CMPT 843 Paper Presentation
March 14,2019

Bigtable : A Distributed Storage System for Structured Data

Presented by Ankita Sakhuja

Overview

- Introduction
- Data model
- Building Blocks
- Implementations
- Refinements
- Performance Evaluation
- Real Applications
- Conclusions

Motivation

- Google has lots of data
- Scale of data is too large. Even for commercial databases.

Even though *Google* is best known for its fast and reliable services, but what's working behind there ?

Undoubtedly, there are number of aspects that matter behind this (like Hardware, Software, OS, Best staff in the world etc.)

But what I am going to discuss here is the Software part

- ❑ GFS
- ❑ Chubby
- ❑ Bigtable

Why not a DBMS ?

- Scale is too large for commercial databases
- Cost would be very high
- Low-level storage optimizations help performance significantly
- Hard to map semi-structured data to relational database
- Non-uniform fields makes it difficult to insert/query data

What is Bigtable ?

Bigtable is a distributed storage system for managing structured data

Bigtable doesn’t support a full relational model

Scalable

Self-managing

Used for variety of demanding workloads

- Throughput oriented batch processing
- Latency specific data serving

Used by more than 60 google products

- Google Analytics, Google Finance
- Personalized Search ,Google Earth
- Google Documents...

Goals

- Wide applicability
- Scalability
- High Performance
- High Availability

Simple data model that supports dynamic control over data layout and format

Data Model

- A Bigtable is a sparse , distributed , persistent multidimensional sorted map.

Bigtable the key/value pairs are kept in strict alphabetical order.

A given row can have any number of columns in each column family, or none at all.

```
{
  "1" : "x",
  "aaaaa" : "y",
  "aaaab" : "world",
  "xyz" : "hello",
  "zzzzz" : "woot"
}
```

It’s built upon distributed filesystems so that the underlying file storage can be spread out among array of independent machines.

Data you put in this special map “persists” after the program that created or accessed is finished.

A map of all maps

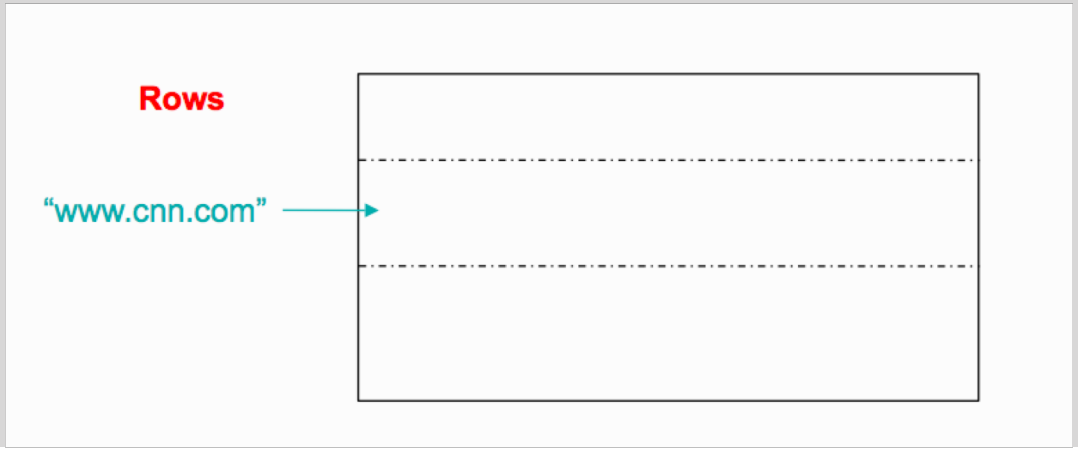
```
{
  "1" : {
    "A" : "x",
    "B" : "z"
  },
  "aaaaa" : {
    "A" : "y",
    "B" : "w"
  },
  "aaaab" : {
    "A" : "world",
    "B" : "ocean"
  },
  "xyz" : {
    "A" : "hello",
    "B" : "there"
  },
  "zzzzz" : {
    "A" : "woot",
    "B" : "1337"
  }
}
```

Data model

- The map is indexed by a row key, column key and a timestamp.

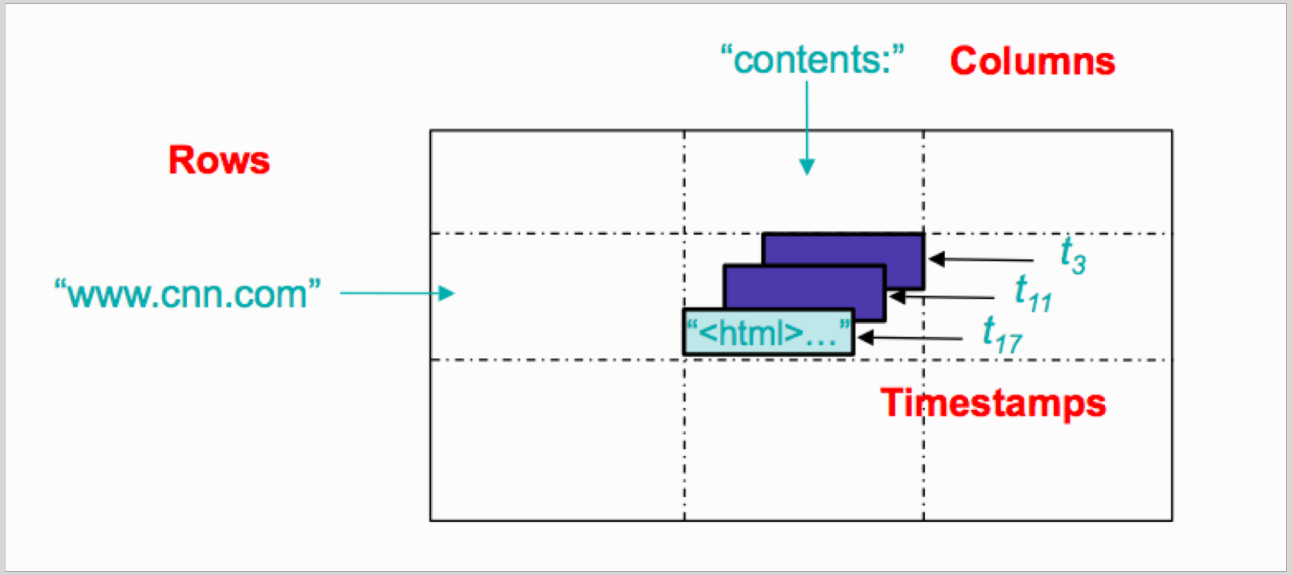
  (row:string, column:string, time:int64) → string

- Row :
  - Row keys in a table are arbitrary strings
  - Data is maintained in lexicographic order by row key
  - Each row range is called a tablet, which is a unit of distribution and load balancing.

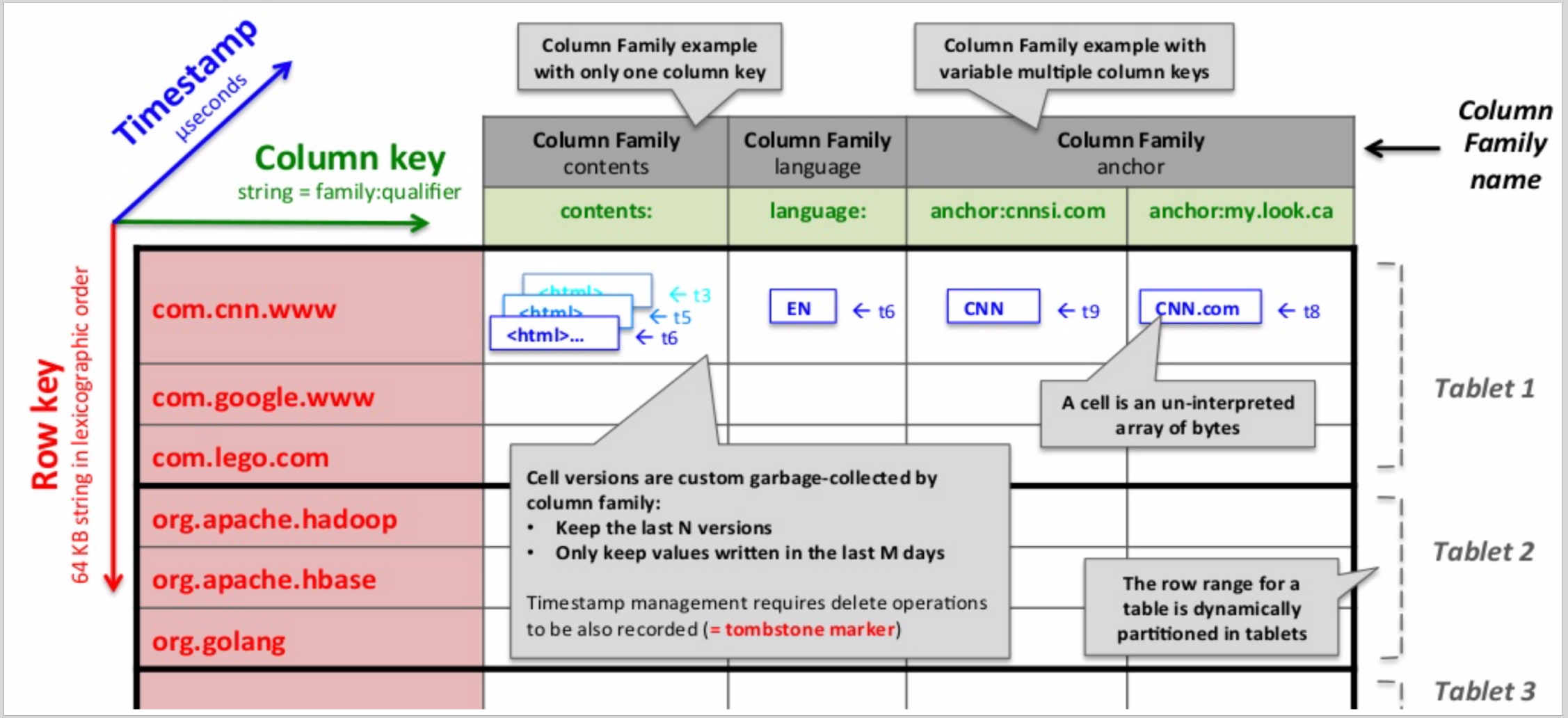


- Column
  - Column keys are grouped into sets called *column families.*
  - Data stored in column family is usually of the same type.
  - A column key is named using the syntax : *family: qualifier.*

- Timestamp
  - Each cell in data table can contain multiple versions of the same data.
  - Versions are indexed by 64-bit integer timestamps.
  - Timestamps are assigned:
    - Automatically by Bigtable , or ,
    - Explicitly by client applications



Data model


Timestamp
µseconds
Column key
string = family:qualifier
Row key
64 KB string in lexicographic order
Column Family example with only one column key
Column Family example with variable multiple column keys
Column Family name
Column Family contents
Column Family language
Column Family anchor
contents:
language:
anchor:cnnsi.com
anchor:my.look.ca
com.cnn.www
com.google.www
com.lego.com
org.apache.hadoop
org.apache.hbase
org.golang
<html>...
t3
t5
t6
EN
t6
CNN
t9
CNN.com
t8
A cell is an un-interpreted array of bytes
Cell versions are custom garbage-collected by column family:
• Keep the last N versions
• Only keep values written in the last M days
Timestamp management requires delete operations to be also recorded (= tombstone marker)
The row range for a table is dynamically partitioned in tablets
Tablet 1
Tablet 2
Tablet 3

Bigtable API's

- Provides following functions
  - Creating and deleting tables and column families

```
CreateTable(table) / DeleteTable(table)
CreateColumnFamily(columnFamily) / DeleteColumnFamily(columnFamily)
```

  - Changing cluster , table and column family metadata

```
SetTableFlag(table, flags) / . . .
SetColumnFamilyFlag(table, colfamily, flags) / . . .
```

  - Allows cells to be used as integer counters

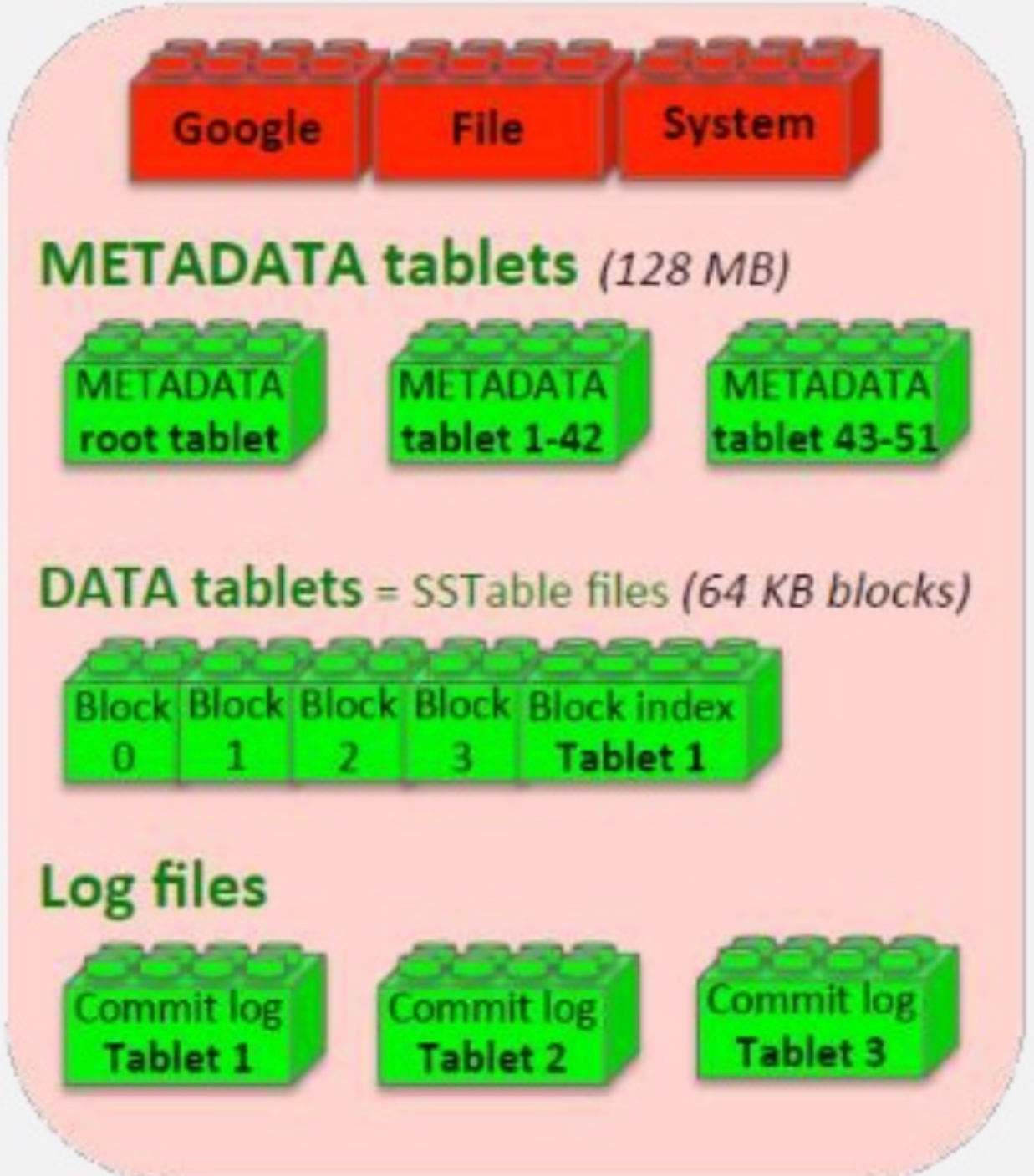
```
Increment(rowkey, columnkey, increment)
```

  - Client supplied scripts could be executed in the address space of servers

Sawzall

Building Blocks

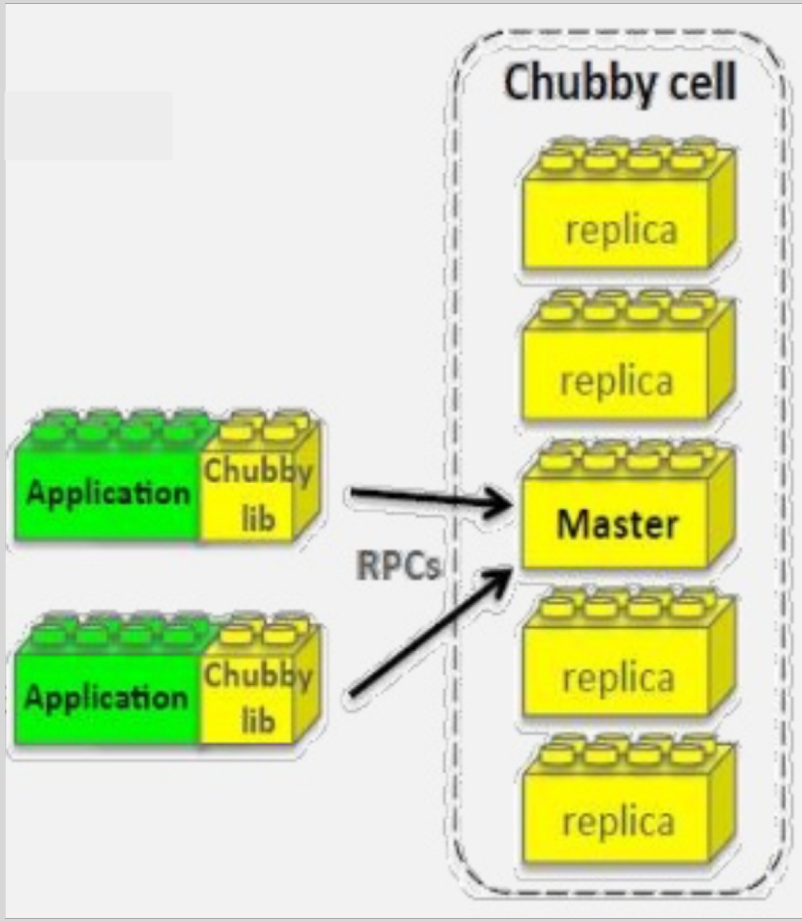
- Bigtable is built on several other pieces of Google infrastructure
  - Google File System (GFS)
    - Used to store log and data files
  - SSTable : Data Structure for storage
    - Used to store table data in GFS
    - Used to store and retrieve the pairs < Key, Value >
    - Used as pointers to pairs < Key, Value > stored in GFS
  - Chubby : Distributed lock-service



Chubby

- Chubby is highly available and persistent distributed lock service
- Chubby service consists of 5 active replicas with one master to serve requests
- Each directory/file can be used as a lock
- Each clients has a session with Chubby.
- The session expires if it is unable to renew its session lease within the lease expiration time.
- Also an OSDI ’06 Paper

Chubby unavailable = Bigtable unavailable



Implementation

- Three major components

  - Library linked to every client

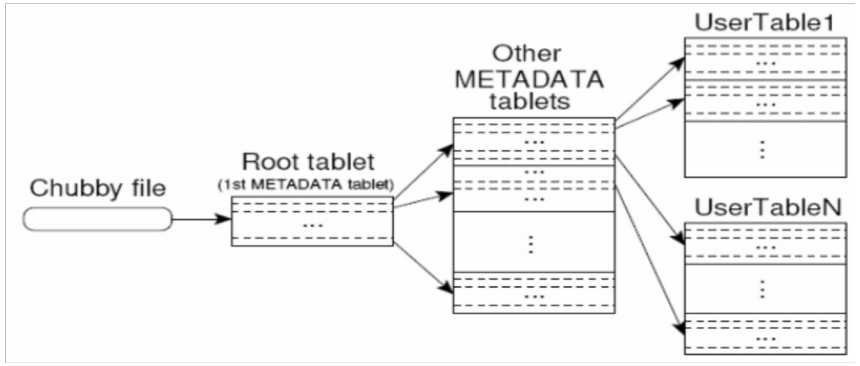
  - Single master server
    - Assigning tablets or tablet servers
    - Detecting addition and expiration of tablet servers
    - Balancing tablet-server load
    - Garbage Collection files in GFS

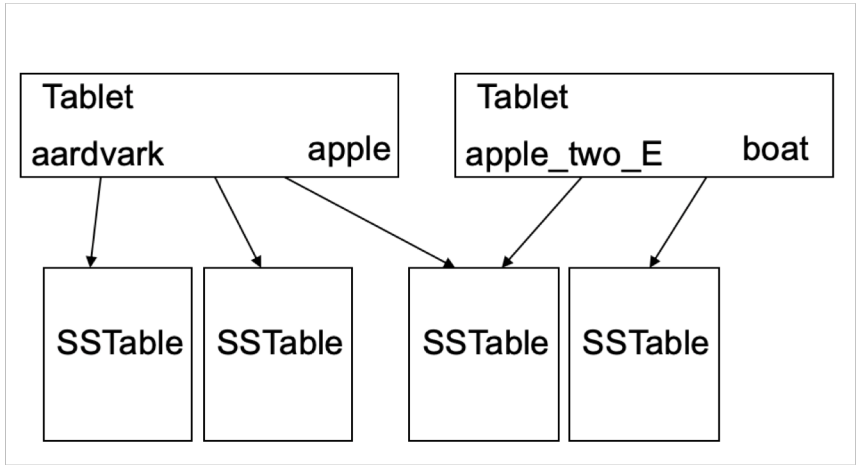
  - Many Tablet servers
    - Manages a set of tablets
    - Tablet servers handles read and write requests to its table
    - Split tablets that have grown too large

Locating Tablets

- Three – level hierarchy
  - Level 1 : Chubby file containing location of the root tablet
  - Level 2 : Root tablet contains the location of Metadata tablets
  - Level 3 : Each METADATA tablet contains the location of user tablets

  Location of a tablet is stored under Row key that encodes table identifier and it's end row.



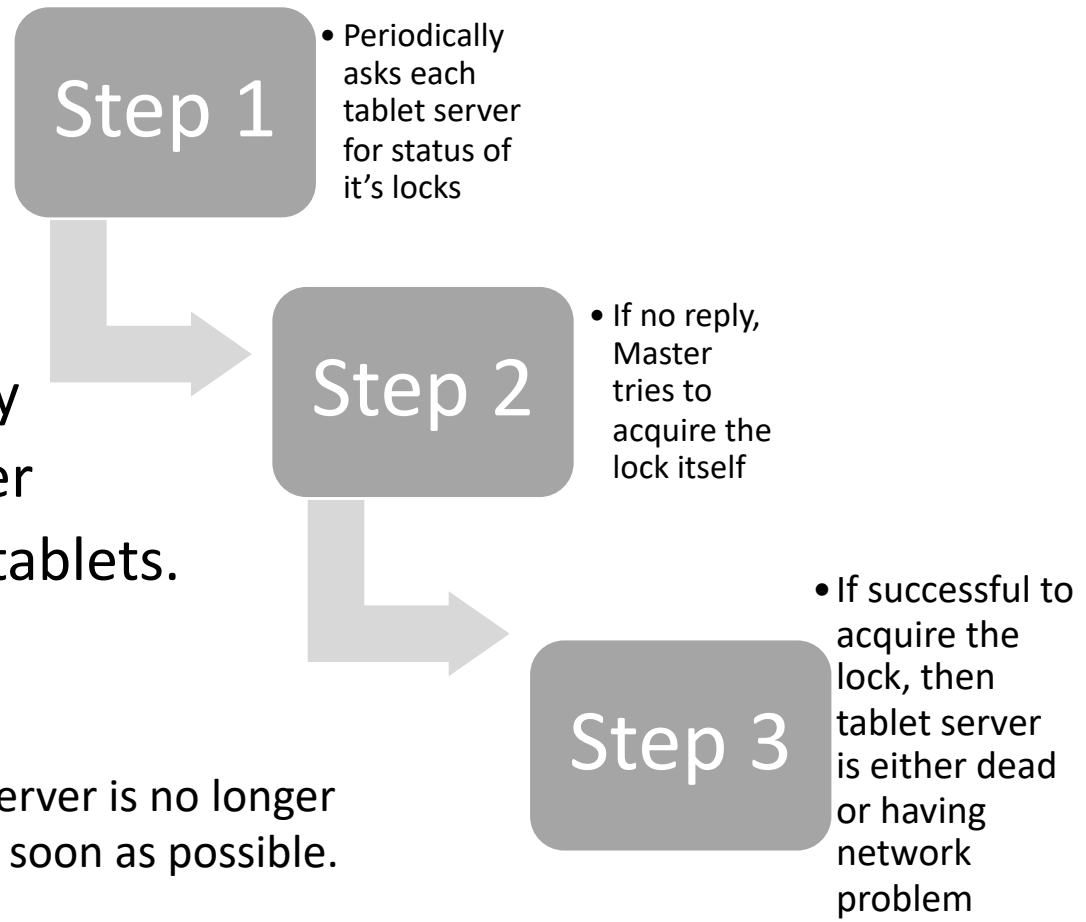


Assigning Tablets

- Tablet server startup
  - It creates and acquires an exclusive lock on a uniquely named file on Chubby.
  - Master monitors this directory to discover tablet servers.
- Tablet server stops serving tablets if..
  - It loses its exclusive lock.
  - File no longer exists, the tablet server will never be able to serve again.

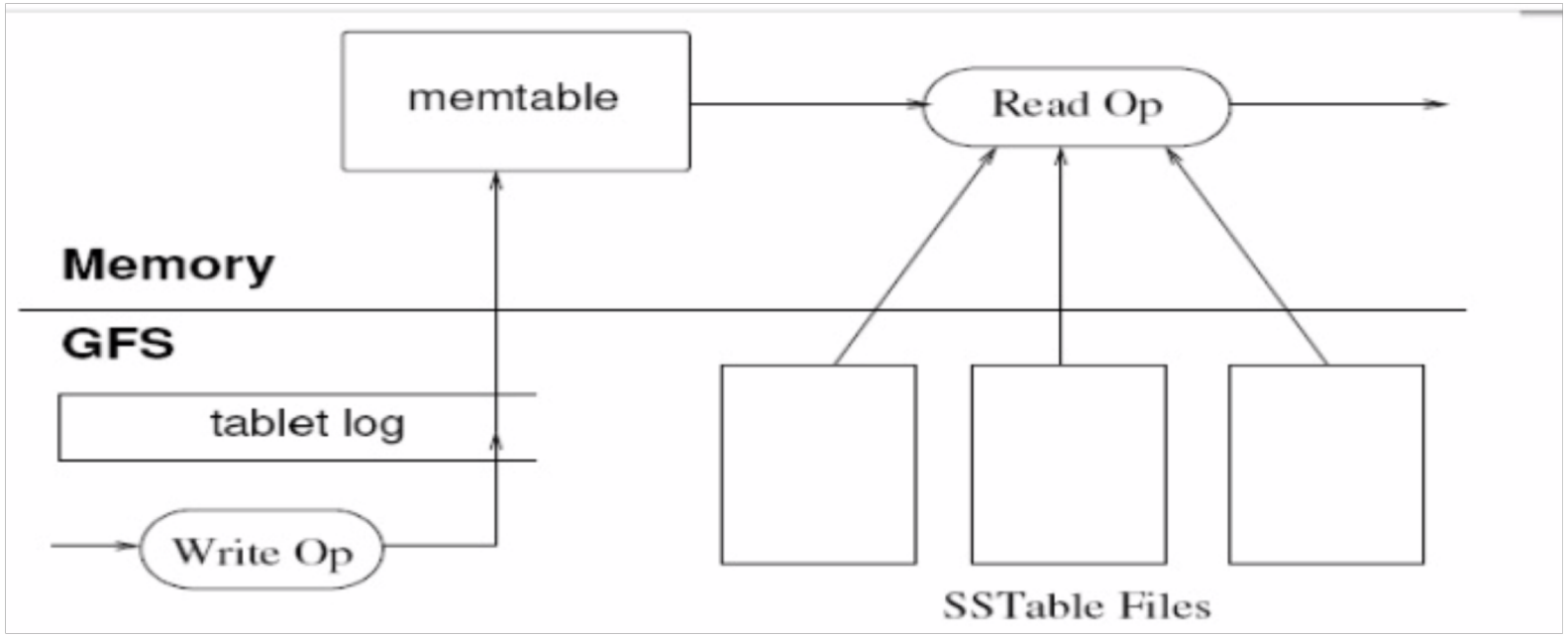
Assigning Tablets

- Master Server Startup
  - Grabs unique master lock in Chubby
  - Scans the tablet server directory in Chubby
  - Communicates with every live Tablet server
  - Scans the METADATA table to learn set of tablets.



Master is responsible for finding when the tablet server is no longer serving it's tablets and reassigning those tablets as soon as possible.

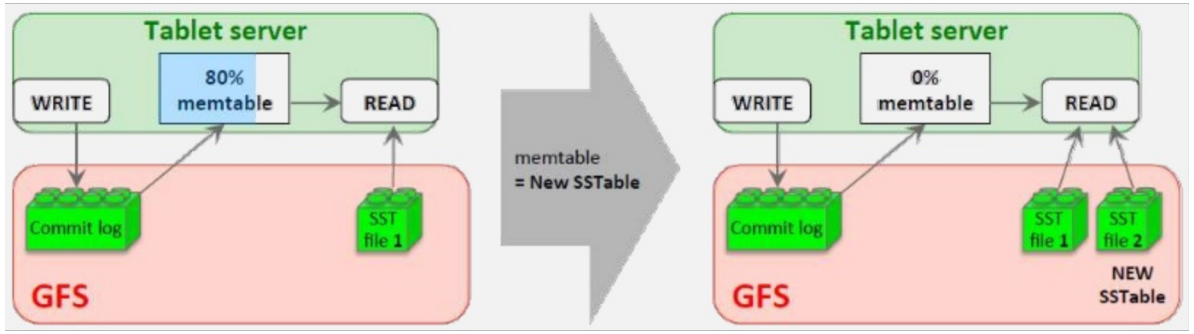
Tablet Serving

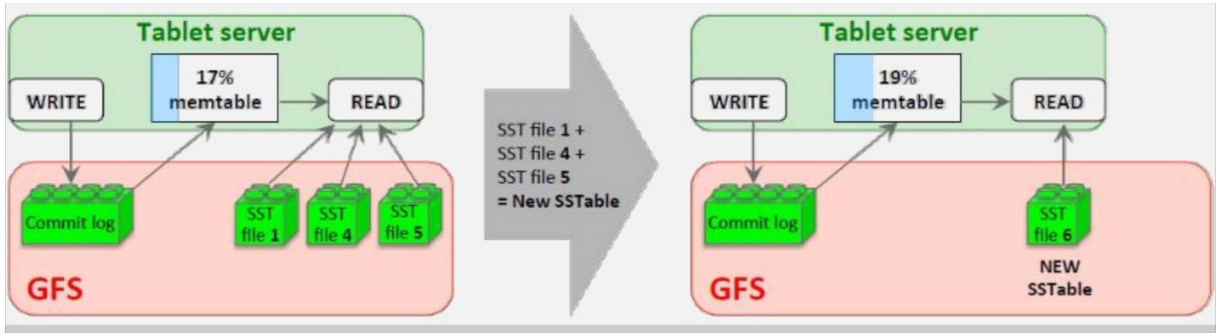


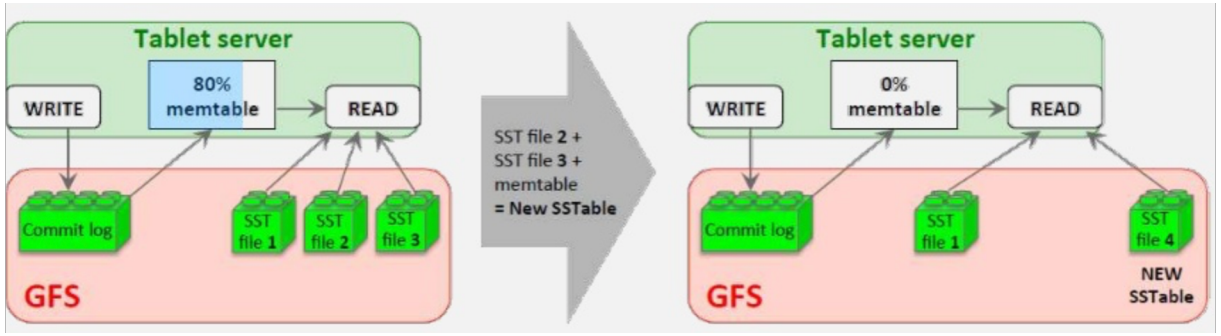
Write Operation	Read Operation
• Server checks if it is well-performed • Checks if the author is authorized • Writes to commit log • After commit, contents are inserted into memtable	• Checks well-formedness of request • Checks authorization in Chubby file • Merge memtable and SSTable to find data • Return data

Compaction

- Minor compaction – convert the memtable into an SSTable
  - Reduce memory usage
  - Reduce log traffic on restart
- Merging compaction
  - Reduce number of SSTables
  - Good place to apply policy “keep only N versions”
- Major compaction
  - Merging compaction that results in only one SSTable
  - No deletion records, only live data







Refinements

Locality Groups

- Clients can group multiple column families together into *locality groups*

Compression

- Compression applied to each SSTable block separately

Bloom Filters

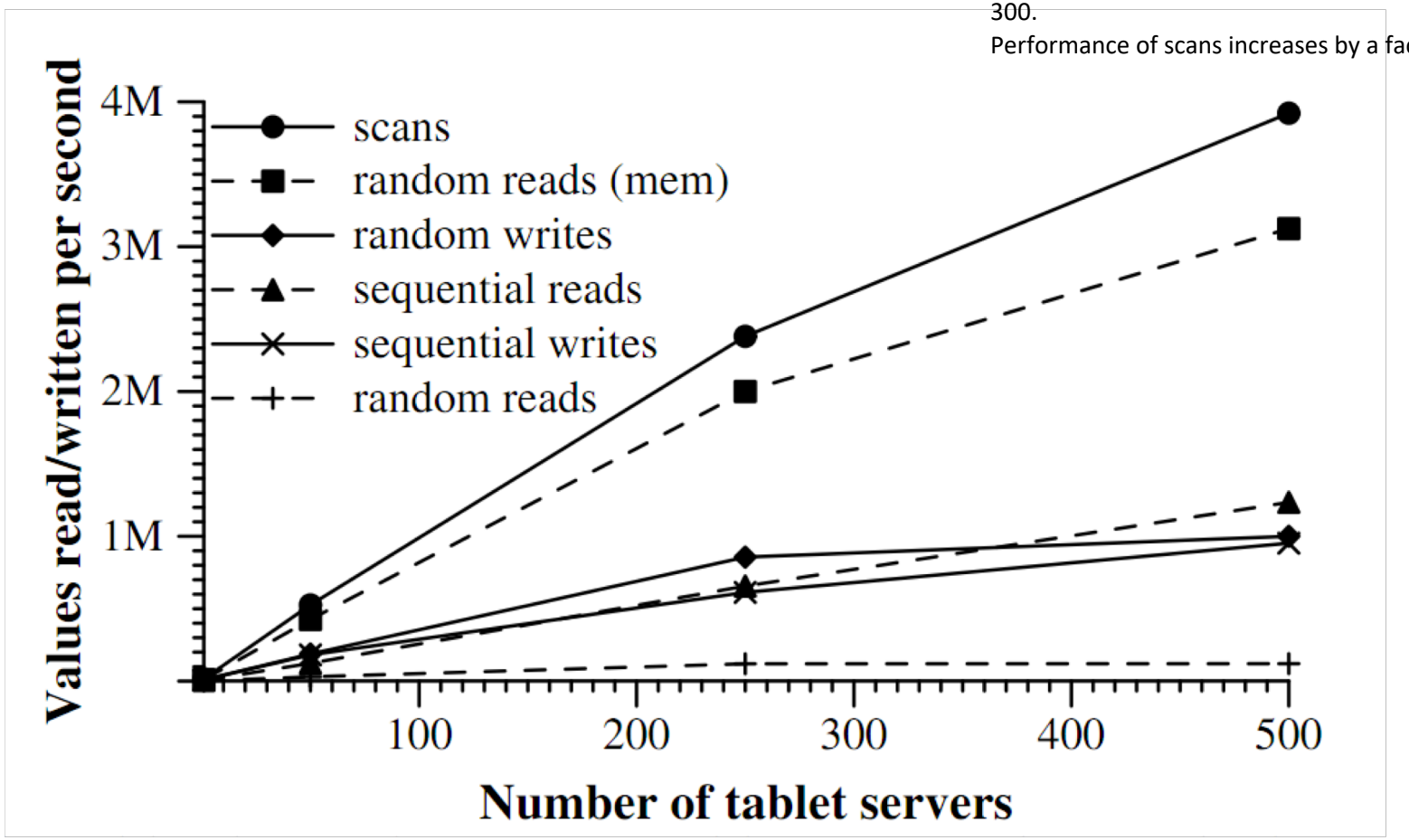
- Reduce the number of disk accesses

Caching

- Caching SSTables for a better peroformance

Performance Evaluation

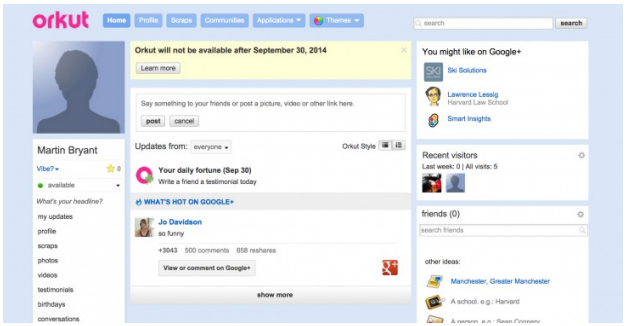
> As the number of tablet servers is increased by a factor of 500:
Performance of random reads from memory increases by a factor of 300.
Performance of scans increases by a factor of 260.



Not Linear!
WHY?

Applications at Google

Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Crawl	800	11%	1000	16	8	0%	No
Crawl	50	33%	200	2	2	0%	No
Google Analytics	20	29%	10	1	1	0%	Yes
Google Analytics	200	14%	80	1	1	0%	Yes
Google Base	2	31%	10	29	3	15%	Yes
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No
Orkut	9	–	0.9	8	5	1%	Yes
Personalized Search	4	47%	6	93	11	5%	Yes



TakeAways

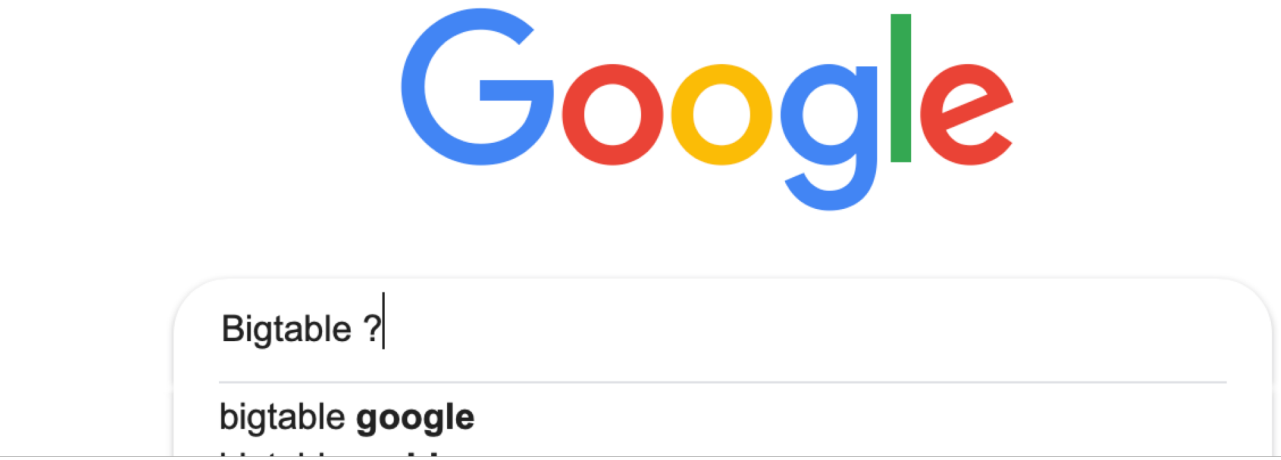
- Lessons Learnt
  - Large distributed systems are vulnerable to many types of failure
  - Importance of proper system-level monitoring
  - The value is in simple designs

- Conclusions
  - 7 years on design and implementations, in production since April 2005
  - +16 projects were using Bigtable (August 2006)

Performance and high availability

Scaling capabilities by simply adding more machines

SQL users are sometimes uncertain of how to best use Bigtable interfaces



Thankyou !