BigTable vs. HBase

Iman Gharib

Schriftliche Ausarbeitung angefertigt im Rahmen des Seminars NOSQL

Universität Leipzig
Fakultät für Mathematik und Informatik

Wintersemester 2012

Betreuerin: Diplom-Bioinformatikerin Anika Groß

Inhaltsverzeichnis

Zusammenfassung

BigTable bzw. die frei verfügbare Variante HBase bieten ein zuverlässiges System, das mit Peta-Bytes von Daten umgehen kann und gegen Ausfälle sicher ist. Erwünscht wäre ein skalierbares System auf Standard-Hardware. BigTable(HBase) hat es so weit geschafft. Die können bei den großen Datenmengen sehr gut verwendet werden und tausende von Befehlen behandeln. Beide Systeme verbessern sich ständig, d.h. sie unterliegen Änderungen. In dieser Arbeit werden HBase und BigTable vorgestellt und miteinander vergleichen.

1 BigTable

1.1 Motivation

Es gibt viele strukturierte oder teilweise strukturierte Daten in der Google Welt, wie zum Beispiel URLs und viele Daten, die wir im Auge behalten wollen. Es gibt Links und Anchors, die auf einer Webseite zu einer anderen verweisen.

Außerdem gibt es benutzerdefinierten Einstellungen und der letzte Suchverlauf, die verwaltet sollen werden. Letztendlich kommen die vielfältigen und massiven geografischen Informationen von den Satellitenfotos zur Lokalisierung von Benutzer. Die Menge von all diesen Informationen ist sehr groß und es ist wichtig, dass die Behandlung von den Informationen und die Antwort zu den Benutzer in einer kurzen Zeit (mit minimaler Verspätung) geschehen.

Aufgrund der oben genannten Probleme ist die Nutzung eines kommerziellen Datenbank-Systems kostenintensiv. Diese Menge von Daten ist zu viel für viele DBS und, wenn ein Datenbank System diese Menge behandeln könnte, würde es zu teuer sein. Außerdem kann ein DBS diese großen Datenmengen nicht verwalten, ohne Performanzverlusten zu unterliegen.

Google betont, dass es große Menge von Daten gibt, die in mehreren Datenzentren in der Welt verwaltet werden müssen.Außerdem können sogar zuverlässige Ausrüstungen scheitern. Erwünscht war eine günstige Lösung auf Basis günstiger Hardware. Es ist auch einfacher, wenn man mit einem selbst entwickelten System arbeitet in Vergleich zu einem System, in dem man nicht alle Sichten steuern kann.

Laut dem Google BigTable Paper[GGL04] ist BigTable „ a sparse, distributed, persistent multidimensional sorted map“.Die Bedeutung dieser Terminologie wird später erklärt.

Bei Google geht es um Tausende von Server, Terabytes von in-memory Datei, Petabyte von Festplatten basierten Daten und Millionen von Schreibe/Lese- Befehle pro Sekunde und es existieren mehr als 500 Exemplare von BigTable. Das größte Exemplar hat mehr als 3000 Maschinen, in denen mehr als 6 Petabyte gespeichert werden können. Einige Instanzen verarbeiten rund um die Uhr mehr als 500.000 Anfragen pro Sekunde. Die Implementierung von BigTable startete im Jahr 2004 und hat ungefähr 1.5 Jahre gedauert. Viele Anwendungen basieren auf diesem System wie zum Beispiel Google Maps, Google Earth, Blogger, Orkut, Google print, Google Code, Google Book search, Gmail, Google Reader, My Search History, Crawling/indexing pipeline , Google Analystic, Google Finance und auch Youtube .

1.2 BigTable Data Model

BigTable ist ein Spalten-orientierter Key-Value-Store,in der jeder Eintrag in der Abbildung mit dem **(row key:string, column key:string, time:int64)String** Triple indiziert ist. Die gespeicherten Werte in der Tabelle sind beliebig lange Arrays von Bytes. Mit dem unten dargestellten Bild (Abb.1) wird alles zusammengefasst dargestellt und danach detaillierter behandelt.

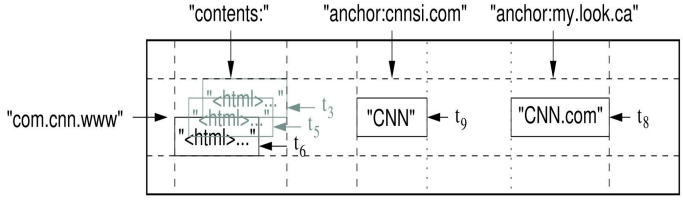

Abbildung 1: Ein Ausschnitt einer Beispieltabelle zum Speichern von Webseiten. [GGL04]

Die Zeilennamen bestehen aus den invertierten URL´s (siehe 1.2.1). Die Content_Column Families enthalten die Inhalte der Webseite. Die Anchor-Column Families speichern den Text der Referenz der Seite. CNN’s Homepage wird von Sport-Illustrated(cnn.com) und MY-look(mylook.com) referenziert. Somit erhält die Zeile die Spalten mit den Namen anchor:cnnsi.com und anchor:my.look.ca. Jede Anchor-Zelle hat einen Zeitste-mpel. Die Contents-Spalte hat drei verschiedene Versionen mit den Zeitstempeln t3, t5 und t6.

1.2.1 **Zeilen**

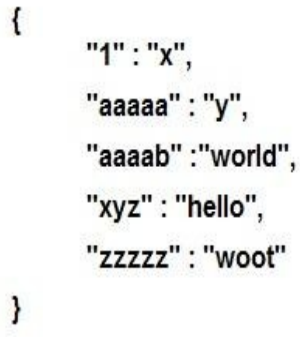

Abbildung 2: lexikographisch geordnete Abbildung.
(Key mit Value)[W08]

Die Zeilen in den Tabellen sind nach dem Schlüssel lexikographisch (Abb.2) und rückläufig sortiert. Dies ist sehr wichtig, damit Werte die zusammen gehören(z.B URLs der selben Domäne) untereinander auftauchen. So können die Schlüssel besser lokalisiert werden, und es bietet einfachere Host und Domäne Analyse.

Die Zeilenschlüssel sind Strings, die momentan eine 64 KB Größe haben, obwohl die typische notwendige Größe für die meisten Benutzer 10 bis 100 Bytes ist. Der Zeilenbereich der Tabelle wird dynamisch partitioniert. Jeder Zeilenbereich wird Tablet[1.2.5] genannt. Tablets sind Einheiten für Distribution und Load-balancierung.

Es ist zu beachten, dass jeder Schreib/Lese Befehl von Daten aus einer Zeile „atomar“ ist. Eine Design-Entscheidung,die Aktualisierungen in der selben Zeile für die Klient einfacher macht.

Um alles besser zu verstehen wird ein Beispiel an Triple Schlüssel dargestellt.Die Zeilen werden wie folgt lexikographisch eingeordnet. So eine URL wie „com.jimbojw.mail“ wird neben „com.jimbojw.www“ gespeichert.

1.2.2 **Column Families**

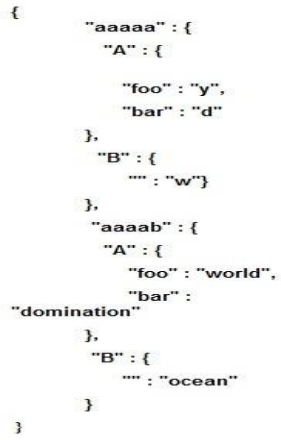

Abbildung 3: Column Families [W08]

Column keys (Spaltenschlüssel) sind in den Einheiten gruppiert und werden Column Families genannt. Die Dateien die in einer Column Family gespeichert sind, sind normalerweise vom selben Typ. Bevor die Dateien in einer Spalte gespeichert werden können, sollte die Column Family definiert sein.

Eine Column Family zu manipulieren oder eine neue Familie hinzufügen ist aufwendig. Deswegen ist es wichtig, dass alle notwendige Column Families ganz am Anfang definiert werden.

Den Spaltenschlüssel ist nach folgender Syntax zu definieren:family:qualifier. Column Families können eine beliebige Anzahl von Spalten haben, die als „qualifier“ bezeichnet werden. Qualifier ist der Name der Spalte innerhalb der Familie. Das wird anhand des Beispiels verdeutlicht, in dem zwei Zeilen mit jeweils zwei Column Families(A und B) stehen. Wobei A zwei Spalten (foo und bar) und B eine leere Zeichenkette "" enthält.

1.2.3 **Zeitstempel**

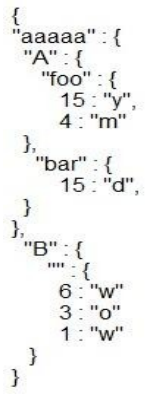

Abbildung 4: Time Stamp [W08]

Die letzte Dimension von unserem Tripel Schlüssel ist der Zeitstempel. Jede Zelle in der BigTable kann verschiedene Varianten von den selben Dateien haben. Diese Varianten sind mit dem 64 Bit großen Zeitstempel (int) indiziert. Diese werden entweder vom BigTable als „real Time“ bezeichnet (in Mikrosekunden) oder vom Client. Anwendungen die eine Kollision vermeiden wollen, müssen selber einen einzigartigen Zeitstempel erzeugen. Die unterschiedlichen Varianten von einer Datei werden mit aufsteigenden Zeitstempel gespeichert, sodass die aktuelle Version immer zuerst gelesen werden kann. In dem obigen Beispiel Suche nach Zeile/Spalte von "aaaaa"/"A:foo" gibt y zurück und anfragen an Zeile/Spalte/-Zeitstempel "aaaaa"/"a:foo/10 gibt "m" zurück. Anfrage an "aaaaa"/A":foo"/2 gibt Null zurück. Das bedeutet, es wird automatisch der letzte Zeitpunkt angegeben, aber bei einer Einschränkung wird der kleinste Zeitpunkt, der zu diesem Zeitpunkt am nahesten ist, gezeigt. Falls die Einschränkung kleiner als der geringste Zeitpunkt ist, kommt null raus.

1.2.4 **Tablets**

Tabellen werden automatisch an geeigneten Zeilengrenzen in so genannte Tablets eingeteilt. Sie beinhalten anliegende Zeilen. Im Durchschnitt gibt es 100 bis 200 MB Daten in jedem Tablet und etwa 100 Tablets pro Maschine. Falls ein Tabletserver außer Betrieb ist, werden die Tablets unter anderen Servern aufgeteilt, sodass jeder Server für eine Menge von den Tablets verantwortlich ist. Ein Tablet besteht aus mehreren SSTable Daten. Die Abbildung 6 soll dies verdeutlichen:

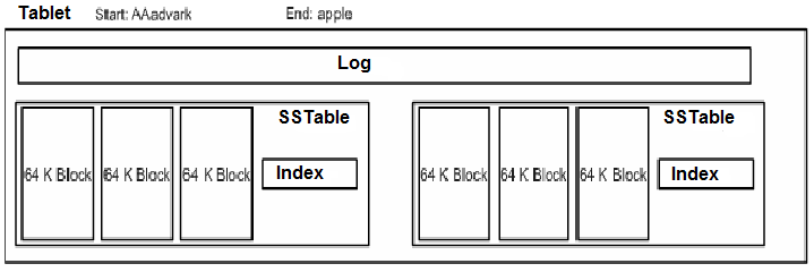

Abbildung 5: Tablet Design [DGHW06]

1.2.5 **Locality Groups**

Clients können verschiedene Column Families in einer “locality group” anordnen. Eine separate SSTable wird für jede “ locality group ” in jedem Tablet generiert. Google SSTable(Sorted Strings Table) Datei Format ist ein Datei aus Key (Row,Column,Time)/Value Paare, die sortiert sind. Diese Dateien dienen zur Speicherung der Tablets. Jedes SSTable besteht aus Sequenzen von Blöcken (normalerweise 64 K). Um die Blöcke zu lokalisieren, wird ein Blockindex, der am Ende der SSTable steht benutzt. Wenn die SSTable schon aufgebaut ist, kann dieser nicht geändert werden. Die alten SSTables (gelöschte) werden mit Garbage Collection behandelt. Neben SSTable ,ist GFS auch eine andere Google Infrastruktur, worauf BigTable aufgebaut ist. Über GFS wird im Kapitel 1.5 mehr gesagt. Die Trennung der nicht miteinander verbundener Column Families in verschiedene locality group, macht das Lesen einfacher.

Metadaten und Inhalt einer Seite können in zwei unterschiedliche Lokalisierte Gruppen eingeordnet werden. So muss ein Programm, das nur die Metadaten lesen will, nicht den gesamten Inhalt von allen Seiten lesen. Abbildung 5 zeigt wie locality group aussieht.

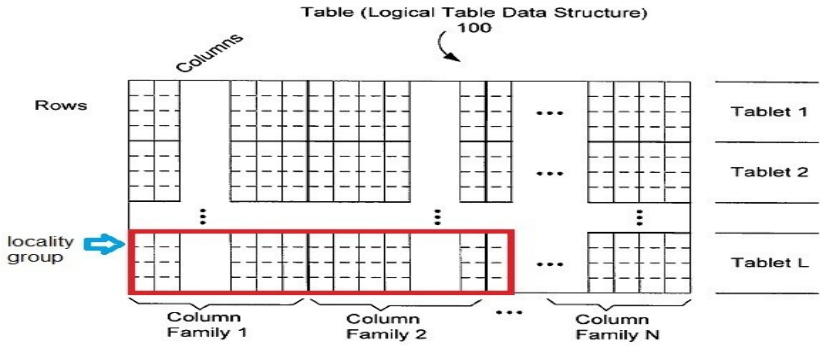

Abbildung 6: locality group Beispiel

1.3 API

Um die Tabellen und Column Families zu erzeugen/löschen oder die notwendigen Funktionen zur Manipulation von Clustern, Tabellen und Column Families, wie zum Beispiel um Zugriffsrecht zu implementieren, wird die BigTable API benutzt. Die Dateien können in einer Zeile **atomar** geschrieben und gelesen werden. Damit können Zellen in die Zeile geschrieben werden und auch alle oder manche Zellen aus der Zeile gelöscht werden. Der API Scanner macht es möglich eine begrenzte Anzahl von den Zeilen zurück zu bekommen, oder Dateien in einer bis allen Zeilen zu suchen. Eine Anfrage an alle, besondere oder spezifische Column Families ist dadurch auch möglich. BigTable kann als Ein/Ausgang für das Map/Reduce Framework verwendet werden. Wobei Google in 2010 unter dem „Caffeine“ Projekt [PD10] das Backend System Percolator vorgestellt hat somit es sein Indizierungssystem erneuert. Das System verbindet Schlüsselwörter mit den URLs. Ein inkrementeller Prozess, der die Verzögerungszeit zwischen kürzlich gecrawleten Seiten und der Darstellung in dem Index stark verkürzt hatte. Kleinere Änderungen, die mit MapReduce nicht günstig zu behandeln sind, weil MapReduce auf großen Daten besser arbeitet, werden mit Percolator gemacht. Das System wird von Änderungen die zum Beispiel in einer Spalte geschehen sind durch Observers informiert und muss nicht die ganze Menge von Daten wieder verarbeiten.

1.4 Implementierung Struktur

Bis jetzt wurden die Grundlagen von BigTable Data Model vorgestellt. In diesem Abschnitt wird die Implementierung von BigTable beschrieben.BigTable besteht aus drei Haupt Komponenten (siehe Abb. 7). Diese Komponenten sind wie folgt:

-Eine Bibliothek, die zu jedem Klient verknüpft ist und den Tablet Standort cached .

-Ein Master Server, der Tablets an dem Tabletserver verweist und für Balancierung des Tabletservers und für die Erzeugung der Column Families verantwortlich ist.

-Tablet Servern, die Lese/Schreib Anfragen an die erzeugten Tablets steuern und automatisch addiert oder entfernt werden können.

BigTable Cluster speichert eine Anzahl von Tabellen, die selber eine Einheit von Tablets haben und jedes Tablet beinhaltet die ganzen Daten, die mit dem Zeilenbereich assoziiert sind. Bei der Vergrößerung einer Tabelle, teilt sich diese Tabelle automatisch in mehrere Tablets der Größe 100 bis 200 MB.

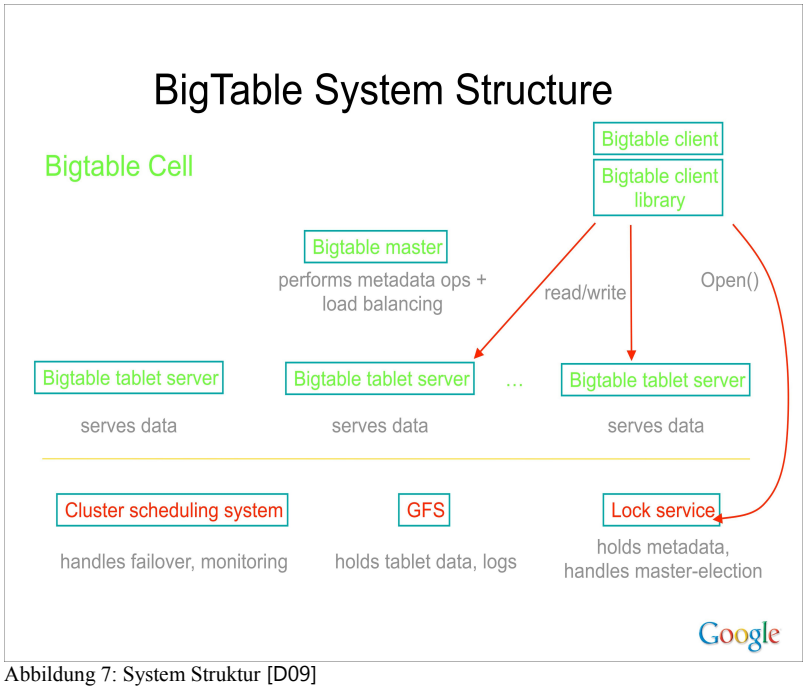

Abbildung 7: System Struktur [D09]

1.4.1 **Tablet Lokalisierung**

Das Finden des richtigen Tablets in der kurzen Zeit und mit den wenigsten Versuchen wird durch eine dreistufige Hierarchie ermöglicht (siehe Abb. 8). Zunächst muss erläutert werden, was ein Chubby ist.

Chubby ist ein verstreuter Lock-Service, der eine BigTable Cluster mit mehrere tausenden Knoten koordiniert und 5 Replikationen hat. Chubby kontrolliert den Lock Management Steuerungstraffic und die Tablet-Server-Lebenszeit. Er speichert Zugriffsrechtslisten und den Bootstarp Standort von BigTable Datei. Falls für eine bestimmte Zeit Chubby unerreichbar wird, wird BigTable auch unerreichbar.

Die Standortslokalisierung des Tablets geschieht in den folgenden Schritten:

1.Chubby speichert den Standort des Root tablet (Wurzeltablets). Das Wurzeltablet ist das erste Tablet in der METADATA Tabelle, das nie geteilt wird.

2. Das Wurzeltablet beinhaltet die Adresse aller METADATA Tablets.

3.Metadata Tablets beinhalten den Standort einer Gruppe von Benutzer- Tablets.

Mit diesem einfachen Schemata können bis zu 2^{34} Tablets adressiert werden. Falls der Standort eines Tablets für die Klienten nicht erkennbar oder falsch ist, muss die dreistufige Hierarchie rekursiv zurückgefahren werden.

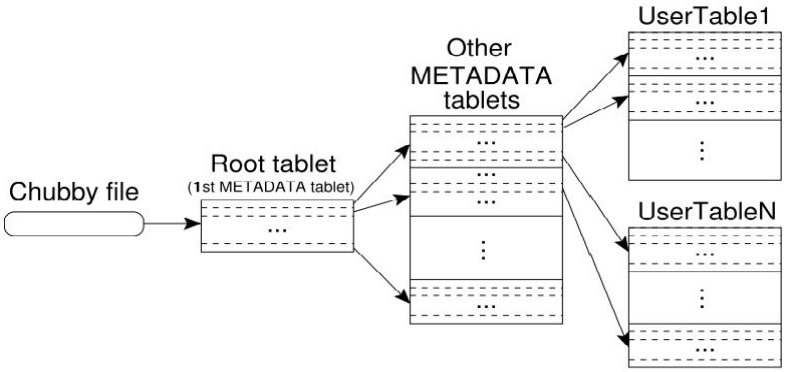

Abbildung 8: Dreistufige Hierarchie[CD08]

1.4.2 **Tablet Zuweisung**

Ein Tablet wird zu jeder Zeit nur von einem Tabletserver bedient. Der Master verfolgt die verfügbare Tabletservers und achtet auch darauf, ob ein Tablet einem Tabletserver zugewiesen ist oder nicht. Falls das Tablet nicht zugewiesen ist, wird es einem Tabletserver mit ausreichendem Speicherplatz zugewiesen. Tabletserver können durch Chubby verfolgt werden. Beim Start des Tabletservers wird ein eindeutiger Schlüssel in einem spezifischen Chubby Verzeichnis (Server Directory) erzeugt. Ein Tabletserver braucht den Schlüssel für seine Dateien solange wie die Datei existiert. Sonst terminiert sich der Tabletserver und gibt seinen Schlüssel frei. Der Master fragt zeitweise den Tabletserver nach seinem Schlüsselzustand. Falls der Tabletserver berichtet, dass er seinen Schlüssel verloren hat (keine Verbindung hat) oder es liefert keine Antwort zurück, versucht der Master den Schlüssel in der Datei des Servers zu finden. Bei Erfolg versteht der Master, dass die Chubby Datei immer noch am Leben ist und der Tabletserver ist entweder tot oder kann das Chubby nicht erreichen. In diesem Fall ist der Tabletserver nicht mehr nutzbar und wird bei der Entfernung des Server-File von Master als obsolet markiert. Die Tablets, die bis jetzt zu dem entfenten Tabletserver gehörten, werden unter nicht zugewiesenen Tablets eingeordnet.

1.4.2.1 **Master Aufgaben beim Start**

Das Cluster Management System startet den Master-Server. Der Master-Server muss die aktuellen Tabletzuweisungen herausfinden. Zunächst muss er sich einen eindeutigen Master Schlüssel in Chubby besorgen. Dann sucht er in dem Server-Directory um aktive Servers zu erkennen.

In diesem Moment kommuniziert er mit jedem Live Tabletserver und findet heraus, welche Tablets schon jedem Server zugewiesen wurden. Er fügt das Root-Tablet nicht zugewiesene Tablets hin zu. Am Ende scannt er die Metadaten Tabelle (nach der Zuweisung der Metadaten Tabellen) ein, um die nicht zugewiesene Tablets unter der Menge von noch nicht zugewiesenen Tablets einzuordnen. Alle Änderungen in der Tablet Menge sind von dem Master-Server verfolgbar außer der Tablet-Aufteilung, weil diese Prozess vom Tabletserver durchgeführt wird. Nach der Registrierung von Informationeneinträgen des neues Tablets, wird der Master-Server vom Tabletserver informiert.

1.4.3 **Tablet Änderungen**

Tablets Änderungen werden in ein „Commit Log“ geschrieben, indem „Redo-Records“ gespeichert sind. Die neuen Änderungen werden in einem Puffer auf den Arbeitsspeicher geschrieben, die wir memtable nennen (siehe Abb. 9) und die ältere Änderungen in einer Folge von SSTable Dateien. Wenn der Tabletserver ein Tablet wiederherstellen will, muss er zunächst die Metadaten, in denen es eine Liste von SSTabellen und entsprechende Redopunkte gibt, aus der METADATA Tabelle lesen. Die Redo-Points weisen auf den entsprechenden Commit-Log, in denen die Daten für das Tablet existieren. So kann der Server die memtable rekonstruieren.

Wenn ein Schreibbefehl ankommt, überprüft der Tabletserver die Berechtigung beim Lesen der Chubby Datei Liste, in der der Name der erlaubten Schreiber steht. Der Tabletserver überprüft also das Format der Befehle um sicher zu gehen, dass sie richtig formatiert sind. Dann wird der Befehl auf den Commit-Log und der Inhalt des Befehls auf die Memtable geschrieben.

Dies hat zur Folge, dass Memtable immer größer wird bis es seine Schwelle erreicht. Ab diesem Moment ist memtable gesperrt und ein neues memtable wird erzeugt. Die gespeicherte Tabelle wird in SSTable konvertiert. Diesen Prozess nennen wir „**minor compaction**“.

Beim Ankommen eines Lesebefehls an einem Tabletserver, wird der Befehl, wie beim Schreibbefehl überprüft. Wie wir oben gesagt haben erzeugt, die Minorverdichtung ein neues SSTable. Um das Lesen zu vereinfachen gruppieren wir eine Anzahl von Inhalt des SSTabels und memtable zusammen, und geben ein neues SSTable raus.

Dieser Prozess heißt „**merging compaction“**. Ein gültiger Lesebefehl führt auf dieser gemischten Sicht aus. Aber so kann es zu zu vielen Zugriffen auf die Festplatte kommen. Um das zu vermeiden erlauben wir den Klienten ein **Bloom-Filter** für SSTable in bestimmten Lokality-Gruppen zu spezifizieren. So können wir verstehen, ob eine SSTable für eine bestimmte Zeile oder Spalte Daten hat oder nicht. Suchen nach der nicht existierenden Zeile oder Spalte verursacht keine

Festplatten Zugriffe.

Zu den oben genannten Verdichtungen gibt es auch das „**major compaction**“.Es ist ein merging-compaction, die alle SSTables in eine SSTable hinschreibt. Andere Verdichtungen demgegenüber, produziert major compaction die SSTables, die keine obsolet markierte Informationen oder Daten haben, um zu sichern , dass entfernte Daten vom System in regelmäßigen Abständen gelöscht werden.

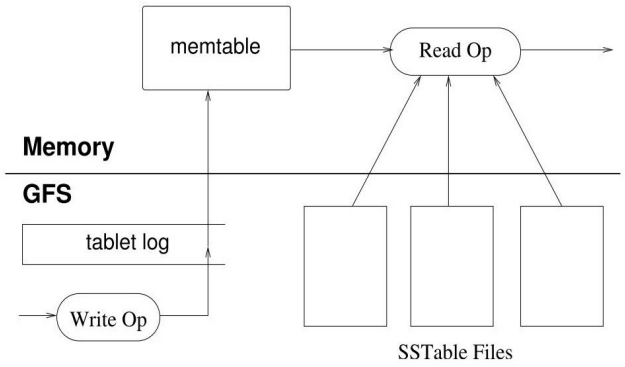

Abbildung 9: Tablet Bedienen [CD08]

1.4.4 Mehr Verfeinerungen

Eine noch wichtige Verfeinerung der BigTable Struktur ist die Compression(Komprimierung) des SSTables für eine locality group. Die Entscheidung ob eine Komprimierung passieren soll oder nicht und falls ja was für eine Komprimierung, trifft der Klient. Es gibt normalerweise zwei Komprimierungen. Eine BMDiff Methode, die auf allen Werten in einer Column-Familie stattfindet (komprimiert lange Strings in einem großen Fenster). Die zweite ist Zippy-Methode, die Wiederholungen in einem kleinen Fenster(16 KB) sucht.

1.5 GFS

BigTable benutzt GFS (siehe Abb.10) um Daten und Log-Dateien zu speichern. Hier wird GFS genauer betrachtet, obwohl nicht alles über GFS in dieser Arbeit behandelt werden kann. Laut Google's Forschungs-Wissenschaftler Jeffery Dean, ist die Idee hinter GFS, trotz Benutzung von den unzuverlässigen Maschinen die Daten zuverlässig speichern zu können. Das GFS-System arbeitet auf dem Master-Slave-Modul. Das heißt, eine Maschine als Server und mehrere andere, die Sklaven oder Knoten sind. Der Master ist verantwortlich, zu verfolgen, welche Daten auf welchem Rechner gespeichert sind. Die GFS soll drei Kopien von Daten oder Dateien einschließlich ausführbar zu erhalten.

Die Meta-Daten liegen auf Hauptspeicher des Master, d.h auf RAM, sodass ein schneller Zugriff möglich wird. GFS wurde entwickelt, um große Datenmengen zu speichern, bis das aktuelle Datum, die größten Google-Clusters (Cluster ist eine Gruppe von Computern zusammen als Netzwerk) speichern Hunderte von Terabyte über Tausende von Festplatten. Die Daten werden nicht durch schwer zu kontrollierende Schreibzugriffe verändert, sondern man fügt neue Daten einfach hinzu. Dadurch können mehrere Clients auf die Datenblöcke schreiben,sodass es nicht zu einer ständigen Synchronisation zwischen diesen kommt.

Ein GFS-Cluster besteht aus hunderten oder tausenden Chunk-Servern und einem Masterserver. Die Chunk-Servers sind dafür zuständig die Daten zu speichern. Die Dateien werden dabei in 64 MB große Stücke (Chunks) zur besseren Verwaltung aufgeteilt. Um Datenverlust bei einem Ausfall der einzelnen Server zu vermeiden, speichert GFS die Dateien dreifach ab und verteilt diese auf dem Cluster. Je nach Bedarf kann die Anzahl der Kopien auch höher gesetzt werden. Der Master Server speichert dagegen keine Chunks, sondern ihre Metadaten wie Dateiname oder Dateigröße und kennt den genauen Speicherort der originalen Datei sowie ihrer Kopien im Cluster. Die Anfrage von einem Client wird so zuerst von dem Master verarbeitet und als Antwort bekommt der Client die Adresse des dazugehörigen Chunkservers zurück.

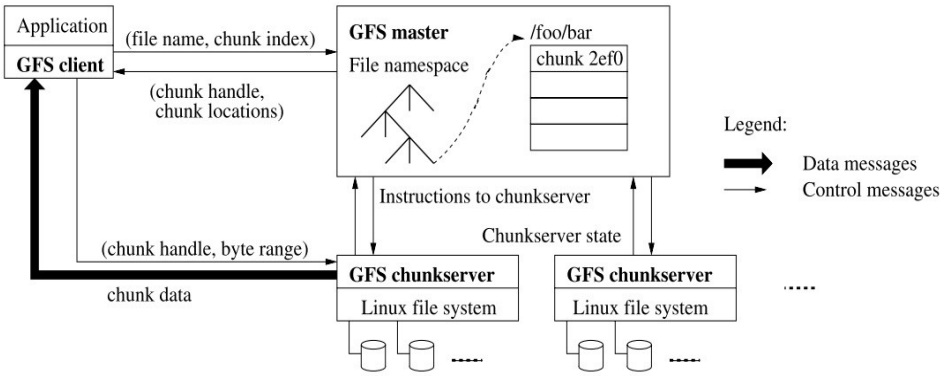

Abbildung 10: GFS [GGL04]

2 HBase

2.1 Einführung

Laut der Wikipedia [WK] ist HBase eine skalierbare, einfache Datenbank zur Verwaltung sehr großer Datenmengen innerhalb eines Hadoop-Clusters. Die HBase-Datenbank basiert auf einer freien Implementierung von BigTable. Diese Datenstruktur ist für Daten geeignet, die selten verändert werden, dafür aber sehr häufig ergänzt werden. Mit HBase lassen sich Milliarden von Zeilen effizient verwalten. Es wurde in Java geschrieben und begann als ein Projekt in **Powerset** Kompanie. Viele Firmen und Unternehmen benutzen HBase heutzutage. Dazu zählen Facebook Nachrichten Platform, Twitter und Yahoo genannt werden.

2.2 HBase Versionen

Die drei unten genannten Versionen sind momentan, die letzte Versionen [HL]:

- **0.20** –Diese Version unterstützt keine HDFS Durability und Änderung können bei einem Ausfall verloren gehen..
- **0.89** –Ist immer noch nicht empfehlenswert für Anwendungen. Diese Version unterstützt HDFS Durability.
- **0.90** – empfehlenswerte Version für Anwendungen, die HDFS Durability unterstützt.

Wie gesagt worden ist, ist HBase BigTable sehr ähnlich und es soll hier nicht alles wieder beschrieben werden sondern, nur die wichtigsten unterschiedlichen Aspekte von HBase in Vergleich zu BigTable-

2.3 Terminologie

Es gibt einige terminologische Unterschiede zwischen HBase und BigTable, die wie folgt sind:

Hadoop FS entspricht GFS und Hbase Master/RegionServer sind jeweils Master /Tablet Server Äquivalente in BigTable. MapFile ähnelt der Datei wie SSTable und in HBase wird Memtable Memchach genannt.

Der HBase Master Server steuert Zuweisungen von Regionen zu Servern und überwacht deren Status. Die Standorte von allen Regionen der META-Tabelle werden in der Tabelle ROOT

gespeichert. Die ROOT Tabelle besetzt immer eine Region. ROOT Tabelle und Meta-Tabelle sind in HBase zwei Tabellen , aber in BigTable sind sie in der gleichen Tabelle. Die Funktionalität ist ähnlich in den beiden Systemen.

HBase ist mit Java realisiert worden. Das System hat mehrere Interfaces für Client-Anwendungen, dass sind z.B. Java API und das REST- Interface. Durch Hive kann man SQL Befehle auf HBase führen ,aber es dauert 4 bis 5 mal länger als in Relationalen Datenbanken. In der Zukunft soll dieser Unterschied kleiner werden.

3 Vergleich

Einige Unterschiede zwischen den Systemen wurden bereits genannt.

Regionen Namen in HBase bestehen aus einer Kombination des Tabellennames und Anfang des Zeilenschlüssels, aber in BigTable kommt der Tabellenname mit Endzeilenschlüssel. BigTable benutzt BMDiff und Zippy. HBase verwendet Java Komprimierungs Methoden, will aber auch in der Zukunft BMDiff benutzen. HBase hat einen Server-side-Filter, der die Dateienmenge, die an den Benutzer gehen muss, reduziert. BigTable benutzt seine Sawzall Skript Sprache dazu.

HBase kann neben dem HDFS auch auf andere Dateieinsysteme wie S3 von Amazon laufen, aber BigTable ist auf GFS basiert.

BigTable kann die Dateien in den Arbeitsspeicher mappen, damit weniger Festplattenzugriffe stattfindet.Bisher kann HBase dies noch nicht, aber es wird daran gearbeitet.

Die locality group existiert nur in BigTable, wobei HBase nur die Column Families nutzt.

HBase kann commit-log beim Ankommen von sehr großen Datenmengen auslassen,wenn es wahrscheinlich ist, dass nach einem Serverausfall dieser Logs nicht zu behandeln sind. Somit kann HBase die Leistungsfähigkeiten erhöhen. Aber BigTable benutzt diese Option nicht.

Die Zusammenfügung der Tabellen in HBase ist manuell, aber in BigTable wird diese Arbeit über den Master und automatisch verhandelt.

BigTable kann die Daten in mehreren Datenzentren replizieren.

Im Gegensatz zu Hbase verfügt Bigtable über eine Zugriffsrechtskontrolle.

Zu der oben genannten Unterschiede kann man auch kleinere Unterschiede nennen, wie zum Beispiel, der Fakt, dass die TimeStamps in BigTable in Mikrosekunden und in HBase Millisekunden angegeben wird.

Zusammenfassend lässt sich sagen, dass beide Systeme einander sehr ähnlich sind und trotz der terminologischen und manchmal semantischen Unterschiede, verfolgen beide das gleiche Ziel.

Literatur:

[CD08] Fay Chang, Jeffrey Dean et al. Bigtable: A Distributed Storage System for Structured Data(2008), published in ACM Transactions on Computer Systems (TOCS)

[DGHW06] Jeffrey Dean,Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber:Ffrom talk by Erik Paulson,UW Madison.

[D09] Jeffrey Dean: Handling large Datasets at google (2009), From carfield.com.hk

[GGL04] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung: The Google File System(2003), published in: SOSP '03 Proceedings of the nineteenth ACM symposium on Operating systems principles

[HL] HBase Literatur, http://hbase.apache.org/book.html#other.info

[PD10] Daniel Peng and Frank Dabek: Large-scale Incremental Processing Using Distributed Transactions and Notifications (2010), From www.usenix.org

[Wk] WWW: http://en.wikipedia.org/wiki/HBase

[W08] Jim R. Wilson: Understanding HBase and BigTable (2008), From http://jimbojw.com/wiki

Abbildungsverzeichnis