Technische Universität Berlin
Fakultät IV

Lehrveranstaltung

„Large Scale Data Mining mit Apache Mahout “

Dozent: Isabel Drost

„NoSQL Datenbanken am Beispiel von HBase“

Daniel Georg, Matr.-Nr.: 305599

Berlin, den 03.02.2010

Inhaltsverzeichnis

1. Motivation

Heutzutage ist die Menge von Daten drastisch gestiegen. Eine große Rolle in der modernen Welt spielen deshalb Technologien, welche eine effiziente Speicherung und Verarbeitung von großen Datenmengen ermöglichen.

Das System soll in der Lage sein gleichzeitig mit mehreren tausenden Anfragen umzugehen und die richtigen Informationen innerhalb von Millisekunden in den Datenbanken entweder zu speichern oder an den Nutzer zurückzugeben.

Ein gutes Beispiel dafür sind die Sozialen Netzwerke oder E-Mail Dienste bei denen hunderte von Gigabyte täglich erstellt werden müssen. Dabei sind schreibende Zugriffe mindestens ebenso wichtig wie lesende.

Das größte Problem beim den traditionellen relationalen Datenbanken ist, dass die Performance bei high-load Projekten sehr niedrig ist. Es gibt bei RDBMS genau drei Problembereiche. Der erste Bereich ist die horizontale Skalierung mit großen Datenmengen. Zum Beispiel Facebook mit 50 Terabyte allein zum suchen von eingehenden Nachrichten oder Ebay mit einer Größe von 2 Petabytes insgesamt. Der zweite Punkt ist die Leistungsfähigkeit der einzelnen Server. Der dritte Punkt ist, dass die logische Gestaltung von solchen Systemen unflexibel ist.

Alle diese Probleme haben dazu geführt, dass viele der größten Organisationen gezwungen wurden, neue Lösungsmöglichkeiten zur Speicherung und Skalierung von riesigen Datenmengen zu suchen.

Zahlreiche Projekte sind sehr fortgeschritten, beispielsweise das BigTable Speichersystem, das Google selbst entwickelte und bei sich verwendet. Es ist nötig um die riesigen Mengen an Daten zu verwalten, wie sie unter anderem bei Google Maps, Google Earth, Blogger oder auch bei Youtube anfallen. Es handelt sich nicht um eine relationale Datenbank, sondern um ein Speichersystem, dass es erlaubt Entities oder Datensätze abzulegen. Eine genauere Beschreibung von Bigtable befindet sich im dritten Abschnitt. Aufgrund dessen das BigTable ein geschlossenes System ist, wurden auch verschiedene Open-Source Implementierungen auf den Markt gebracht. Ein ähnliches System ist HBase. Eine etwas andere Systemarchitektur besitzt z.B. Yahoo`s PNUTS. Das System wurde speziell für Web Applikationen entwickelt. In der Zeitlinienkonsistenz eine sehr wichtige Rolle spielt.

In dieser Arbeit möchte ich die wichtigsten Ideen dieser Systeme darstellen.

2. Was ist NoSQL?

NoSql ist eine Bezeichnung für nicht relationale Datenbanken. Die NoSql Bewegung ist erst im Jahr 2009 gestartet und definiert sich als eine Open-Source Community. Diese Bezeichnung wurde von Eric Evans erfunden, als Johan Oskarsson aus Last.fm eine Veranstaltung über Open-source verteilte Datenbanksysteme organisierte.

„Oft wird vermutet, dass sich hinter dem Ausdruck ein "No to SQL at all" verbirgt. Vielmehr ist NoSQL als "Not only SQL" zu verstehen. Im Allgemeinen ist bekannt das unterschiedliche Probleme differenzierter Lösungsstrategien bedürfen. Daher ist es nur konsequent, dass eine breite Menge von unterschiedlichen "Not only SQL"-Systemen existieren. Das wiederum unterstreicht den Innovationsgeist der Open-Source Community. Ebenfalls ist festzustellen, dass traditionelle Datenbanksysteme (RDBMS) nicht nur heute, sondern auch in Zukunft eine hohe Bedeutung haben werden.“[1]

Die alternativen Speichersysteme der NoSql-Bewegung haben bewusst auf einige Vorteile moderner RDBMS verzichtet. Dadurch haben die NoSql Datenbanken eine bessere Performance erreicht. Zum Beispiel wird die horizontale Skalierbarkeit unter anderem dadurch ermöglicht, dass weitgehend auf Joins verzichtet wird. Die Community bietet verschiede Lösungen, dass sind beispielsweise Casandra, CouchDD, HBase, Hypertable.

---

[1] *Matthias Wessendorf und Bartosch Warzecha*
http://it-republik.de/jaxenter/artikel/NoSQL-%96-Schoene-neue-Welt-2710.html

3. BigTable von Google

Weil Hbase eine freie Implementierung von BigTable ist, ist es sehr sinnvoll zuerst die Struktur von Bigtable kennen zu lernen und danach auf spezielle Besonderheiten von Hbase im unterschied zu BigTable einzugehen.

BigTable ist ein verteiltes Datenbanksystem zur Speicherung und Verwaltung riesiger Mengen schwach strukturierter Daten. Es ist eine in sich geschlossene propritäre Entwicklung von Google. Es erlaubt Petabytes auf tausenden Servern zu speichern. Bei Google existieren mehr als 500 Exemplare von BigTable. Das größte Exemplar hat mehr als 3000 Maschinen, in dehnen mehr als 6 Petabyte gespeichert werden können. Einige Instanzen verarbeiten rund um die Uhr mehr als 500 000 Anfragen pro Sekunde. Das System wird auf mehr als sechzehn Produkten von Google verwendet. Unter anderem sind dass; Google Analytics, Google Finance und Orkut.

Bei der Entwicklung wurden sehr große Akzente auf folgende Charakteristiken gesetzt z.B. vielseitige Anwendbarkeit, Skalierbarkeit, höhere Performance und die verbesserte Zuverlässigkeit. An machen Stellen sieht BigTable wie eine Relationale Datenbank aus, dieser Eindruckt trügt jedoch. Es werden ähnliche Strategien zur Realisierung ausgewählt aber es existieren prinzipielle Unterschiede zwischen BigTable und RDBMS. Im folgenden werden diese Unterschiede näher beleuchtet.

3.1 Datenmodell

„A Bigtable is a sparse, distributed, persistent multidimensional sorted map“. Wie aus der offiziellen Dokumentation folgt, wird bei Bigtable im Unterschied zu den klassischen Datenbanken eine dünn besetzte, verteilte, sortierte und mehrdimensionale Abbildung verwendet. Jeder Eintrag in der Abbildung wird als row key, column key, und timestamp ; **(row:string, column:string, time:int64) → string** indiziert. Ein Gespeicherter Wert in der Tabelle als Bytearray repräsentiert und nicht von System Interpretiert. Solche Abbildungen kann man sich als Tabellen vorstellen, in der jeder Zeile und Spalte einen individuellen Schlüssel besitzt. In den Zellen dieser Tabelle werden die Werte gespeichert, wobei jeder Wert mehrere Versionen haben kann und jede Version einen eigenen Zeitstempel hat. Anderes ausgedrückt hat die Tabelle verschiedene Zeitschichten (Timelayers). Im folgendem Beispiel (Abb.1) wird diese Struktur graphisch dargestellt. Beispielsweise können Webseiten in dieser Tabelle

abgelegt werden. Die Schlüssel in den Tabellen sind die URL's der Seiten und in den Zellen die Versionen des Inhaltes, welche zu verschiedenen Zeitpunkten gespeichert werden. In den anderen Spalten werden die Zeiger auf dieser Seite gespeichert.

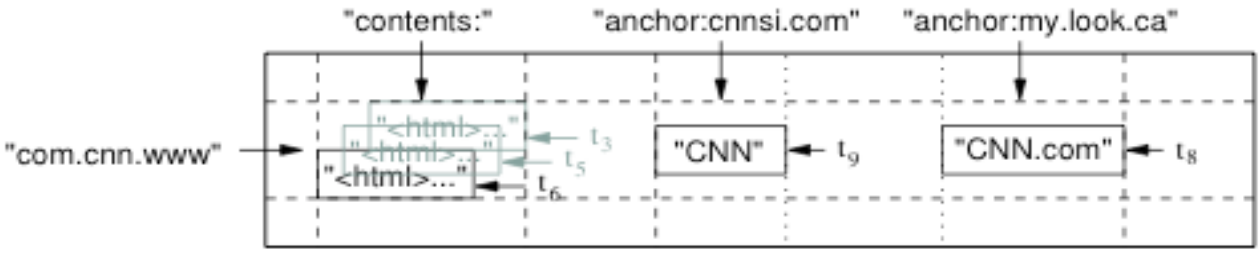

Abbildung 1: Ein Ausschnitt einer Beispieltabelle zum Speichern von Webseiten. Die Zeilennamen bestehen aus der invertierten URL´s. Die Contentspaltenfamilie enthält die Inhalte der Webseite. Die Anchorspaltenfamilie speichert den Text der Referenz der Seite. CNN's Homepage wird von Sport Illustrated und MY-look referenziert, somit erhält die Zeile die Spalten mit den Namen anchor:cnnsi.com und anchor:my.look.ca. Jede anchor Zelle hat einen Zeitstempel; Die contents-Spalte hat drei verschiedene Versionen mit den Zeitstempeln t3, t5 und t6. [1]

Eine andere Besonderheit dieser Tabelle ist, dass in den verschieden Zeilen verschiedene Gruppen von Spalten ausgefüllt werden. Für nicht existierende Werte werden keine NULL Werte gespeichert welche bei den relationalen Datenbanken Platz im Anspruch nehmen würden. Für eine Anfrage auf nicht existierende Daten wird ein leeres Resultat geliefert.

BigTable speichert in sich eine Reihe von den Benutzern erzeugenden Tabellen. Zeilen in den Tabellen sind nach dem Schlüssel in der lexikographischen Ordnung sortiert. Der Schlüsselbereich wird dynamisch in mehrere Teile geteilt. Diese werden als Tablet benannt. Die Tabletverteilung für die Datendistribution auf mehrere Maschinen des Clusters übernimmt der Masterserver. Bei Ersterzeugung besteht die Tabelle aus einem Tablet. Bei Vergrößerung der Datenmenge wird die Tabelle automatisch auf Tablet's aufgeteilt, auf diese Weise ist jedes Tablet etwa 100-200 Megabyte groß. Daraus folgt, dass Leseoperationen auf kleiner Anzahl von Zeilen effizienter und mit weniger Kommunikationsaufwand durchgeführt werden können.

Spalten-Schlüssel sind ebenfalls in Gruppen organisiert und werden Spaltenfamilien (colum-family) genannt. Der Schlüssel der Spalte wird wie folgt als **family:qulifier** gebildet, wobei family – ein Name der Familie und qualifier – ein einzigartiger Name der Spalte innerhalb dieser Familie ist. Bevor die Daten in der Tabelle gespeichert werden, sollen erstmalig benutzte Spaltenfamilien in der Tabelle definiert werden, dann ist es erlaubt beliebige Namen im Rahmen von Spaltenfamilien

für die Spalten auszuwählen.

In jeder Tabellenzelle können mehrere Versionen von Daten mit verschiedenen Zeitstempeln gespeichert werden. Diese Markierungen können von Programm des Clients erzeugt und zugewiesen werden, oder von BigTable generiert werden. Werden die Zeitstempel vom Benutzer selbst erzeugt, muss die Anwendung dafür sorgen, dass sie einzigartige Zeitstempel erzeugt um Kollisionen zu vermeiden. Zellenversionen werden nach Zeitmarkierungen in absteigender Reihenfolge gespeichert, das heißt dass die neusten Einträge als erste gelesen werden. BigTable unterstützt die automatische Entfernung von veralteten Daten.

Im Unterschied zum RDBMS unterstützt BigTable keine SQL Sprache. Für Zugriffe auf die Daten wird eine Bibliothek benötigt die eine einfache Schnittstelle bereitstellt. BigTable API stellt Funktionen zum generieren und terminieren von Tabellen, Metadaten und Spaltenfamilien. Um Iterationen über ganze Bereiche zu ermöglichen, werden hier verschiede Mechanismen zur Limitation der Zeilen, Spalten und Zeitstempel bereitgestellt.

3.2 Architektur und Implementierung von BigTable

BigTable-Cluster bestehen aus dem Master-Server und einer Menge von Tablet-Servern, welche automatisch zu dem Cluster hinzugefügt oder entfernt werden können. Die Aufgaben von dem Master-Server sind, Verteilungen von Tablets auf die Tablet-Server, Überwachung von Tablet-Servern, Load-Balancing zwischen Servern und Entfernung von veralteten Daten. Ansonsten steuert der Master-Server die Erzeugung und Änderungen von Tabellen sowie der Family-Columns.

Jeder Tablet-Server ermöglicht den Zugriff auf bestimmte Tablets. Der Server verarbeitet die Lese-Schreibanfragen an die betreuten Tablets sowie die Teilung von Tablets deren Größe sich maximiert hat. Die Daten werden direkt zwischen Client und Tablet-Server ausgetauscht, dadurch wird der Traffic durch den Master-Server minimiert, was eine geringere Auslastung für denselben zur Folge hat.

Zur Speicherung des Standortes der Tablets wird eine dreistufige Hierarchie verwendet. (Abb.2). Die erste Schichte besteht aus einer Datei in der die Adresse des Root-table gespeichert ist. Die Datei wird auf dem äußerst zuverlässigen Chubby-Service gelagert. In dem Root-tablet werden die Verweise auf die Tablets in der

Systemtabelle als METADATA-Tabelle gespeichert. In den METADATA-Tabellen werden Verweise auf alle Tablets der Benutzertabellen gespeichert. Das Root-tablet ist also das erste Tablet in der Metadaten-Tabelle und besitzt deshalb eine gesonderte Eigenschaft. Es wird nie aufgeteilt, um sicherzustellen, dass es nicht mehr als drei Levels in der Hierarchie werden. Information über Standorte der Tablets werden auf den Client gecachet, dadurch wird die Anzahl von Anfragen auf die METADATA-Tabellen verringert. Passiert es dass ein Client den Standort eines Tablets nicht erkennt, wird dieser in der Hierarchie rekursiv aufwärts gesucht.

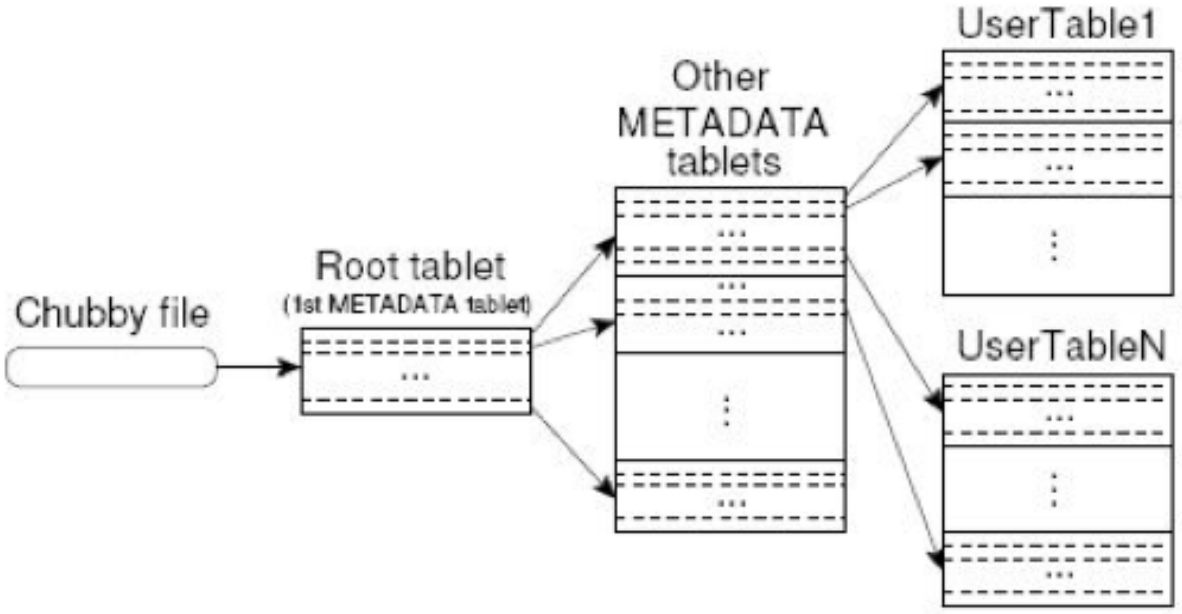

Abbildung2: Dreistufige Hierarchie [1]

Jedes Tablet wird zu jeder Zeit genau von einem Tablet-Server bedient. Der Hauptserver beobachtet den Status von den Tablet-Servern und den aktuellen Standort des Tablets. Kann ein Tablet nicht zugeordnet werden, wird ein neuer Tabletserver mit dem frei verfügbarem Speicherplatz des Hauptservers gefunden und es wird eine Ladeanweisung von dem Master-Server an den Tablet-Server geschickt.

Um den Zustand des Tablet-Servers zu überwachen, wird wie bereits erwähnt ein Chubby-Service verwendet. Wird ein neuer Server gestartet, so erzeugt er eine Datei und erhält eine Sperre in der Chubby-Bibliothek für diese. Der Master-Server erkennt die Erzeugung von dem neuen Tablet-Server durch die erzeugende Datei in diesem Verzeichnis. Wenn ein Tablet-Server die Sperre verliert, wird er gestoppt. Falls der

Server gezielt von dem System getrennt wird, gibt er die Sperre frei und der Master sucht nach einem Ersatz für diesen. Der Hautpserver fragt periodisch jeden Server nach dem Stand seiner Sperre ab. Wenn der Server die Sperre verliert oder nicht erreichbar ist, dann probiert der Hauptserver eine Sperre auf die Dateien des Servers zu legen. Wenn der Hauptserver die Sperre bekommen hat, löscht er in diesem Fall die Datei des Servers. So garantiert er, dass dieser Server von keinem Benutzer belegt ist. Danach verschiebt der Hauptserver die Tablets des Servers in eine Liste von nicht zugewiesenen Tablets. Diese werden dann später neu verteilt. Falls es keine Verbindung zu dem Cubby Service gibt, eliminiert sich der Hauptserver selbständig, denn es ist nun keine zuverlässige Arbeit mehr möglich. Das verwendete Steuerungssystem überwacht und startet automatisch den Hauptserver. Beim Booten führt der Hauptserver folgende Aktionen aus: der Server kriegt eine Sperre auf die Datei im Cubby-Service, welche speziell für den Hauptserver des BigTables zugewiesen ist. So wird garantiert dass im System nur ein Hauptserver existiert. Danach scannt er die Servereinträge in dem Verzeichnis des Chubby-Service, um alle verfügbaren Server zu finden. Im nächsten Schritt, kommuniziert er mit jedem Server um zu erfahren welche Tablets bereits zugewiesen sind. Wenn kein zugewiesenes Tablet gefunden wird, trägt der Server dieses Tablet in eine Liste von nicht zugewiesenen Tablets ein. Danach scannt er die MATADA-Tabelle, um alle Informationen über die Tabellen zu erfahren.

Die Menge von Tablets kann sich ändern, wenn eine Tabelle gelöscht oder hinzugefügt wird sowie zwei Tablets zusammengefasst oder ein Tablet aufgeteilt wird. Für alle diese Änderungen außer der Tabletaufteilung ist der Hauptserver zuständig. Das Aufteilen von einem Tabelt wird nur von Ihm vollzogen. Der Tabletserver bestätigt die Teilung, in dem er neue Informationen in die METADA-Tabelle einträgt und benachrichtigt den Masterserver.

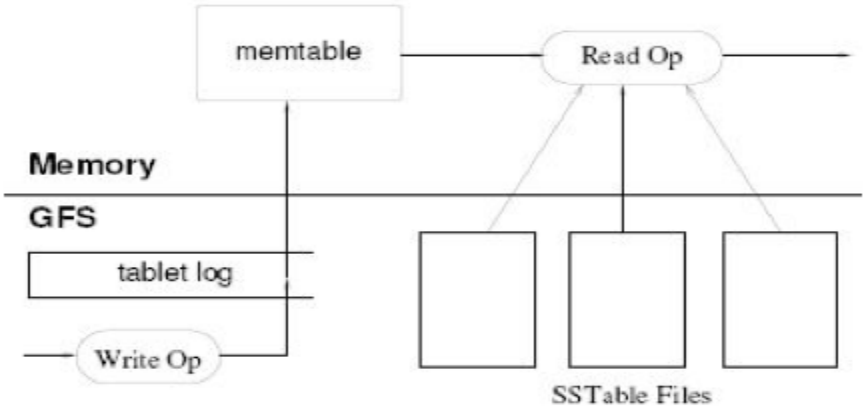

Abbildung 3. Tablet Darstellung[1]

Die Tablets werden in Form einer Datei auf das verteilte Filesystem GFS abgelegt (Abb.3). Zur Datenspeicherung von BigTable wird das spezielle Format SSTable verwenden. Alle Änderungen werden in einem Log-File gespeichert. Die kürzlich ausgeführten Änderungen werden im Arbeitsspeicher abgelegt und in einem gesonderten Puffer gespeichert. Dieser Puffer wird als mentable benannt. Alle älteren Änderungen werden in den verschiedenen SSTables gespeichert. Die Leseoperationen werden durch Zusammenführung von SSTable Dateien und memcache bedient. Bei der Überschreitung der zulässigen Größe, wird der Inhalt von memtable in das SSTable Format konvertiert und auf das GFS gespeichert. Diese Prozedur heißt minor compaction. Eine große Anzahl von SSTable Dateien verringert die Leistungsfähigkeit und vergrößert die Menge gespeicherter Daten. Deswegen wird in regelmäßigen Abständen eine Vermischung namens margin compaction durchgeführt. In diesem Verfahren wird aus mehreren SSTable und dem Inhalt von memtable, eine SSTable Datei generiert. Eine Vermischung in dem alle SSTables des Tablets vermischt werden, wird als mahjor compaction benannt. Nach dem mahjor compactionprozess bleiben keine gelöschte Einträge in dem SSTable. Beim Booten des Tablets, liest der Tablet-Server in den Hauptspeicher die Indizes des SSTable, sowie der Logdatei mit den letzten Änderungen, danach stellt der Server den Inhalt des memcaches durch die Anwendung aller Änderungen die seit der letzten minor compaction gemacht wurden wieder her .

Um die Zuverlässigkeit, Geschwindigkeit und Verfügbarkeit zu erhöhen, wird im BigTable eine Reihe von zusätzlichen Mechanismen verwendet.

Clients können mehrere Spaltenfamilien in so genannte locality groups zusammenfassen. Für jede Gruppe wird in jedem Tablet eine getrennte Struktur erzeugt. Das Einteilen von Spaltenfamilien, welche nicht in einer locality group zusammengefasst sind, erlaubt es die Effizienz von Leseoperationen zu erhöhen. Außerdem man kann bei den Gruppen eine Reihe von zusätzlichen Parametern eingeben. Beispielsweise kann eine Gruppe so deklariert werden, dass sie immer im Hauptspeicher gelagert wird. In diesem Fall werde die Daten die zu dieser Gruppe gehören im Hauptspeicher des Tablet-Servers gecacht.

BigTable unterstützt auch die Datenkomprimierung. Das Komprimieren der Daten führt dazu, dass weniger Speicherplatz benötigt wird.

Um die Performance von Leseoperationen zu erhöhen wird in den Tablet-Servern ein Zwei-Level-Cache-System verwendet.

Beim Lesen wird es benötigt alle SSTable-Dateien anzusprechen. Um die

Anzahl von aufwändigen Festplattenzugriffen zu minimieren, erlaubt BigTable-Bloom Filter zu definieren. Anwendungen von Bloom Filtern erlaubten es mit einem Zugriff auf die Festplatten zu fragen, ob sich die Daten im SSTable befinden.

4. HBase

Das verteilte Datenbanksystem HBase gehört zu einer OpenSource Plattform. Hadoop ist nützlich für die verteilten Berechnungen von OpenSource Plattformen. HBase ist ein allgemein ein Analogon zum oben genannten BigTable-System. Das System arbeitet mit dem verteiltem Dateisystem HDFS. Das Datenmodell sowie Architektur und die Realisation von Hbase sind der von BigTable sehr ähnlich. Ich werde in diesem Abschnitt nur über spezifische Charakteristiken und Unterschiede zu BigTable eingehen.

HBase verwendet eine etwas andere Terminologie als BigTable. So wird das Analog zu dem Tablet als Region bezeichnet. Server die Regionen betreuen werden RegionServer genannt und innerhalb dieser Regionen werden Daten auf die vertikale Familie von Spalten aufgeteilt. Jede Familie innerhalb dieser Regions wird in einer separaten Datenstruktur gespeichert und Store genannt. Das Store ist wieder ein Analogon zum SSTable im BigTable. Der Inhalt von Store wird in mehreren Dateien auf dem DHFS abgelegt und die heißen StoreFiles. Die letzte Änderungen im MemStore werden vom RegionServer gecacht und periodisch in die neue Store-Dateien gespeichert. Äquivalent zur Verdichtung von BigTable werden hier auch die Daten vermischt.

Der MasterServer von HBase steuert Zuweisungen von Regionen zu Servern und überwacht deren Status. Im Gegensatz zu Bigtable verwendet HBase zur koordinierung der Server keinen zuverlässigen Chubby Service. Im Falle, dass ein RegionServer keine Verbindung zu dem MasterServer hat, trennt sich der Server und Start neu. In diesem Fall werden die Regionen des Servers auf andere RegionServer vom Hauptserver verteilt. Im Gegensatz dazu, kann ein TabletServer auch noch die Clients betreuen, obwohl die Verbindung zum HauptServer nicht mehr existiert. Nach dem Neustart von dem HauptServer verbinden sich die RegionServer und übergeben die Liste von betreuten Regionen.

Metadaten und Standorten der Regionen werden in der META-Tabelle gespeichert. Die Standorte von allen Regionen der META-Tabelle werden in der Tabelle ROOT gespeichert. Die ROOT Tabelle besetzt immer eine Region. Clients senden eine Anfrage über den Standort von der ROOT Region an den HauptServer.

Wie andere Komponenten die zur Handoop Plattform gehören, ist HBase mit Java realisiert worden. Das System hat mehrere Interfaces für Client-Anwendungen, dass sind z.B. Java API und das REST- Interface. Die Klienten können auch über die

Hbase Shell mit dem System kommunizieren. HBase bietet auch ein Webinterface das eine anschauliche Information über das System und die in ihm gespeicherten Tabellen liefert.

In der NoSQL Welt befinden sich verschiedene Systeme. Im nächsten Abschnitt möchte ich eine andere Idee zur Realisierung von Datenmanagementsystemen vorstellen.

5. Yahoo PNUTS

Das PNUTS ist ein Parallel verteiltes und geografisch orientiertes Datenbanksystem. Dieses System ist speziell für Webapplikationen von Yahoo entwickelt worden. Im PNUTS werden die Daten nicht nur als geordnete Datenstruktur wie bei HBase und BigTable, sondern auch in der gehashten Datenstruktur gespeichert. PNUTS ist ein verteiltes und zentral gesteuertes Servicesystem.

Hier wird eine Betonung auf die Skalierbarkeit, ein kürze Antwortzeit und die Verfügbarkeit gemacht. Das System arbeitet mit den Duplikaten. Bei PNUTS wird auf die Transaktionen verzichtet, weil es mit den Duplikaten zu viel Leistungen im Anspruch nehmen würde. Insbesondere wurde bei der Entwicklung von PNUTS ein besonderes Augenmerk auf die Datenkonsistenz gelegt.

Das Datenmodell wird im PNUTS im Form eines vereinfachten relationalen Modells dargestellt. Die Daten können in geordneten sowie auch gehashten Tabellen angelegt werden. Das Schema ist wie bei HBase sehr flexibel. Neue Attribute können auch zur Laufzeit hinzugefügt werden. PNUTS erlaubt es dem Anwender zu definieren ob die Tabellen gehasht oder geordneten erzeugt werden.

Die Anfragesprache von PNUTS unterstützt Selektionen und Projektionen auf die einfachen Tabellen. Beim Update oder Delete muss erstmal der Primärschlüssel deklariert werden. Anfragen auf die einfachen Tabellen bieten sehr flexible Artbeimöglichkeiten mit den verteilt gespeicherten Daten. Das System unterstützt keine Join- oder Gruppierungs Anfragen.

5.1 Funktionalitäten des PNUTS

Das PNUTS bietet ein Modell, das sich zwischen Realisierbarkeit und Konsistenz befindet. Das Modell fundiert auf der Tatsache das Webapplikationen nur einen Datensatz gleichzeitig manipulieren, während verschiedene Datensätze an verschieden geografischen Orten gebraucht werden. PNUTS bietet eine per-record-time-consistency. Hier gelten für alle Duplikate von dem Datensatz alle Update Operationen in der gleichen Reihenfolge.

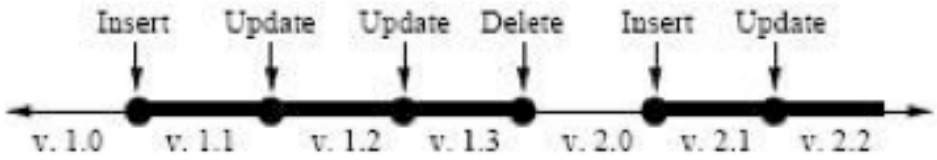

Abbildung 5.[2]|

In der Abbildung 5 wird eine Reihe von Änderungen graphisch dargestellt. Die Änderungen der Daten sind die Inserts, Updates und Deletes eines Primärschlüssels. Die Zeit in der die Daten physikalisch in der Datenbank vorhanden sind, zeigt eine dicke Linie an. Der Lesevorgang der Repliken liefert eine konsistente Version auf dem Zeitlinie. Die Repliken bewegen sich immer vorwärts auf dem Zeitstrahl.

Dieses Model wird wie folgt aufgebaut. Ein Replikat von dem Datensatz bezeichnet man als Master. Alle Änderungen dieses Datensatzes werden zum Master weitergeleitet. Um die Auslastung besser zu verteilen, wird das Replikat, auf dem häufige Schreibanfragen passieren als Master gekennzeichnet. Jeder Datensatz hat eine Nummer, die bei Änderungen erhört wird. Wie es aus der Abbildung 5 folgt, besteht die Nummer aus der Generation und der Versionsnummer. Jedes Insert ist eine neue Generation und bei jeden Update wird eine neue Version erzeugt.

Mit dem per-record time consistency Modell, wird eine Reihe von API Funktionen gewährleistet. Die Funktionen bieten unterschiedliche Konsistenzgarantien. Die Funktion Read-any liefert eine gültige und möglichst veraltete Version vom Datensatz. Read-critical (requered_version) liefert die Daten mit der eingegebenen Version oder eine neuere Version als requered_version. Die neueste Version wird mit der Read-lastes Funktion geliefert. Die Funktion Write ändert einen Datensatz. Test-and-set-write(required), bei dieser Funktion wird ein Datensatz gespeichert der genau der aktuellen Version entspricht. Die Funktion kann zur Implementierung von Transaktionen verwendet werden. Solche Transaktionen lesen erstmal ein Objekt ändern es dann.

Mit diesem Konzept können die Nutzer selbst entschieden ob sie mit den Einschränkungen der Konsistenz arbeiten wollen und dadurch die Performance erhöhen wie z.B. die Funktionen Read-any oder Read-critical. Wenn man eine Transaktion ohne Sperrungen nachbilden möchte, nimmt man Funktion test-and-set-write. Das PNUTS Modell bietet Serialisierbarkeit auf einer Per-Record Basis. Insbesondere wenn eine Anwendung mehrere Lese- und Schreiboperationen in einer Transaktion ausführt, so muss sie mit den Datensatz Versionen Arbeiten.

Für die Zukunft sind Funktionen wie Bundle Updates und Relaxed consistency vorgesehen. Die Funtkion Bundle Update bietet die Konsistenzgarantien für Schreiboperation auf mehreren Datensätzen. Im dem Fall, dass bei der Ausführung einer Funktion der Master ausfällt, wechselt das System auf ein anderes Replikat. Das kann dazu führen, dass die Konsistenz der Daten verletzt wird. Dafür wird eine neue Funktion (Relaxed consistency) entwickelt, die es erlaubt einer Anwendung zu entschieden wie in diesem Fall fortzufahren ist.

Triggerähnliche Benachrichtigungen sind für einige Anwendung sehr wichtig. Solche Benachrichtigungen dienen dazu um Kopien von Daten die sich im Cache befinden als ungültig zu markieren. Dafür erlaubt PNUTS dem Nutzer einen Update-Stream zu abonnieren. Aufgrund der Tatsache, dass es hier eine Publish/Subscribe (pub/sub) Infrastruktur verwendet wird, sind solche Benachrichtigungen einfach umsetzbar. Es bietet die gleichen Konsistenzgarantien wie der Mechanismus zur Datenreplikation.

5.2 Systemarchitektur und Implementierung von PNUTS

In der Abbildung 5.1 wird die System Struktur von PNUTS grafisch dargestellt.

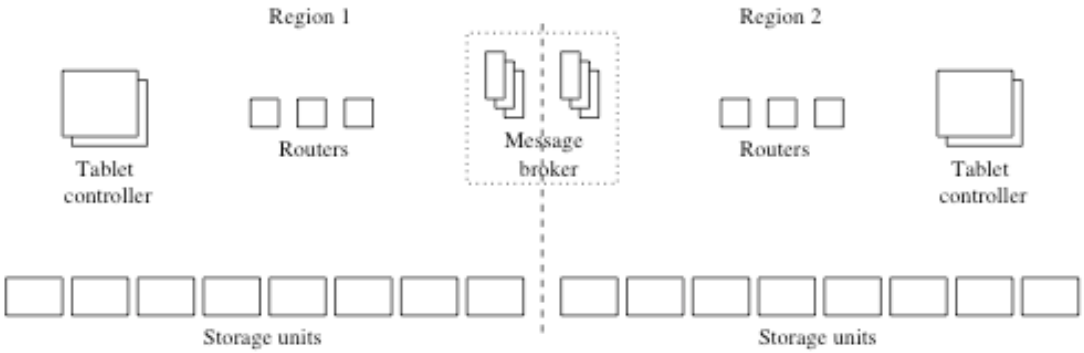

Abbildung 5.1 [2]

Das System wird in Regionen geteilt. Jede Region hat alle benötige Systemkomponenten und eine Kopie von allen Tabellen. Ein wichtiges Merkmal von PNUTS ist die Verwendung vom pub/sub-Mechanismus für die Zuverlässigkeit und die Replikation. Das macht die Verwendung von Redo-log unnütz. Weil die Daten auf

verschiedenen Regionen repliziert sind, ist hier die Verwendung von Archiven oder Backups unnötig.

Tabellen sind in Tablets unterteilt. Die Tablets werden von den verschiedenen Servern gesteuert. Jedes Tablet ist innerhalb einer Region nur auf einen Server gespeichert. Die Größe der Tablets bei PNUTS beträgt mehrere hunderte Megabytes. Die Zuordnung von Tablets zu den Servern ist flexibel. Das erlaubt die Auslastung gleichmäßig zu verteilen. Zum Beispiel beim Ausfall eines Servers werden die wiedererstellten Tabellen auf anderen Servern verteilt. Datenwiederherstellung bedeutet verlorene Tablets von einem seiner Repliken zu kopieren.

Die Storage-Units, Router und Tablet-Controller sind für Verwaltung und den Zugriff auf die Tablets zuständig. Bei den Storage-Units werden die Tablets gespeichert. Sie beantworten die get(), set() und scan() anfragen. Bei Hash-Tabellen wird ein Unix Dateisystem verwenden. Für die geordnete Tabelle wird mySQL mit InnoDB verwendet, weil die Datensätze geordnet nach Primärschlüssel gespeichert werden.

Der Router bestimmt welche Storage-Unit für einen zu lesenden oder zu schreibenden Datensatz zuständig ist. Um das zu bestimmen, ermittelt der Router zuerst welches Tablet diesen Datensatz enthält und welche Storage-Unit das Tablet verwaltet.

Geordnete Tabellen werden in einem Intervall nach Primärschlüsseln zugeteilt und einem Tablet zugeordnet. Der Router speichert eine Liste mit den Intervallen und den dazugehörigen Storage-Units. Dieser Vorgang heißt Interval-Mapping. Zum suchen von Tablets wird eine Binärsuche über Abbildungen verwendet. Sobald die Tabelle gefunden wird, werden dann gleich zugehörige Storage Units gefunden. Eine grafische Darstellung zum Interval Mapping ist in der Abbildung 5.2 (a) zu sehen.

Um die Hash-Tabellen zu organisieren wird ein n-bit Hashfunktion verwendet. Die Funktion erzeugt 2^n verschiedene Werte. Die Hash-Werte werden auch in Intervalle geteilt und dem Server zugewiesen. Um einen Schlüssel einem Tablet zuweisen wird er gehasht und dann mit binärer Suche der dazugehörige Interval gesucht. Aufgrund der Symmetrie des geordneten Tabellenverfahrens wird der gleiche Quellcode für das Suchen der Intervalle verwendet.
Die Hashtabellenarstellung ist in der Abbildung 5.2 (b) dargestellt.

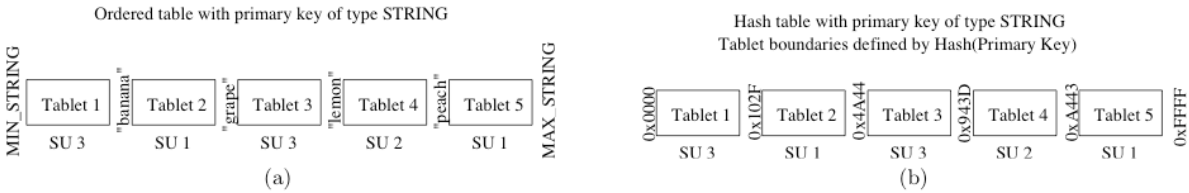

Abbildung 5.2. [2]

Die Router speichern bei sich nur eine gecachte Kopie aus den Interval Mappings. Das mappen wird vom Tablet-Controller übernommen. Der Router fragt in regelmäßigen Abständen die Änderungen der Tablets nach. Tablet Controller entscheidet über die Zeit für die Verteilung von Tablets auf mehrere Server. Nach diesem Vorgang kann es dazu kommen, dass die Kopien auf dem Router veraltet sind. Eine Anfrage an den Router wird dann mit einem Fehler beantwortet und der Router ruft eine neue Kopie vom Controller ab. Fällt der Router aus wird er neu gestartet.

Das PNUTS-System verwendet asynchrone Repliken. Das ermöglicht die Zugriffszeiten zu verkleinern. Als Ersatz zum Redo-log und Replikationen wird hier der Yahoo Message-Broker verwendet. Datenänderungen werden erstmal dem YMB übergeben, danach verteilt YMB asynchron die Änderungen auf verschiedene Regionen und sie werden danach auf ihre eigenen Repliken angewandt. PNUTS verwendet YMB für das Replizieren und Protokollieren aus zweierlei Gründen.

Erstens nimmt YMB mehrere Schritte, um sicherzustellen, dass die Nachrichten nicht verloren gehen bevor sie auf die Datenbank angewendet wurden. Die Idee ist, dass die Nachrichten auf mehreren Servern protokolliert werden. Bevor PNUTS nicht bestätigt hat dass die Änderungen auf alle Replikate angewendet wurden, werden keine Nachrichten aus dem Protokoll gelöscht. Zweitens ist YMB für wide-area Replikationen entwickelt worden. Es clustert seine Gebiete geografisch in separaten Rechenzentren. Die Nachrichten des Rechenzentrums werden an andere Cluster weitergeleitet. Dies ist der Mechanismus für individuelle PNUTS-Cluster im Umgang mit Update-Verbreitungen zwischen Regionen.

Für die Zeitliche Konsistenz wurde ein per-record masterschip Mechanismus eingebaut. Eine Kopie eines Datensatz wird als Master gekennzeichnet und alle Updates werden auf Ihr ausgeführt. Unterschiedliche Datensätze aus den gleichen Tabellen können in unterschiedlichen Clustern ihre Master haben. Zur Master Identifizierung enthält jeder Datensatz ein verstecktes

Metadatenfeld. Master Status kann von einer Replik auf eine andere übergeben werden. Das ermöglicht die Auslastung gleich zu verteilen.

Um die Primärschlüsselbedienung einzuhalten, müssen alle Inserts Operationen senden. So wird es ermöglicht die Reihenfolge von den Operationen zu bestimmen. Das System bietet auch noch andere Funktionalitäten.

Am Ende dieses Abschnittes möchte ich die Hauptmerkmale von NoSql Datenbanken zu den Klassischen Dateisystemen zusammenzuführen :

- Fokus auf Speicherung riesiger Mengen schwach strukturierter Daten;
- Hohe Performance und Skalierbarkeit
- Schemafrei, es existieren keine Fremdschlüssel
- Es wird kein Relationales Model unterstützt, die Spalten sind nicht typisiert, es werden keine Joins unterstützt, es gibt keine Transaktionen usw.

6. Zusammenfassung

Die NoQSL Welt ist groß. Verschiedene Organisationen entwickeln unterschiedliche Datenbanksysteme die für unterschiedliche Zwecke in unterschiedlichen Bereichen dienen.
Obwohl Projekte von Google und Yahoo sich an einigen Stellen sehr ähnlich sind, gibt es auch sehr große Unterschiede. Beide Systeme haben ein ähnliches Datenmodel. Sie bieten auch die Möglichkeit neue Attribute innerhalb der Laufzeit hinzuzufügen. Im Unterschied zu BigTable kann PNUTS die Daten nicht nur in geordneten Tabellen speichern, sondern auch Daten in den gehashten Tabellen ablegen.

Das Datenmanagmentsystem PNUTS ist konzeptionell dazu gezwungen permanent Datenpakete zu replizieren und diese auch einer konstanten Synchronisation zu unterziehen um eine ausreichende Konsistenz der Datensätze zu gewährleisten. Das BigTable-System von Google hingegen verwendet den Lock-Service Chubby um die Ordnung und Wiederverstellbarkeit der gespeicherten Daten zu garantieren.

Um Daten zu Speichern oder zu Löschen, z.B. bestimmte Zeilen lesen oder Berieche zu interieren, verwenden PNUTS und BigTable unterschiedliche Konzepte. Beide Konzepte sind relative durchsichtig implementiert. PNUTS verwendet unterschiedliche Methoden mit unterschiedlichen Konsistenzgarantien und BigTable erweitert seinen Funktionsbereich zum Konfigurieren seiner Spaltenfamilien.

Trotz der Tatsache, dass originale Realisationen dieser Technologien geschlossene Entwicklungen sind, werden von der Open-Source Community öffentliche Analogien sehr aktiv entwickelt.

Literaturverzeichnis

1. Bigtable: A Distributed Storage System for Structured Data. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber. 2008

2. PNUTS: Yahoo!'s Hosted Data Serving Platform. Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver and Ramana Yerneni Yahoo! Research. 2008

3. Hadoop: The Definitive Guide. Tom White. 2009

4. Pro Hadoop Jason Venner. 2009

5. NoSQL – Schöne neue Welt? Alternative Speichersysteme
http://it-republik.de/jaxenter/artikel/NoSQL-%96-Schoene-neue-Welt-2710.html

6. Publikation von Michael Stonebraker The "NoSQL" Discussion has Nothing to Do With SQL
http://cacm.acm.org/blogs/blog-cacm/50678-the-nosql-discussion-has-nothing-to-do-with-sql/fulltext

7. Is the Relational Database Doomed?
http://www.readwriteweb.com/enterprise/2009/02/is-the-relational-database-doomed.php

8. NoSql Databases
http://www.vineetgupta.com/2010/01/nosql-databases-part-1-landscape.html