Notizen zur Diskussion des Papers: Bigtable: A Distributed Storage System for Structured Data

von F. Chang, J. Dean, S Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, und R. E. Gruber

Welche drei Dimensionen verwendet Bigtable, um eine Zelle zu indizieren?

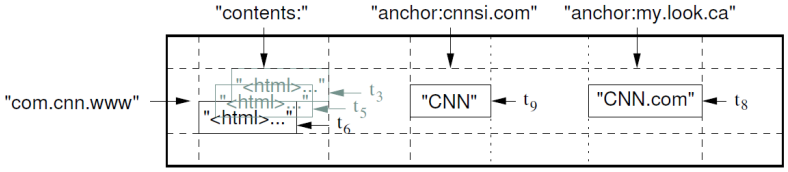

Figure 1: A slice of an example table that stores Web pages. The row name is a reversed URL. The contents column family contains the page contents, and the anchor column family contains the text of any anchors that reference the page. CNN's home page is referenced by both the Sports Illustrated and the MY-look home pages, so the row contains columns named anchor:cnnsi.com and anchor:my.look.ca. Each anchor cell has one version; the contents column has three versions, at timestamps t_3, t_5, and t_6.

Bigtable ist eine dünn besetzte, persistente, multidimensionale sortierte Map. Diese Map wird mittels eines Zeilenschlüssels, eines Spaltenschlüssels und eines Zeitstempels indiziert (row : string, $column$: string, $time$: int64 $\rightarrow$ string). Jeder Eintrag ist ein noch nicht interpretierter Byte-Array.

Die Daten in Bigtable sind lexikographisch nach dem Zeilenschlüssel sortiert. Die Zeilenintervalle einer Tabelle werden dynamisch unterteilt. Jedes Zeilenintervall bildet ein Tablet. Zeilenschlüssel können so gewählt werden, dass sie eine gute Lokalität der Daten garantieren.

Spalten können als sogenannte Spaltenfamilien gruppiert werden. Diese Spaltenfamilien müssen erstellt werden, bevor man sie füllt. Es gibt eine unbegrenzte Anzahl and Spaltenfamilien, aber unbegrenzt viele Spalten. Wir verwenden folgende Syntax: $family : qualifier$, um eine Spalte zu benennen. Zugriffskontrolle findet auf der Ebene der Spaltenfamilien statt.

Zeitstempel ermöglichen es mehrere Versionen von Daten in derselben Zelle zu speichern. Der Client kann festlegen, wie viele frühere Versionen behalten werden.

Welche Rolle spielt Chubby bei Bigtable und wie interagiert der Master beim Start mit Chubby?

Bigtable baut auf einen hoch verfügbaren, persistenten, verteilten Lock-Service namens Chubby auf. Ein Chubby-Service besteht aus fünf Kopien, einer von ihnen zum Master gewählt. Chubby bietet einen Namespace, welcher aus Ordnern und kleinen Dateien besteht. Jeder Ordner und jede Datei kann gelockt werden. Lese- und Schreiboperationen sind atomar. Jeder Client hat eine offene Sitzung mit dem Chubby-Service. Erneuert der Client diese Sitzung nicht innerhalb des

vorgegebenen Zeitrahmens, läuft diese aus und er verliert alle Locks und offene Datei-Handles.

Bigtable verwendet Chubby für eine Vielzahl von Ausgaben.

- Zur Garantie, dass nur ein Master aktiv ist.
- Um Bootstrap-Daten, Schemainformation und Access-Control-Listen abzuspeichern.
- Zum Auffinden von Tablet-Servern und um inaktive Tablet-Server aufzuspüren.

Wenn ein Tablet-Server startet holt er sich ein exklusives Lock bzgl. einer eindeutig identifizierten Datei innerhalb eines bestimmten Chubby-Ordners. Der Master beobachtet nun diesen Ordner, um festzustellen, ob Tablet-Server hinzugekommen oder weggefallen sind. Ein Tablet fällt weg, wenn es sein Lock verliert. Wann immer ein Tablet-Server herunterfährt gibt er sein Lock frei, sodass der Master die Datei schneller wieder zuteilen kann.

Der Master erkundigt sich bei den Tablet-Servern regelmäßig über den Status ihrer Locks. Meldet ein Server, dass er sein Lock verloren hat oder ist er vom Master aus nicht erreichbar, versucht der Master das Lock zu beanspruchen. Bekommt der Master das Lock, dann ist Chubby aktiv und der Tablet-Server ist entweder Offline oder er hat Probleme Chubby zu erreichen. Wird eine Datei gelöscht markiert der Master die entsprechenden Tablets als nicht zugewiesen.

Nachdem der Master von der Cluster-Verwaltung gestartet wurde führt er folgende Schritte durch.

1. Der Master holt sich ein einzigartiges Master Lock über Chubby.
2. Der Master liest den Server-Ordner in Chubby, um die aktiven Server ausfindig zu machen.
3. Der Master kommuniziert mit jedem aktiven Tablet-Server und findet heraus, welche Tablets bereits zugewiesen sind.
4. Der Master liest die `METADATA`-Tabelle, um mehr über alle Tablets herauszufinden.

Was sind SSTables und wofür werden sie benötigt? Welche Vorteile entstehen dadurch, dass SSTables unveränderlich sind?

SSTables (Sorted-String-Tables) ist ein Dateiformat, um Bigtable-Daten abzuspeichern. SSTables bieten eine persistente, geordnete, unveränderliche Map. Schlüssel und Werte sind beliebige Byte-Strings. Es werden Operationen zum lesen von Werten, welche mit einem bestimmten Schlüssel assoziiert werden, und zum Iterieren, über ein Intervall von Schlüssel/Werte-Paaren, bereitgestellt. Intern enthält jeder SSTable eine Folge von Blöcken (typischerweise 64KB). Ein Blockindex wird dazu verwendet, um einen Block zu lokalisieren. Dieser wird in den Hauptspeicher

geladen, wenn der SSTable geöffnet wird. Ein Lookup benötigt nur einen Disk-Seek.

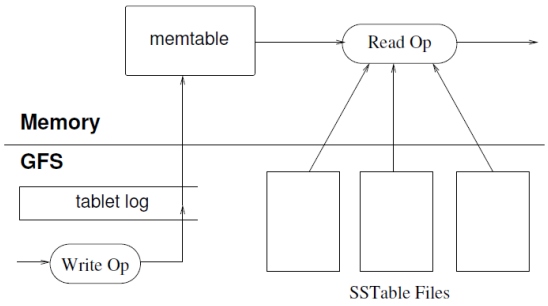

Figure 5: Tablet Representation

Wie funktionieren Bloom-Filter und welche Art von Anwendungen können durch deren Einsatz profitieren?

Ein Bloom-Filter ist eine probabilistische Datenstruktur, mit deren Hilfe sehr schnell festgestellt werden kann, welche Daten in einem Datenstrom schon einmal vorgekommen sind. False-Positives sind möglich, False-Negatives allerdings nicht. Es wird entweder "Sicher nicht Teil der Menge" oder "Möglicherweise in der Menge" zurückgegeben. Solch ein Filter erlaubt es uns anzufragen, ob ein SSTable Daten eines speziellen Zeilen/Spalten-Paares enthält. Für viele Anwendungen wird nur wenig Speicherplatz für Bloom-Filter benötigt, im Gegenzug erhöhen sie die Ausführungszeit jedoch drastisch. In Bigtable werden Bloom-Filter dazu benützt, um Lookups für nicht vorhandene Zeilen zu vermeiden.

Was passiert, wenn ein Tablet-Server stirbt? Wie wird er wiederhergestellt?

Der persistente Zustand eines Tablets wird im GFS gespeichert. Updates werden im Commit-Log abgelegt, welches auch Daten über Redos speichert. Kürzlich vorgenommene Updates werden in einem sortierten Buffer (im Hauptspeicher) gespeichert, dem sogenannten memtable. Die älteren Updates werden in einer Reihe von SSTables abgespeichert. Um ein Tablet wiederherzustellen, liest ein Tablet-Server seine Metadaten aus der `METADATA`-Tabelle. Diese Metadaten enthalten eine Liste von SSTables, welche ein Tablet und eine Menge an Redo-Punkten enthalten. Diese Punkte sind Pointer in jene Commit-Logs, welche Daten dieses Tablets enthalten könnten. Der Server liest diese Indizes (der SSTables) in den Hauptspeicher und rekonstruiert den entsprechenden memtable, indem er alle eingetragenen Updates, welche seit den Redo-Punkten übergeben wurde, anwendet.

Welche Refinements wurden umgesetzt und wie funktionieren sie?

- Clients können mehrere Spaltenfamilien in eine *Locality-Group* zusammenfassen. Ein SSTable wird für jede Locality-Group (in jedem Tablet) er-

stellt. So kann z.B. festgelegt werden, dass eine bestimmte Locality-Group im Hauptspeicher gehalten wird (für oft verwendete Daten).

- Der Client kann festlegen, ob und wie die SSTables einer Locality-Group komprimiert werden.
- Es können Bloom-Filter angewandt werden, um Anfragen für nicht vorhandene Daten zu vermeiden.
- Wenn ein *Commit-Log* für jedes Tablet in einer separaten Log-Datei gespeichert wird, würden sehr viele Dateien gleichzeitig in das GFS geschrieben. Eine einzige Log-Datei löst dieses Problem, verkompliziert allerdings den Wiederherstellungsprozess einzelner Tablets. Stirbt ein Tablet-Server, werden seine Tablets auf auf eine Große Anzahl anderer Tablet-Server aufgeteilt, um ein Tablet nun wiederherzustellen, wendet der neue Tablet-Server alle im Commit-Log eingetragenen Updates auf das Tables an. Um zu vermeiden, dass das Log mehrfach gelesen wird werden Einträge im Commit-Log nach folgenden Schlüsseln sortiert: (*tablet*, *row*, *name*, *log sequence number*). Die sortierte Ausgabe, aller Mutationen bzgl. eines bestimmten Tablets, befindet zusammenhängend im Speicher, so ist nur ein Disk-Seek gefolgt von einer sequentiellen Leseoperation notwendig.
- Wenn der Master ein Tablet verschiebt wird es vorher leicht komprimiert.
- Der Fakt, dass SSTables unveränderlich sind kann ausgenutzt werden. Es ist bspw. keine Synchronisation notwendig, wenn aus SSTables gelesen wird. So kann der Zugriff auf Zeilen einfach und effizient Parallelisiert werden. Da SSTables unveränderlich sind ist das Problem gelöschte Daten zu entfernen equivalent zum Problem obsolete SSTables der Garbage-Collection zuzuführen. Abschließend können wir Tablets somit einfach und schnell aufteilen. Anstatt neue SSTables für jedes Kind-Tables zu erzeugen, lassen wird zu, dass sich Kinder- und Eltern-Tablets die SSTablets teilen.

Wie gut skaliert Bigtable? Gibt es Bottlenecks?

Experiment	# of Tablet Servers			
	1	50	250	500
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843

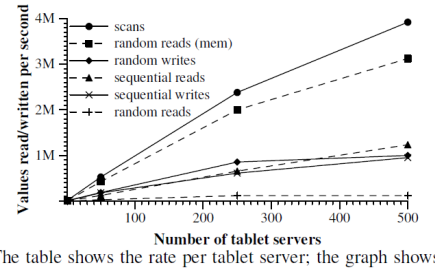

Figure 6: Number of 1000-byte values read/written per second. The table shows the rate per tablet server; the graph shows the aggregate rate.

Der Durchsatz erhöht sich dramatisch, wenn man die Anzahl der Tablet-Server von 1 auf 500 erhöht. Die Geschwindigkeit von Leseoperationen erhöht sich um nahezu 300. Die CPU der einzelnen Tablet Server stellt ein Bottleneck dar. Die Performanz erhöht sich allerdings nicht linear. Der Durchsatz pro Server fällt

extrem ab, wenn wird die Anzahl der Server von 1 auf 50 erhöhen. Dies lässt sich mit der unterschiedlichen Belastung der Server erklären.

Perlen der Datenbankliteratur

Alexander Miller (1120667)

25. Juni 2015

Notizen zum Paper *Bigtable: A Distributed Storage System for Structured Data* von F. Chang et al, 2006.

1 Vasker 1: Explain 3 dimensions of Bigtable.

- Row
- Column/ColumnFamilies
- Timestamp ("Versionierung")

Zugriff über (row-key, column-key [, timestamp]). Siehe Beispiel in Abbildung 1.

2 Augsten: Vergleich mit relationalem Modell

Relational: In den Zeilen sind die Daten, während die Spalten quasi nur das Schema beinhalten. BigTable: Die Anzahl der Spalten kann für jede Zeile variabel sein. Spalten enthalten auch Daten.

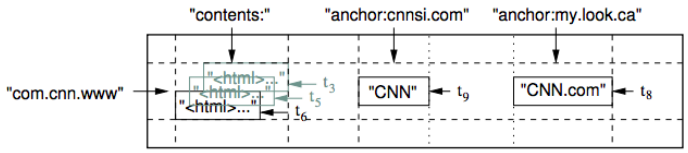

Figure 1: A slice of an example table that stores Web pages. The row name is a reversed URL. The contents column family contains the page contents, and the anchor column family contains the text of any anchors that reference the page. CNN's home page is referenced by both the Sports Illustrated and the MY-look home pages, so the row contains columns named anchor:cnnsi.com and anchor:my.look.ca. Each anchor cell has one version; the contents column has three versions, at timestamps t_3, t_5, and t_6.

Abbildung 1: Example table

3 Vasker 2: How does the distributed lock service Chubby works?

Chubby ist ein von Google selbst entwickelter, persistenter und verteilter Lock-Service. Es bietet eine Art Filesystem an, das aus Directories und Files besteht, die als Lock genutzt werden können. Schreib- und Leseoperationen sind atomar. BigTable nutzt den Chubby-Service für mehrere Aufgaben:

- Es wird damit sichergestellt, dass nur ein Master gleichzeitig aktiv ist.
- Weiters wird die Bootstrap location der Bitgtable-Daten dort gespeichert.
- Die Suche nach Tablet Server und deren Ableben
- Das Schema der Table
- Zugriffskontrolle

Es wird nur der Teil in Chubby gehalten der unbedingt die Garantien benötigt, alles andere im Google Filesystem. Clients müssen mit Master meistens nicht kommunizieren.

4 Robert 1: Welche Rolle spielt Chubby bei Bigtable und wie interagiert der Master beim Start mit Chubby?

Die Rolle wurde bereits in Abschnitt 3 erklärt. Der Startup des Masters sieht folgendermaßen aus:

1. Der Master holt sich das Master Lock von Chubby
2. Scan des Server directories in Chubby nach aktiven Servern
3. Kommunikation mit Servern um herauszufinden, welche tablets den Servern zugeteilt wurden
4. Scan der Metadaten um die Menge der Tables herauszufinden

5 Patrick 1: Was sind SSTables und wofür werden sie benötigt?

Google SSTable ist ein Fileformat, in dem die BigTable-Daten gespeichert werden. Ein SSTable ist ein unveränderlicher, persistenter, geordneter Key-Value-Speicher. Es wird ein Block-Index benutzt. Ein Tablet besteht aus mehreren SSTables.

Neue Daten werden erst in die Memtable geschrieben, eine Datenstruktur im Hauptspeicher, und erst später dann in SSTables geschrieben.

6 Robert 3: Welche Vorteile entstehen dadurch, dass SSTables unveränderlich sind?

- Schnell splitbar
- Keine Synchronisierung beim Lesen von der Platte.
- Das Löschen von Daten geschieht durch einen Garbage Collection Algorithmus

7 Vasker 3: How does a new tablet server recovers the previous state from commit log, when a tablet server dies?

Zum Wiederherstellen eines Tablets liest der Tablet Server aus den Metadaten die Liste der SSTables und eine Menge von redo points (Pointer ins Commit file). Danach werden alle Updates seit den Redo Points angewandt. Redo Points: Erste Operation die nicht mehr in einem SSTable ist. Das Set kann sich auf mehrere SSTables beziehen.

Beispiel einer Schreib-Operation: Neue Zeile schreiben

- Berechtigung und validieren
- Eintrag ins commit-file (liegt im GFS) (Optimierung: Group-Commits)

  $\Leftarrow$ Commit an den Client
- Eintrag in memtable

8 Patrick 2: Welche Refinements wurden umgesetzt und wie funktionieren sie?

- Locality group: Column Families die gemeinsam gelesen werden, werden in gemeinsame SSTables geschrieben.
- Compression: Daten in SSTables werden blockweise komprimiert
- Caching: Die Lesegeschwindigkeit wird erhöht durch einen zweistufigen Cache.
- Bloom filter: Zur schnellen Prüfung von Existenz von Row/Column-Paaren ohne Diskzugriff
- Commit log: Logs für unterschiedliche Tablets werden in einem File zusammengefasst.
- Immutability of SSTables: Wurde bereits beschrieben in Abschnitt 6.

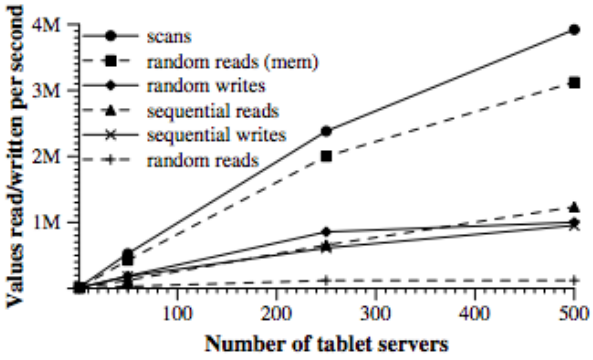

Abbildung 2: Durchsatz mit steigender Anzahl Server

- Speeding up recovery: Tablets werden kompaktiert bevor sie zu anderen Servern verschoben werden.

9 Robert 2: Wie funktionieren Bloom Filter und welche Art von Anwendungen können durch deren Einsatz profitieren?

Ein Bloom Filter ist eine probalistische Datenstruktur um die Existenz von Elementen in einer Menge zu prüfen. Dabei können False Positives auftreten, aber keine False Negatives. Dadurch werden SSTables ausgeschlossen, gemeinsam mit LocalityGroups wird die Anzahl an SSTables die gelesen werden müssen, minimiert.

10 Patrick 3: Wie gut skaliert Bigtable? Gibt es Bottlenecks? (In Bezug auf die Experimente)

Der Gesamtdurchsatz skaliert sehr gut, aber nicht linear. Als Bottleneck zeigte sich die CPU der Tablet Server. Siehe Abbildung 2. Die Server standen nicht exklusiv für das Experiment zur Verf¨gung, sondern mussten auch andere Aufgaben erledigen.