, wird eine einfachere FSM konstruiert, die einfacher zu führen ist. *Abbildung **4*** stellt ein FSM für das Lesen der Felder *DocId* und *Name.Language.Country*. dar. Die Abbildung zeigt die ausgegeben Datensätzen $s_1$ und $s_2$, die von dem Automaten produziert werden.

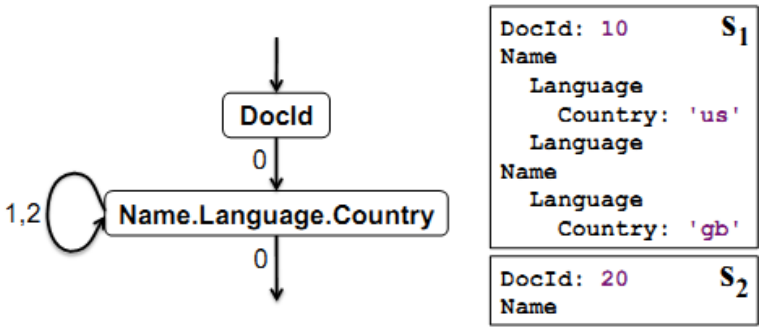


*Abbildung 4 : Automat für die vollständiger Rekonstruierung der Datensätze aus zwei Feldern; die neu ausgegebenen Datensätze*

Dremels Kodierung und Versammlung-Algorithmus bewahrt die Strukturen des Feldes *Country*. Dies ist wichtig für Applikationen, die für den Zugriff benötigt werden.

Z.B das Feld *Country* tritt in dem ersten *Language* bei zweiten *Name* auf. In Xpath kann der Zugriff auf *Country*-Wert direkt wie hier ausgedrückt werden: */Name[2]/Language[1]/Country*.

## 2.4 Anfragesprache

### 2.4.1 Übersicht

Dremels Anfragesprache basiert auf SQL und wird zur effizienten Realisierung auf säulenförmiger verschachtelter Speicherung angewandt. Die Ideen sind dafür, dass jede SQL-Anweisung eine oder

mehrere verschachtelte Tabellen und ihre Schemata als Eingabedaten übernimmt. Nach der Ausführung der Anweisungen werden angeforderte verschachtelte Datensätze und ihre entsprechenden Schemata produziert. Die *Abbildung **5*** zeigt ein Beispiel über eine Anfrage, die Projektion, Selektion und Aggregation von Datensätzen führt. Die Anfrage wird über die Tabelle *t* mit 2 Datensätzen *r1*, *r2* ausgewertet. Die Felder sind mit Pfadausdrücken verwiesen. Die Anfrage erzeugt ein verschachteltes Resultat, obwohl es keinen Datensatz-Konstruktor in der Anfrage gibt.

Um die Anfrage-Realisierung zu erklären, wird zuerst die Selektionsoperation (im WHERE-Klausel) betrachtet. Ein verschachtelter Datensatz ist als ein markierter Baum zu verstehen, wo jedes Label einem Feldnamen entspricht. Der Selektionoperator führt durch die Zweige des Baumes, die den angegebenen Bedingungen entsprechen. Deshalb werden nur die verschachtelten Datensätze erhalten, wo die *Name.Url* definiert wird und mit *http* beginnt. Als nächste Betrachtung ist der Projektionsoperator eingeführt. Jeder Skalarausdruck in der SELECT-Klausel gibt einen Wert auf der gleichen Stufe der Verschachtelung als den am meisten wiederholten Eingabefeld an, der in diesem Ausdruck verwendet wird. Der Ausdruck von Zeichenfolgenverkettung emittiert die *Str*-Werte auf der Stufe der *Name.Language.Code* in das Eingabeschema.

```
SELECT DocId AS Id,
  COUNT(Name.Language.Code) WITHIN Name AS Cnt,
  Name.Url + ',' + Name.Language.Code AS Str
FROM t
WHERE REGEXP(Name.Url, '^http') AND DocId < 20;
```

```
Id: 10                                  t1
Name
  Cnt: 2
  Language
    Str: 'http://A,en-us'
    Str: 'http://A,en'
Name
  Cnt: 0
```

```
message QueryResult {
  required int64 Id;
  repeated group Name {
    optional uint64 Cnt;
    repeated group Language {
      optional string Str; }}}
```

*Abbildung 5 : Beispielabfrage, ihr Resultat und ausgegebenes Schema*

Der *COUNT*-Ausdruck stellt eine Aggregation für Datensatz dar. Die Aggregation ist mit jedem *Name*-Teildatensatz durchgeführt und emittiert die Anzahl von Auftreten der *Name.Language.Code* für jeden Namen als *uint64*.

Die Dremels Anfragesprache unterstützt verschachtelte Unteranfragen, *inter* und *intra*- aggregierten Datensatz, top-k, Verbund, benutzerdefinierte Funktionen, usw.

### 2.4.2 Hierarchische Anfragesausführung

Die Dremel-Anfragen sind am meisten Aggregationen in einem Durchlauf.

***Baumarchitektur*** Dremel nutzt einen mehrstufigen Dienstbaum (*serving tree*) zur Anfrageausführung (*Abbildung **6***). Ein root-Server enthält eingehende Anfragen und liest Metadaten aus der Tabelle, die in dieser Anfrage existieren und routet die Anfragen zur näschten Stufe im Dienstbaum. Der Blatt-server kommuniziert mit der Speicherschicht oder greift auf die Daten auf der lokalen Festplatte zu.

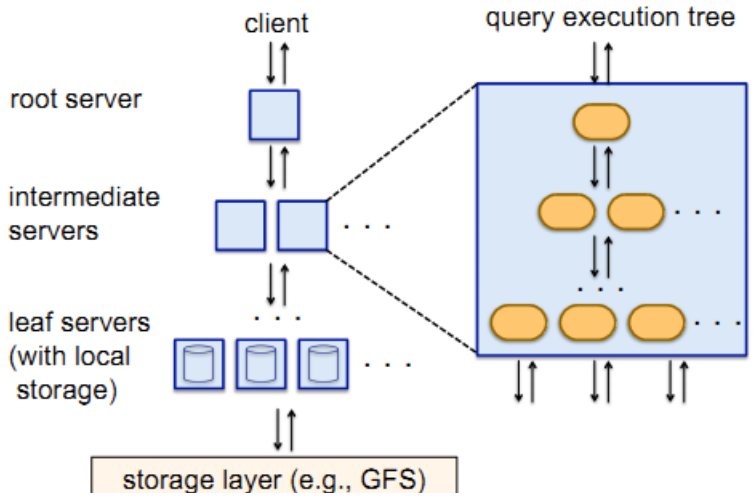


*Abbildung 6 : Systemarchitektur und die Ausführung innerhalb einem Serverknoten*

Es wird nun folgendes Beispiel betrachtet:

$$SELECT\ A,\ COUNT(B)\ FROM\ T\ GROUP\ BY\ A$$

Hättet der root-Server die Anfrage bekommt, ermittelt er alle „kleine“ Tabellen d.h horizontale Partitionen der Tabellen, die die Tabelle *T* umfasst und schreibt die eingehende Anfrage als neue folgende Anfrage um:

$$SELECT\ A,\ SUM(c)\ FROM\ (R^1_1\ UNION\ ALL\ ...\ R^1_n)\ GROUP\ BY\ A$$

Die Tabellen $R^1_1$ ,..., $R^1_n$ sind die Resultate von Anfragen, die zu den Knoten *1,...,n* in der Stufe 1 des Dienstbaumes gesendet werden:

$$R^1_i = SELECT\ A,\ COUNT(B)\ AS\ c\ FROM\ T^1_i\ GROUP\ BY\ A$$

$T^1_i$ ist eine disjunkte Partition von *Tablets* in *T*, die durch Server *i* auf Stufe 1 verarbeitet wird. Jede Dienststufe führt eine ähnliche Anfrageumschreibung. Letztendlich erreichen die Anfragen die Blätter, wo die Daten lokalisiert werden können. Auf dem Weg hinauf, führen dazwischenliegenden Server eine parallele Aggregation von Teilergebnissen durch. Die wie oben vorgestellte Modellausführung ist sehr gut für die aggregierte Anfragen geeignet, die nur kleine und mittlere Ergebnisse abfragen, die eine sehr häufige Klasse von interaktiven Abfragen sind. Große Ansammlungen und andere Klassen von Abfragen müssen möglicherweise auf die Ausführung von parallelen Datenbanksystemen und MapReduce bekannt vertrauen.

***Anfrageverteiler (Query dispatcher).*** Dremel ist ein *Multi-User System*, d.h in der Regel können mehr-ere Anfragen gleichzeitig ausgeführt werden. Ein Anfrageverteiler ordnet Anfragen auf der Basis von ihren Prioritäten ein und verteilt die Belastung. Seine weitere wichtige Aufgabe ist es, die Ausfall-sicherheit zu erhöhen, wenn ein Server viel langsamer als andere ist oder eine Tablette-Replikat nicht mehr erreichen kann.

Die Datenmenge in jeder verarbeiteten Anfrage ist oft größer als die Anzahl von Verarbeitungseinheiten für die Ausführung, die als ***Slot*** genannt wird. Ein Slot entspricht einer Ausführungs-Thread auf einem Blattserver.

z.B. Ein System has 3000 Blattserver. Jeder Blattserver nutzt 8 Threads. So bietet das System insgesamt 24000 Slots zur Anfrageausführung. Eine Tabelle, die in 100.000 Tabletten überspannt, kann durch Zuordnung etwa 5 Tabletten in jeden Slot verarbeitet werden. Während der Ausführung der Anfrage berechnet die Anfrageverteiler ein Histogramm der Tablette-Bearbeitungszeiten. Wenn eine Tablette eine unverhältnismäßig lange Zeit in Anspruch nimmt, es planet sie auf einem anderen Server um.

Die Blattservern lesen verschachtelten Daten in säulenartigen Darstellung. Die Blöcke in jedem Streifen werden asynchron vorausgelesen. Tabletten sind in der Regel Drei-Wege-Replikation. Wenn ein Blattserver kann nicht auf eine Tablette-Replikat zugreifen, fällt er auf einen anderen Replikat.

Jeder Server hat einen inneren Ausführungsbaum, der wie auf der rechten Seite in der ***Abbildung 6*** dargestellt wird. Der interne Baum entspricht einem physikalischen Abfrageausführungsplan, einschließlich der Bewertung von skalaren Ausdrücken. Zur Optimierung wird ein Typ-spezifischer Code für die meisten skalaren Funktionen generiert. Ein Ausführungsplan für Project-Select-Aggregation Anfragen besteht aus einer Reihe von Iteratoren, die Eingabespalten im Gleichschritt scannen und emittieren Ergebnisse von Aggregaten und skalaren Funktionen , die mit den richtigen Definitionsstufen, Wiederholungsstufen annotieren. Die Datensatz-Rekonstruktion kann komplett während der Abfrage erfolgen. Einige Dremel-Anfragen wie *top-k* und *count-distinct* liefert ungefähre Ergebnisse unter Verwendung eines *„one-pass“* Algorithmen zurück.

## 2.5 MapReduce vs Dremel

| Featuren | MapReduce | Dremel |
|---|---|---|
| **Geschichte** | Seit 2004 bei Google Lab | Seit 2006 bei Google Lab |
| **Type** | Verteilt und parallel Programming Framework | Verteilt interaktive Ad-Hoc Anfragessystem |
| **Skarlierbarkeit und Fehlertoleranz** | ++ | ++ |
| **Datenverarbeitung** | Zeilenorientiert | Spaltenorientiert |
| **Batchsverarbeitung** | Ja | Nein |
| **In situ-Verarbeitung** | Nein | Ja |

Die MapReduce Framework ist zur Bearbeitung der großen Skarlierbarkeit in Computing im Rahmen von lang laufenden Batchjobs entwickelt. Ähnlich wie MapReduce, Dremel unterstützt neben einer Fehlertoleranz einer Ausführung noch ein flexibles Datenmodell und *in situ* Datenverarbeitung.

Dremel ist zur Bedienung bei Skalierbarkeit entworfen. Obwohl es denkbar ist, dass parallel DBMS zur Skalierung bis zu Tausenden von Knoten bearbeitet werden können. Aber es ist noch unbekannt, veröffentliche Arbeiten oder Industrieberichte darüber zu finden, besonders mit einer Untersuchung des MapReduces auf säulenförmige Speicherung.

Die im Dremel vorgestellte säulenförmige Darstellung von verschachtelten Daten basiert auf Ideen, die aus mehreren Jahrzehnten stammen: Trennung von Struktur aus Inhalt und umgesetzte Darstellung. Das in Dremel verwendete Datenmodell ist eine Variante der komplexen Wert-Modelle und verschachtelten Modelle, die in [5] diskutiert werden.

Die Dremel-Anfragesprache baut auf den Ideen von [6] auf. Die Sprache vermeidet Umstrukturierung beim Zugriff auf verschachtelte Daten. Im Gegensatz dazu wird Umstrukturierung in der Regel in Xquery und objekt-orientierte Abfragesprachen erforderlich, z.B verschachtelte Nutzung der *for*-Schleife und Konstruktoren. Eine aktuelle SQL-ähnliche Sprache, die auf verschachtelte Daten betreibt, ist **Pig**.

# 3 CHEETAH

Während das MapReduce System flexible und skalierbar ist, müssen die Benutzer viel Mühe dazu aufwenden, ein MapReduce Program zu schreiben. Da das Map-Reduce nur ein Ausführungsmodell ist, so lassen sich die grundlegende Datenspeicherung und Zugriffsmethode den Benutzern komplett implementieren. Dies bietet dafür vorteilhaft einige Flexibilität, aber es fehlt leider eine optimierte Gelegenheit, wenn die Daten selbst etwas in struktuierter Form hätten. Relationale Datenbanken haben die oben genannten Fragen für eine lange Zeit behandelt, z.B es existiert bereits eine deklarative Anfragesprache wie SQL. Die Speicherung und der Datenzugriff sind ebenso hoch optimiert.

Es motiviert dazu, ein hybrides System aufzubauen. Das System besitzt die Vorteile beider Paradigmen: relationale Datenbank und deklarative Anfragesprache. Cheetah [4] benutzt Daten effizienter mit hohen Skalierbarkeit, Leistungsoptimierung und zentrale Verwaltung der *Campaign* sowie Daten über einen Bereich von Bestandsdatenquellen.

Bei *TURN* Unternehmen sind folgende Herausforderungen der Datenverwaltung betrachtet werden:

- Daten

- Einfach aber starke Anfragesprache.

- Data Mining Applikationen

- Leistungsfähigkeit

Aus dieser Herausforderungen ist ein Data Warehouse auf Hadoop zu bauen, die flexibel und skalierbar ist.

## 3.1 Architektur

Die Architektur des Cheetah bietet seinem Anwendern einen frei direkten Zugriff auf Daten über entweder Web UI oder Kommandozeile (CLI) oder Java code via JDBC Treiber. d.h das System bietet nicht nur dem Benutzern eine einfache und effiziente Anfrageausführung im Data Warehouse sondern auch dem Ad-Hoc MapReduce Programm.

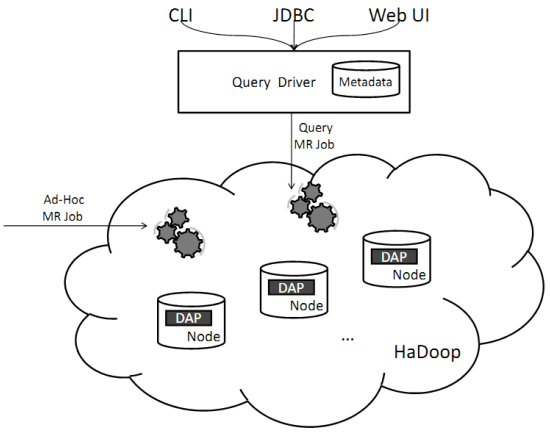


*Abbildung 7 : Cheetahs Architektur*

Eine eingehende Anfrage ist an einem Knote geschickt, auf dem ein Anfragestreiber (*Query Driver*) betreibt. Die Hauptaufgabe des Anfragestreibers ist Übersetzung einer Anfrage in ein MapReduce-Job, welche die beide *Map-Phase Plan* und *Reduce-Phase Plan* erfasst.

Mehrere mögliche Optimierungen sind während Übersetzung-Phase dieser Anfrage betrachtet. z.B es sollte die Anfrage-MapReduce-Job konfiguriert, indem eine geeignete Anzahl von Reducers eingestellt wird. Sind mehrere Benutzer-Abfragen verteilbar, kann eine Batch-Verarbeitung verwendet werden oder wenn eine vordefinierte materialisierte Ansicht mit der Benutzer-Abfragen überein-

stimmt, kann die Ansicht übergenommen werden. Insgesamt jedoch sind diese sehr leichten Optimierungen.

Wie bei der Ausführung der Anfrage, bietet jeder Knoten im Hadoop-Cluster einen primitive Datenzugriff-Schnittstelle (DAP), die wesentlich einem Scanner über virtuelle Sichten entspricht. Die Anfrage MapReduce Job verwendet diesen Scanner, der SPJ-Teil (*Select-Projection-Join*) der Anfrage führt. Die Ad-hoc-MapReduce Job kann ebenso eine sehr ähnliche sogenannte API für eine feinere Datenzugriff sein.

## 3.2 Datenspeicherung

***Speicherungformat***

Bei der Speicherung von tabellarischen Daten existieren bereits viele Methoden z.B Text in CSV Format, serialisiertes Java-Objekt, Zeilenbasiert binäre Array und Spaltenbasiert binäre Array.

Die ersten 3 Methoden, nähmlich Text, serialisiert Java-Objekt und zeilenbasiert binäre Array, entsprechen konzeptuell mit zeilenbasierte Speicherung.

Die letzt vorgestellte Speicherungsmethode "Spaltenbasiert binäre Array" ist ein Hybrid der Zeilen-Spalten Speicherung. d.h *n* Zeilen können auf einer Zelle abgespeichert werden (mit n < N; N ist die maximale Zeilenanzahl. Im Cheetah System ist *N* c.a 200,000 so eingestellt, dass ein Lastverteilung zwischen Komprimierungsverhältnis und Speichernutzung erhält). In der Zelle werden alle Werte der deselben Spalten miteinander unter der sogenannten Spaltengruppe "*Column Set*" zusammengefügt.

***Spaltenkomprimierung***

Da nebenstehende Daten innerhalb einer Spalte sind sehr ähnlich, so bietet die spaltenorientierte Datenbank besonders ein großes Potenzial für die Datenkomprimierung.

Das Speicherformat von einer Zelle hat 2 Teile: Kopf und Datenkörper. Der Kopf erfasst die Schemaversion und die Zeilenanzahl in der Zelle. Der Datenkörper erhält alle Spaltengruppen und Zeiger, die an dem Anfang jeder Spaltengruppe verweisen. Jede Spaltengruppe beginnt mit einem Komprimierungskennzeichen (CT), z.B: *dictionary encoding, run length encoding, default value encoding* oder *no compression*.

Ein Auswahl von solchen Typen für jede Spaltengruppe wurde dynamisch basiered auf die Daten in jeder Zelle ermittelt. Mittlerweile wird die beste Komprimierungsmethode für Spalten im ETL-Phase (*The extract, transform and load phase*) ausgewählt, um vollständige Vorteile von *run length encoding*-Verfahren und beste Komprimierungsverhalten zu bringen.

## 3.3 Anfragesprache

### 3.3.1 Übersicht

Cheetah hat allgemeins 3 hervorgehobenen Merkmale:

- bündige Anfragesprache

- höhe Leistungsfähigkeit

- nahtlose Integration der MapReduce und Data Warehouse.

***SQL-ähliche Anfragesprache***

Momentant bietet Cheetah-Systeme eine SQL-ähnliche Sprache. Diese Anfragesprache ist ziemlich so einfach, dass Benutzer nicht den zugrundeliegenden Schemaentwurf verstehen müssen. Außerdem müssen Benutzer auch nicht jedes Vergleichselement vom Verbund wieder schreiben. In der Praxis können Kunden des *TURN* in der Lage die vordefinierte Cheetah-Anfragesprache ohne vorherige SQL-Kenntnisse schnell erfassen.

Als Beispiel wird die folgende SQL-ähnliche Anfrage betrachtet:

```
SELECT advertiser name,
sum(impression), sum(click), sum(action)
FROM Impressions, Clicks, Actions
DATES [2010 01 01, 2010 06 30]
WHERE site in (‘cnn.com’ ,‘foxnews.com’)
```

Die DATES-Klausel ist explizit unterstützt, damit Benutzer nur interesiertes Zeitinterval über Datum abschätzen können. Die FROM-Klausel enthält eine oder mehr virtuelle Sichten. Wenn es nur eine virtuelle Sicht gibt, ist seine Semantik einfach zu verstehen. Aber es würde zwei oder mehr virtuelle Sichten in der FROM-Klausel existieren, dann ist ihre Semantik daran definieren, dass alle Zeilen sich aus jeder einzelnen virtuellen Sicht zusammen vereinigt werden. Und wenn eine bestimmte Spalte, die in der Anfrage verwiesen ist, nicht in der Sicht vorhanden wäre, dann wird sie als NULL-Wert behandeln.

Um sinnvolle Anfragesemantik zu erreichen, werden alle Gruppen von Spalten benötigt, damit sie in jeder der Sicht in der FROM-Klausel vorhanden sein können. Ebenfalls erlauben sie die Vereinigung der Semantik und die Anfragen, die parallel ausgeführt werden.

Im betrachteten Beispiel gibt es drei Faktentabellen oder virtuelle Sichten, nämlich *Impressions*, *Clicks* und *Actions*.

Das *Impression* ist ein Ereignis, dass ein Benutzer an einer Anzeige geliefert ist, wenn er eine Internet-seite besucht. Das *Click* ist ein Ereignis, dass Benutzer auf die Anzeige klickt. Das *Action* ist eine Überführungsereignis an Inserenten Website, z.B ein Benutzer hat ein Produkt durch eine Anzeige gekauft, die dem Beutzer mindestens einmalig geliefert hat.

In dem obigen Beispiel ist *advertiser_name* ein Dimensionsattribut (durch JOIN-Operation auf der ID Spalte). Alle drei virtuelle Sichten (*Impressions, Clicks, Actions*) enthalten die Spalte *advertiser_name* Der *impressions*-Spalte befindet sich nur in *Impressionen*-Sicht. Der *click*-Spalte existiert auch nur in der *Clicks*-Sicht, während der *action*-Spalte nur im *Actions*-Sicht angesehen ist. Schließlich haben die SELECT-und WHERE-Klauseln die gleiche Semantik wie SQL-Standard.

Als Vereinfachung wird die GROUP BY-Klausel so weggelassen, dass alle *non-aggregate*-Spalten z.B. advertiser_name in der SELECT-Klausel wird als GROUP BY-Spalten behandelt.

***Multi-Block Anfrage****:* Cheetah unterstützt auch CREATE TABLE AS (CTAS) Anweisung. Um eine an Multi-Block nicht zugeordnete Anfrage zu melden, kann der Benutzer zuerst eine Tabelle erzeugen und danach über diese Resultat der Tabelle abfragen.

### 3.3.2 Anfragesausführung

Die Anfragesausführung im Cheetah ist eine Übersetzung der Anfrage in einen Hadoop MapReduce-Job. Allgemein, um einen Hadoop MapReduce-Job anzulegen, sollten Eingabedateien festgelegt werden, die auf dem Hadoop verteilte Dateissystem (HDFS) abgespeichert sind.

***Eingabedateien***

Die an dem MapReduce-Job zufürende Eingabedateien sind immer Faktentabellen. Hadoop Framework wird Map-Tasks angesetzt, um die Faktentabelle parallel zu scannen. Es ist nicht notwendigt, Dimensionstabellen als Eingabesdaten anzugeben. Die Dimensionstabellen werden automatisch durch die Map-Phase abgeholt. In der *TURNs* Anwendung wurden die Faktentabellen durch Datum partitioniert. Durch DATE-Klausel können die verbundenen Partitionen gefunden werden.

***Map Phase Plan***

Jeder Knoten speichert im Cluster einige Portionen der Faktentabelle-Datenblöcke und kleine Dimensiondateien (*Abbildung **8***). Die Map-Phase der Anfrage enthält zwei Operatoren: *Scanner* und

*Aggregation*. Nach außen hat der Scanner-Operator eine Schnittstelle, die ähnlich einen SELECT gefolgt vom PROJECT-Operator über die virtuelle Sicht sind. d.h eine Filterbedingung auf die Datenzeilen bei Eingabe und eine interessierte Liste der Spalten bei Ausgabe. Hier kann eine Spalte einfach wie eine Anweisung ausdrücken. Die ausgegebenen Tupeln haben einen Iterator-Schnittstelle, so dass sie bei einem *getNext*-Funktion abgerufen werden können. Für eine korrekte Zugriffskontrolle sind eine Benutzerkennung und ein Passwort erforderlich

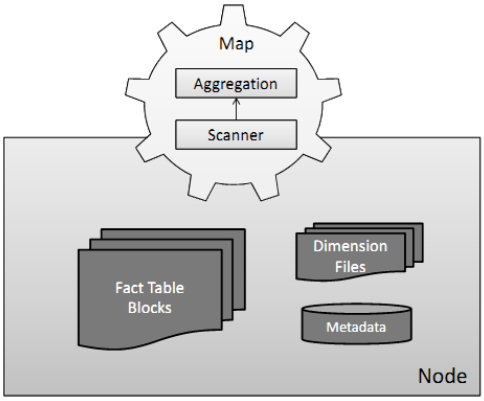


*Abbildung 8 : Map Phase Plan*

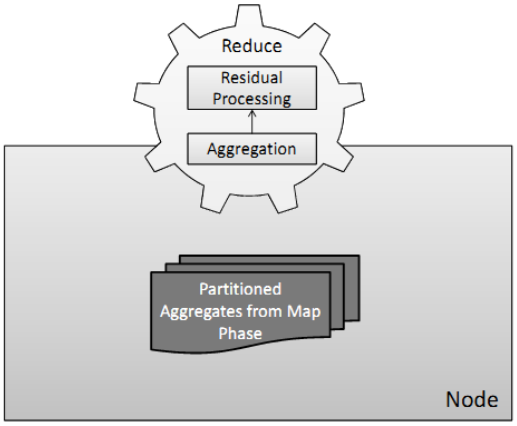


*Abbildung 9 : Reduce Phase Plan*

Innerlich übersetzt der Scanner-Operator die Anforderung an eine gleichwertige SPJ Abfrage, die Attribute auf den Dimenstiontabellen zu holen ist, aber nur die in der Abfrage genannten Dimensiontabellen zugegriffen und miteinander verbunden werden müssen. Hier wird ein Multiway Join-Algorithmus genutzt, da die verbundenen Dimensiontabellen kleine sind (große Dimensionstabellen sind denormalisierte und Verbund sind damit nicht erforderlich). Der Idee des Multiway Join Algorithmus ist, wenn mehrere kleine Tabellen mit einem großen Tabelle verbunden werden oder die Eingabetabellen bereits auf dem Verbundspalte partioniert werden, können die kleine Tabellen in die Hauptspeicher aufgeladen werden. Mehrere Hash-Lookups werden während eines Durchsuchens einer anderen großen Tabelle bei Map-Phase gleichzeitig durchgeführt.

Bei Optimierung werden diese Dimensionen in *in-memory* Hash-Tabellen nur einmal geladen, wenn andere Map-Tasks sich die gleichen JVM teilen.

Neben der Selektion untersucht der Scanner auch eine Optimierung, wenn er erstmal die erforderliche Spalten in der Filterbedingung empfängt. Wenn die Bedingung nicht erfüllt ist, dann kann es das Abrufen der restlichen Spalten überspringen. Diese Optimierung ist sehr nützlich, wenn die Filterbedingung selektiv ist. Die Aggregation-Operator ist für die lokale Aggregation des aktuellen Blocks durchgeführt, wenn die Aggregatfunktion algebraischen ist. Standardmäßig wird eine Hash-basierte Implementierung von GROUP BY-Operator verwendet.

***Reduce Phase Plan***

Die Reduce-Phase der Abfrage ist einfach und direkt wie in *Abbildung **9*** zeigt. Es führt zunächst globale Aggregation über die Ergebnisse aus Map-Phase. Dann bewertet alle verbleibenden Ausdrücke über die aggregierten Werte und/oder der HAVING-Klausel.

Wenn ORDER BY-Spalten eine Gruppe von Spalten sind, würden sie bereits durch Hadoop Framework in der Reduce-Phase sortiert. Diese Eigenschaft kann eingesetzt werden. Wenn die ORDER BY-Spalten die aggregierte Spalten sind, werden die Ergebnisse in den einzelnen Reduce-Task sortiert und als Endergebnisse zusammengeführt, nachdem MapReduce-Job abgeschlossen ist. Bei Verwendung des LIMIT by n-Klausel ist es ausreichend, jeder Reduce-Task nur *n* ersten Zeilen für die letztliche Merge auszugeben.

## 3.4 Optimierung

Um Anfragenausführung effektiv zu verwenden, sind einige einzigartige Optimierungsmöglichkeiten vorgestellt.

***MapReduce Job Konfiguration***

Für vorgegebene Hadoop-MapReduce-Job und Eigabedateien für den Job können die Anzahl von Map-Tasks durch Hadoop Framework bestimmt werden, das auf die Anzahl der Eingabedateien und Blocke per Datei basiert. Die Anzahl der Reduce-Tasks muss jedoch durch den Job selbst versorgt werden und hat einen großen Einfluss auf die Performance. Wenn die Ausgabe der Abfrage klein ist, dominiert das Map-Phase die Gesamtkosten. Hätten eine große Anzahl von Reducers, können ein Verlust von Ressourcen entstehen und die Abfragen verlangsamen.

Einige einfache Heuristiken können zum automatischen Konfiguration der Anzahl von Reducers für die Anfrage-Job verwendet sein. Erstens ist die Anzahl der Reducers proportional zur Anzahl der Gruppe von Spalten in der Anfrage. Zweitens, wenn die Gruppe nach Spalte einige Spalten enthält, die mit sehr großen Kardinalität haben, z. B. URL, die Anzahl der Reducers sind einfach erhöht.

***MultiQuery Optimierung***

Cheetah erlaubt es , den Nutzern gleichzeitig mehrere Anfragen vorzulegen. Die Anfragen werden in einem einzelnen Batch durchgeführt, solange diese Abfragen die gleichen *FROM* und *DATE*- Klauseln haben. Die Analyse für die solche Anfrageplannung ist automatisch von dem System durchgeführt. Um diese kombinierte Abfragen ausführen, die Map- Phase hat jetzt einen gemeinsamen Scanner, die den Scanvorgang von der Faktentabellen verteilt und verbindet mit der Dimension-tabellen

Der Scanner legt eine Anfrage-ID an jeden ausgegebenen Zeile. Die Anfrage-ID gibt dafür an, welche Anfrage die Zeile qualifiziert. Diese *ID* wird auch als eine zusätzliche Gruppe von Spalten verwendet werden. Die Ausgabe von unterschiedlichen aggregierten Operatoren werden in einer einzigen ausgegebenen Stream zusammengeführt. Außerdem spielt das Anfrage-ID eine wichtige Rolle, da Reduce-Phase sich nicht zwischen der Zeilen in verschiedenen Anfragen unterscheiden kann.

Die Reduce-Phase wird zunächst die Eingabezeilen basierend auf ihre Anfrage-IDs zerlegt und schicken sie dann zu den entsprechenden Anfragen-Operatoren.

***Nutzung der materialisierte Sichten***

Die Nutzung materialisierter Sichten ist eine häufige Technik in Data Warehousing. Hier wird es dazu gezeigt, wie die Technik basierend auf virtuelle Sicht verwendet ist.

*a. Definition von materialisierte Sichten*

Zur einfachen Beschreibung der virtualen Sicht als Ergebnis wird eine materialisierte Sicht verwendet. In Cheetah-System beinhalt jede definierte materialisierte Sicht nur die Spalten in Faktentabelle. d.h sie umfassen nicht die Spalten in Dimensiontabellen. Die materialisierte Sichten werden ebenso nach Datum partioniert. Deswegen wird eine materialisierte Sicht jeden Tag erzeugt.

*b. Sicht-Matching und Anfrageumschreibung*

Um materialisierte Sichten zu verwenden, sollten zwei Probleme zugestellt werden: Sicht-Matching und Anfrageumschreibung. Zur Übereinstimmung einer Anfrage müssen die folgende Bedingungen für eine materialisierte Sicht erfüllt sein:

- Die Anfrage muss die virtuelle Sicht verweisen, die auf dieselbe Faktentabelle entspricht und die virtuelle Sicht definiert auf der materialisierten Sicht.
- Die *non-aggregate* Spalten, die in SELECT und WHERE-Klauseln der Anfrage verwiesen sind, muss eine Teilmenge der Gruppe nach Spalten der materialisierten Sicht sein.
- Die aggregierte Spalten müssen von der aggegierten Spalten der materialisierten Sicht berechenbar sein.

Beispiel:

Die *pub* materialisierte Sicht ist als folgende Sicht defniert:

```
CREATE MATERIALIZED VIEW pub AS
SELECT publisher_id,
sum(impression) impression
FROM Impressions
```

- Die übereinstimmte Anfrage:

```
SELECT publisher_name,
sum(impression), sum(click),
sum(action)
FROM Impressions, Clicks, Actions
DATE [2010_01_01, 2010_06_30]
```

- Anfrageumschreibung:

```
SELECT publisher_name,
sum(impression), sum(click),
sum(action)
FROM pub, Clicks, Actions
DATE [2010_01_01, 2010_06_30]
```

***Optimierung der geringen Latenz-Anfragen***

Die aktuelle Hadoop-Implementierung hat selbst einige nicht-triviale Overhead, die z.B Job-Startzeit, JVM Startzeit, usw enthält. Für kleine Abfragen wird dies zu einem beträchtlichen zusätzlichen Aufwand geworden kann. Um dieses Problem zu lösen, wird eine Erkennung von Eingabedateisgröße durch Query-Driver während des Übersetzungphase von Anfrage eingeführt, besonders wenn die Größe der Eingabedatei klein ist. z.B würde eine materialisierte Sicht eintreten, kann die Sicht ausgewählt werden, um die Datei direkt aus dem HDFS zu lesen und danach die Anfrage lokal auszuführen.

Dies vermeidet Aufruf eines MapReduce Job. Auf diese Weise können wir die Reaktionsfähigkeit mit geringen Latenz-Anfragen verbessern sowie eine Reduzierung der Cluster-Auslastung.

## 3.5 Integration

Cheetah bietet verschiedene Möglichkeiten, Anwenderprogramms zu verbinden:

Erstens hat es eine JDBC-Schnittstelle, so dass Anwenderprogramm eine Anfrage anlegen kann und durch die ausgegebene Ergebnisse iteriert.

Zweitens, wenn die Ergebnisse der Abfrage zu groß für ein einziges Programm zu verbrauchen sind, können Benutzer eine MapReduce-Job schreiben, um die auf HDFS gespeicherte Abfrageausgabes-dateien zu analysieren.

Schließlich bietet Cheetah eine *non-SQL*-Schnittstelle, die einfach in jeden Ad-hoc-MapReduce Job integriert werden können, um die Daten am besten Granularität zuzugreifen. Um dies zu erreichen, muss die Ad-hoc-MapReduce Programm zuerst die Eingabedateien festlegen, die einen oder mehreren der Faktentabelle-Partitionen können sein. Im Map-Programm muss es einen Scanner beinhalten, um individelle Datensätzen in Datenbank zu zugreifen. Danach hat der Anwender vollständige Freiheit zu entscheiden, was mit den Daten zu tun.

Es gibt mehrere Vorteile mit der non-SQL-Schnittstelle für externe Anwendungen:

- Zugriff auf lokale Daten zuzugreifen, d.h. es gibt kein Datenstranfer über verschiedene Systeme oder sogar verschiedene Knoten.
- Die virtuelle auf Sicht basierte Scanner versteckt meisten die Schema-Komplexität von Map-Reduce Entwickler d.h. sie brauchen nicht zu beachten, wie die Spalten abzurufen, was und wo die Dimensiontabellen sind, wie sie sich verbinden.
- Gute komprimierte Daten und optimierte Zugriffsmethode.

# 4 ZUSAMMENFASSUNG

Hier wurden zwei verschiedene große skalierbare Analysetechniken betrachtet. Es gibt darüber noch zahlreiche andere Techniken, die in [7] vorgestellt sind, entweder Opensource oder Enterprise zur Verfügung. Einsatz und Anwendungsfall muss abhängig von den jeweiligen Anforderung evaluiert werden, auf welches System zurückgegriffen werden soll.

Google Dremel eignet sich als ein leistungsstarkes Analyse-Werkzeug für nicht relationale bzw. verschachtelte Daten. Dremel bringt Datenanalyse auf Hochgeschwindigkeit im Vergleich mit MapReduce-Framework. Es kann auf *„in situ“* veschachtelten Daten betreiben. Die *„in situ“* bezieht sich auf die Fähigkeit, die Daten an direkter Stelle zuzugreifen. z.B in einem verteilten System wie GFS oder ein anderes Speicherschicht wie BigTable. Dremel versucht, dass durch Aufbau eines ganz anderen Systems, das interaktive Abfragen beschleunigt überwindet. Der Gedanke hinter der Dremelentwicklung ist eine Verbesserung bei analysicher Datenverarbeitung, weil der native MapReduce zu viel Latenz für eine schnelle interaktive Anfrage oder Analyse hat. Aber ist MapReduce-Ausführung tätsachlich nicht langsam. Interessanterweise kann Aster Datas SQL-Map-Reduce [8] auch eine sehr schnelle interaktive Anfrage bieten.

Google hat derzeit BigQuery veröffentlicht, *a web service that enables you to do interactive analysis of massively large datasets*, das auf dem Dremel basiert. Es wäre gut am Anfang mit Dremel, einen Blick auf **BigQuery**[9] zu nehmen, um mehr über die Fähigkeiten von Dremel kennenzulernen.

Cheetah bietet ein Hochleistungssystem für Data Warehouse, das auf dem MapReduce läuft. Es kann effizient, machmal in Sekunden, irgendein Anteil des Petabyte gespeicherten Daten verarbeiten. **Hive** hat die ähnliche Arbeit wie Cheetah. Statt des Aufbaus eines generischen Data Warehouse Systems als Hive, ist Cheetah speziell für eigene Anwendungen bei Turn entwickelt. Deswegen ist Cheetah nicht geeignet für alle Fälle bzw. Wiedervewendung. Trotzdem können viele Ideen und sogar einige der Anpassungen geteilt werden, um eine allgemeine Data Warehouse System zu erstellen.

## Literaturverzeichnis

[1] *Spaltenorientierte Datenbank*, 2012 http://de.wikipedia.org/wiki/Spaltenorientierte_Datenbank, besucht: 30. Januar 2012, Online.

[3] S.Melnik et al. *Dremel: interactive analysis of web-scale datasets.* Proceedings of the VLDB, 2010

[4] S.Chen. *Cheetah: a high performance, custom data warehouse on top of MapReduce*. Proceedings of the VLDB, 2010

[5] S.Abiteboul, R.Hull, and V.Vianu. *Foundations of Databases*. Addison Wesley, 1995

[6] L.S.Colby. *A Recursive Algebra and Query Optimization for Nested Relations*. SIGMOD Rec., 18(2), 1989.

[7] S.Edlich, A.Friedland, J.Hampe, B.Brauer NoSQL – *Einstieg in die Welt nicht relationaler Web 2.0 Datenbanken* . Hanser Fachbuchverlag, München, 1.Auflage, 2010

[8] Rick F. Van der Lans. *Using SQL-MapReduce for Advanced Analytical Queries*. A Technical Whitepaper, 2. Edition. September 20, 2011 bei Teradata

[9] Google BigQuery , 2012 https://developers.google.com/bigquery/ , besucht: 30. Januar 2012, Online