Google Bigtable

Agenda

- Bigtable

What is Bigtable?

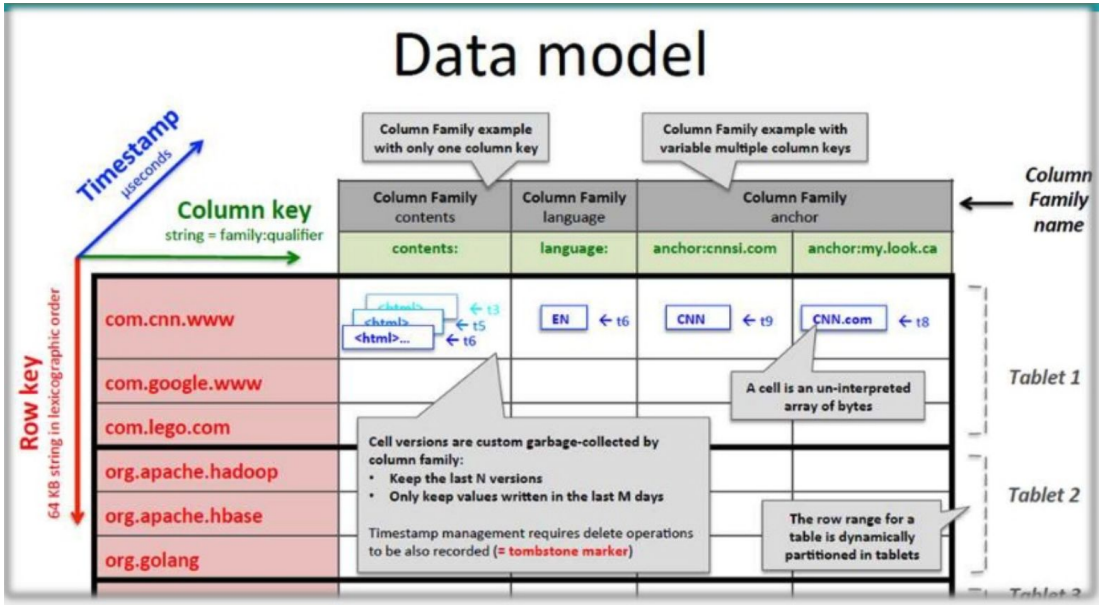
Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers.

Goals

- Wide Applicability
  - Google Analytics
  - Google Finance
  - Google Earth
- Scalability
- High Performance
- High Availability
  - **Locality groups**
  - **Compression**
  - **Bloom filters**
  - **Commit-log implementation**
  - **Speeding up tablet recovery**

Data Model

A Bigtable is a **sparse**, **distributed**, **persistent multi-dimensional sorted map**. The map is **indexed by a row key, column key, and a timestamp**; each value in the map is an uninterpreted array of bytes.



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

Building Blocks

Bigtable is composed of several other innovative, distributed oriented components.

- **GFS**
  - store log and data files
- **SSTable**
  - used to store tablet data in GFS
- **Chubby**
  - to ensure that there is at most one active master at any time
  - to store the bootstrap location of Bigtable data
  - to discover tablet servers and finalize tablet server deaths
  - to store Bigtable schema information
  - to store access control lists
- **Borg**
  - schedule jobs
  - manage resources
  - deal with machine failures
  - monitor machine status

GFS

- one *Master* node
  - namespace
  - access control information
  - control information, the mapping from files to chunks
  - the current locations of chunks
  - chunk lease management
  - garbage collection
  - chunk migration between chunkservers
  - monitor chunkserver status
- a large number of *Chunkservers*

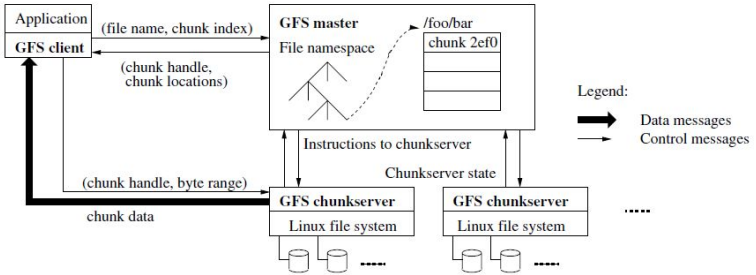
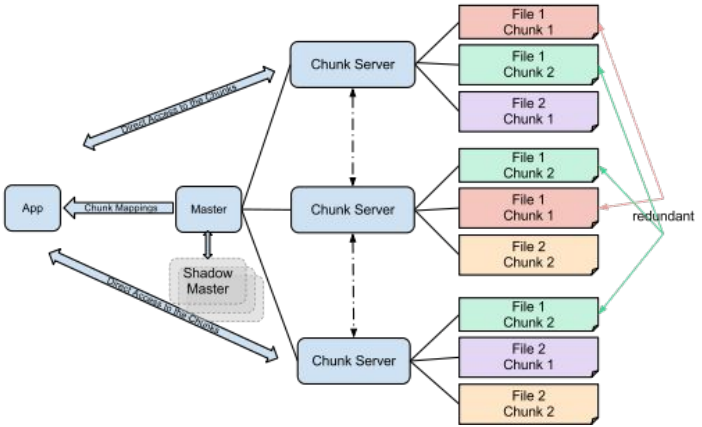
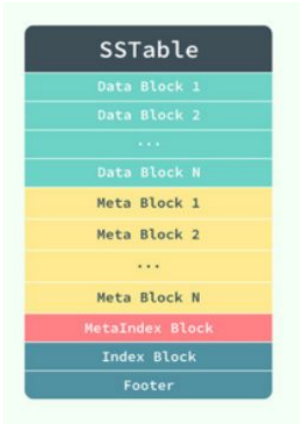


Figure 1: GFS Architecture



SSTable(Sorted Strings Table)

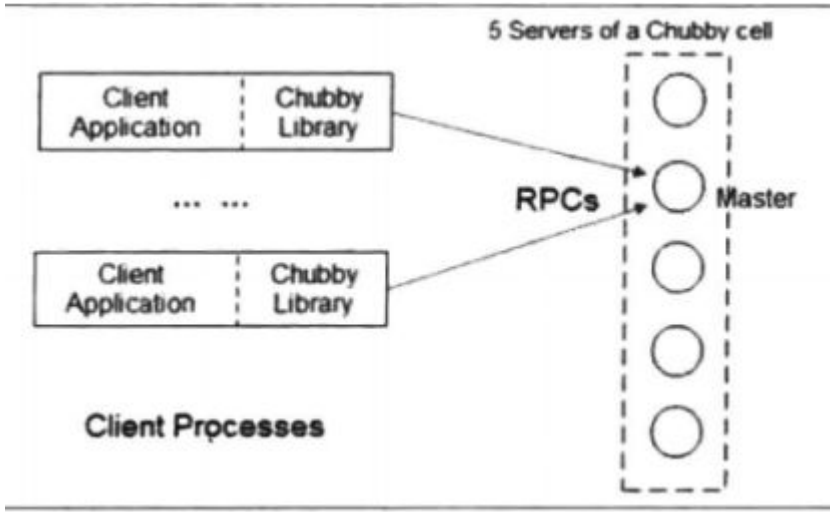
需要高效地存储大量的键-值对数据
数据是顺序写入
要求高效地顺序读写
没有随机读取或者对随机读取性能要求不高



Chubby

a distributed lock manager based on paxos algorithm developed by google.

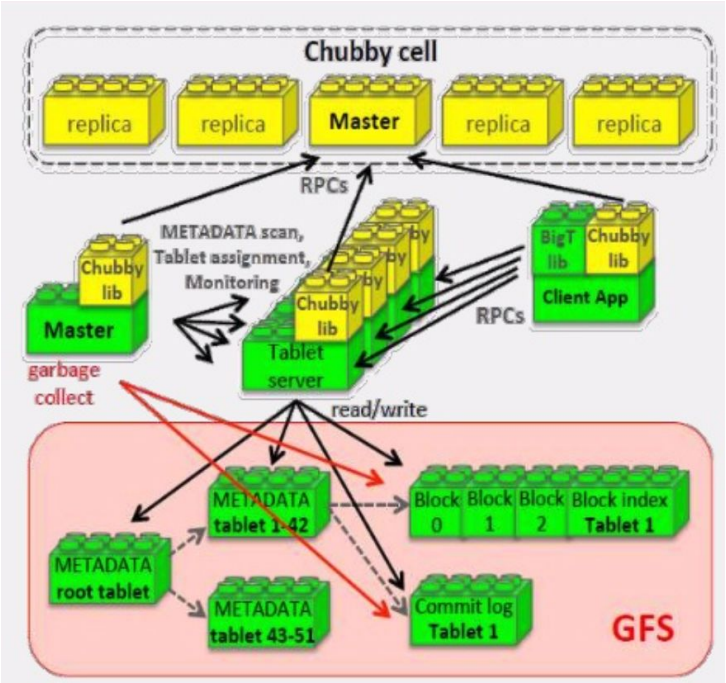
- 提供一个完整的、独立的分布式锁服务，而非仅仅是一个一致性协议的客户端库
- 提供粗粒度的锁服务
- 在提供锁服务的同时提供对小文件的读写功能
- 高可用、高可靠
- 提供事件通知机制



BigTable

The implementation has three major components

- **One** Master server
- **Many** tablet servers
- A **library** is linked into every client



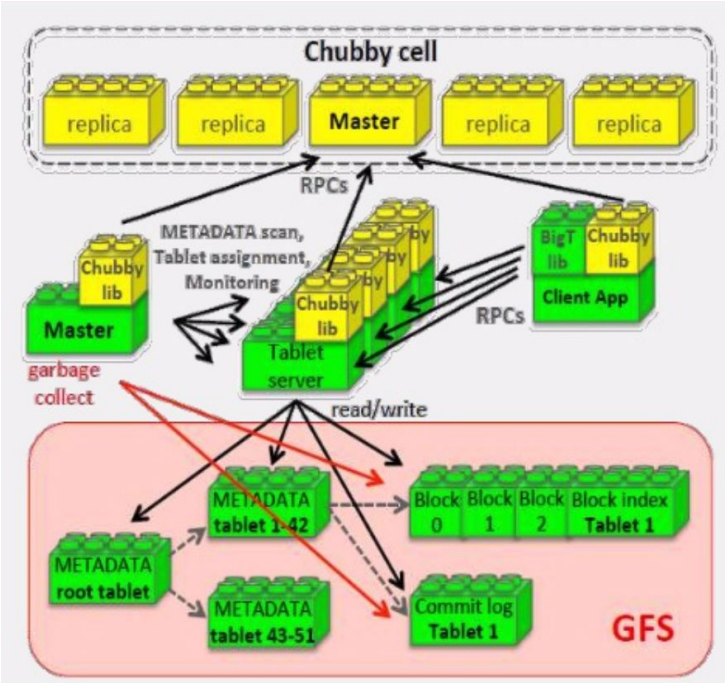
Bigtable runs over Google File System.

Bigtable is store in a structure called SSTable. Each SSTable is divided into 64KB blocks. A SSTable can be loaded to memory.

BigTable

One Master server:

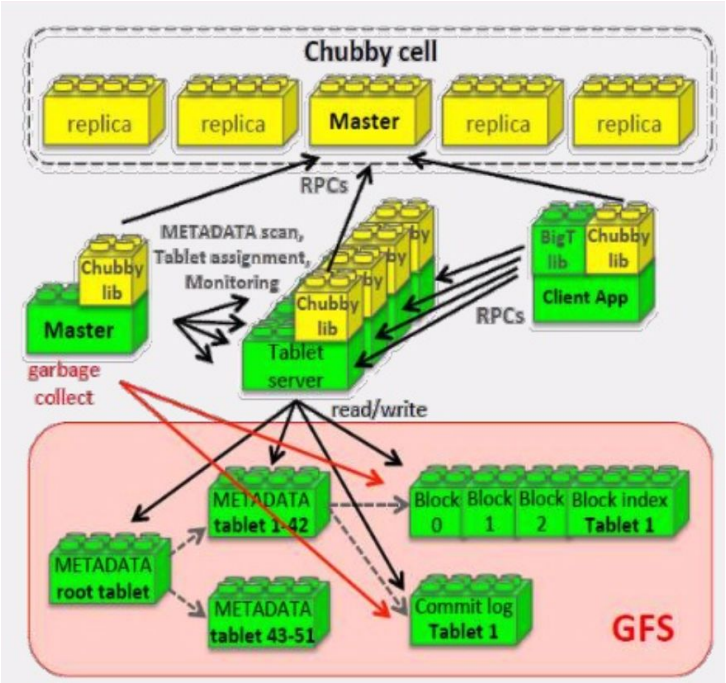
- Assigning tablets to tablet servers
- Detecting addtion and expiration of tablet servers
- Balancing tablet server load
- Garbage collecting of files in GFS
- Handling schema changes



BigTable

Many tablet servers:

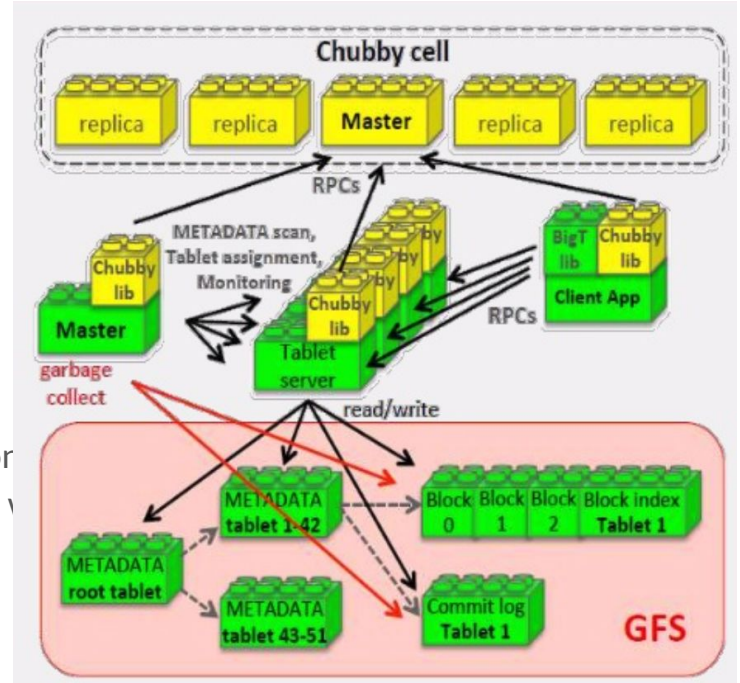
- Manages a set of tablets
- Handles read and write request to the tablets
- Splits tablets that have grown too large



BigTable

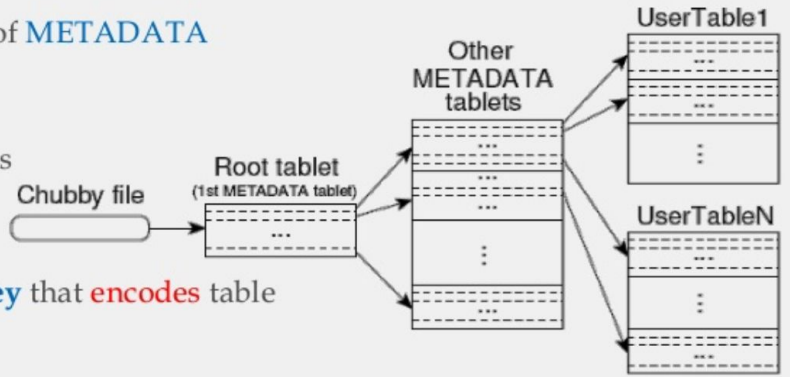
A **library** is linked into every client

- Do not relay on the master for tablet location information
- Communicates directly with tablet servers for reads and w



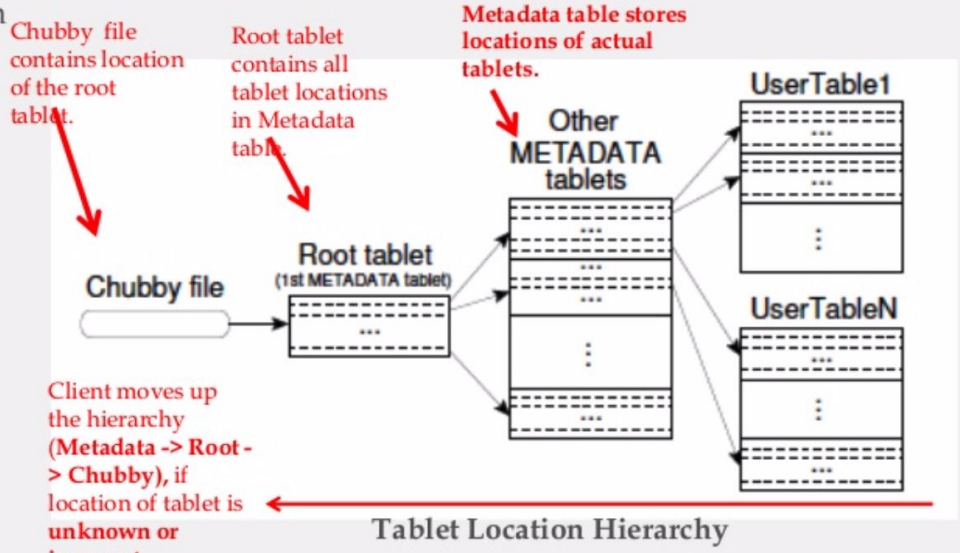
Tablet Location

- 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 tablet is stored under a **row key** that encodes table identifier and its end row



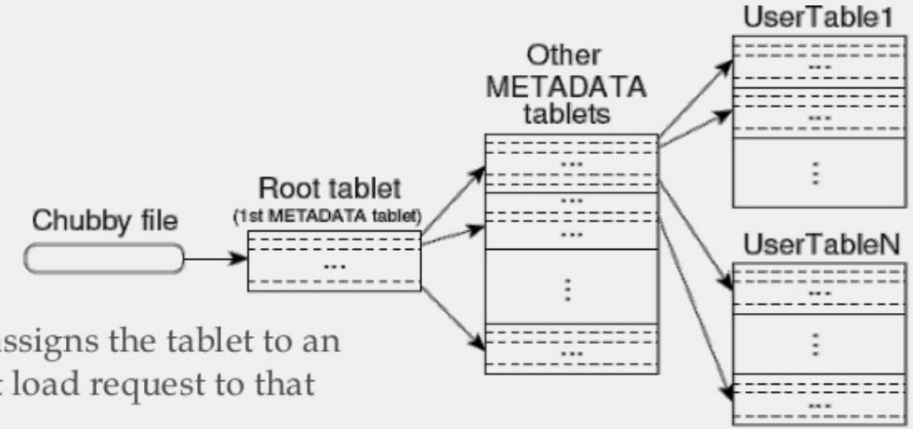
Tablet Location

- Lookup is a **three-level system.**
- **Benefit :- NO Big Bottleneck** in the system and it also make heavy use of **Pre-Fetching** and **Caching**



Tablet Assignment

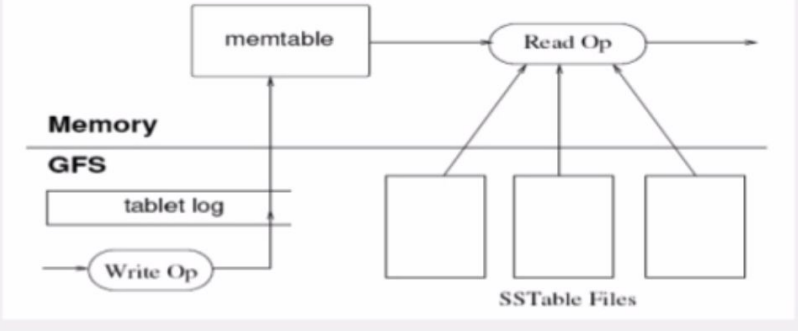
- Each tablet is assigned to **one** tablet server at a **time**
- **Master** keeps tracks of
  - the set of live tablet servers (tracking via Chubby)
  - the current assignment of tablet to tablet servers
  - the current unassigned tablets



- When a tablet is unassigned, the **master** assigns the tablet to an available tablet server by sending a tablet load request to that tablet server

Tablet serving

- Updates committed to a **commit log**
- Recently committed updates are stored in memory –**MEMtable**
- Older updates are stored in a sequence of **SSTables**.



Tablet serving

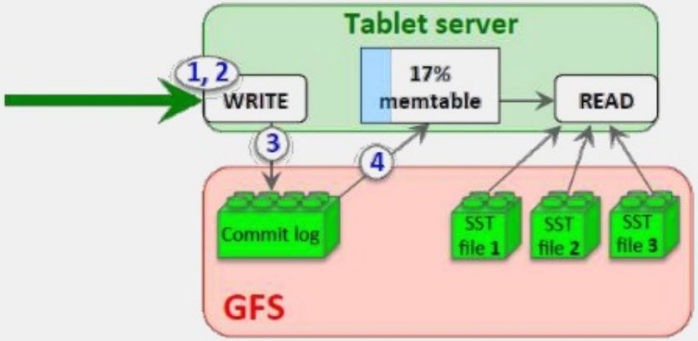
- **Write operation:**

1. Server checks that the request is **well-formed**

2. Server checks that the sender is **authorized** to write (list of permitted writers in a Chubby file)

3. A **valid mutation** is written to the commit log that stores redo records (group commit to improve throughput)

4. After the mutation has been committed, its **contents** are inserted into the **MEMtable** (= in memory sorted buffer)



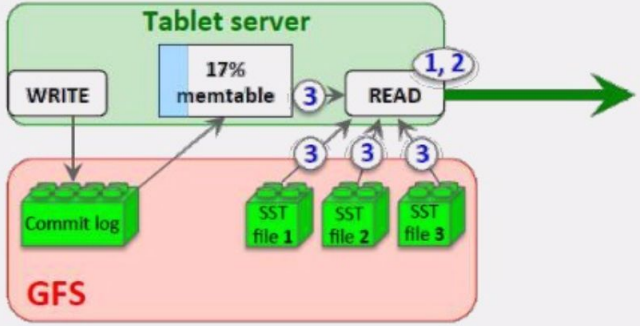
Tablet serving

- **Read operation:**

1. Server checks that the request is **well-formed**

2. Server checks that the sender is **authorized** to read (list of permitted writers in a Chubby file)

3. **Valid read operation** is executed on a merged view of the sequence of SSTables and the MEMtable

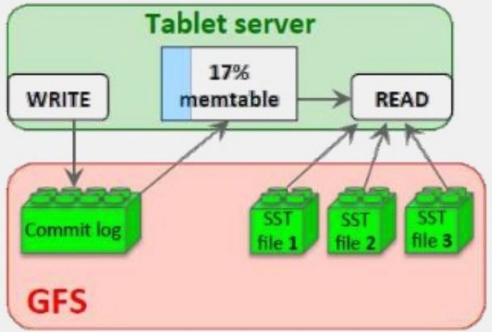


Tablet serving

- **Tablet Recovery**

1. Tablet server reads its metadata from the METADATA table (lists of SSTables that comprise a tablet and a set of a redo points, which are pointers into any commit logs that may contain data for the tablet)

2. The tablet server reads the indices of the SSTables into memory and reconstructs the MEMtable by applying all of the updates that have a commted since the redo points



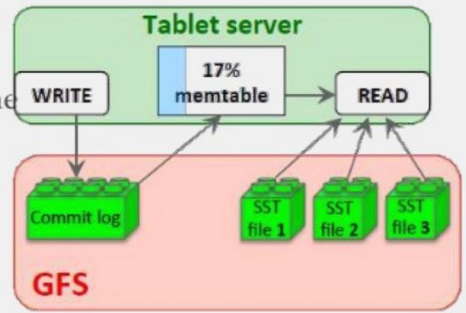
Compaction

- In order to control size of MEMtable, tablet log, and SSTable files, "compaction" is used.

1. **Minor Compaction** - Move data from **MEMtable** to **SSTable**.

2. **Merging Compaction** - Merge multiple **SSTables** and **MEMtable** to a single **SSTable**.

3. **Major Compaction** - that re-writes all **SSTables** into exactly one **SSTable**

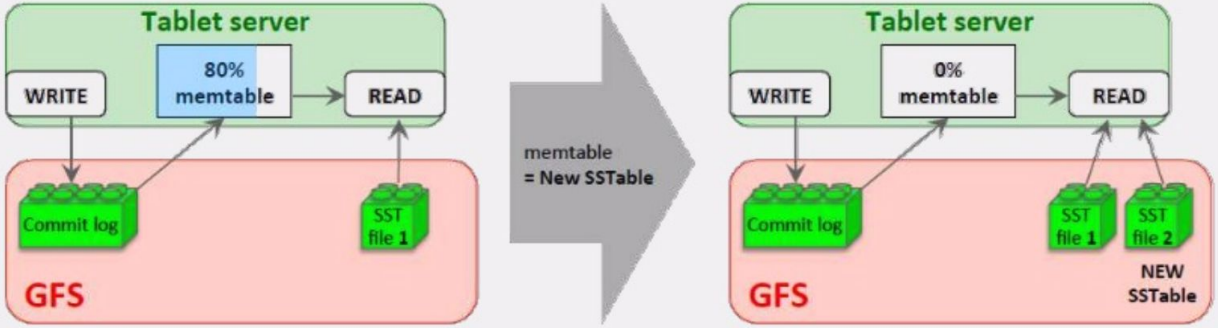


Compaction

1. **Minor Compaction**

– When MEMtable size reaches a threshold, MEMtable is frozen, a new MEMtable is created, and the frozen MEMtable is converted to a new SSTable and written to GFS

– Two goals: shrinks the memory usage of the tablet server, reduces the amount of data that has to be read from the commit log during a recovery

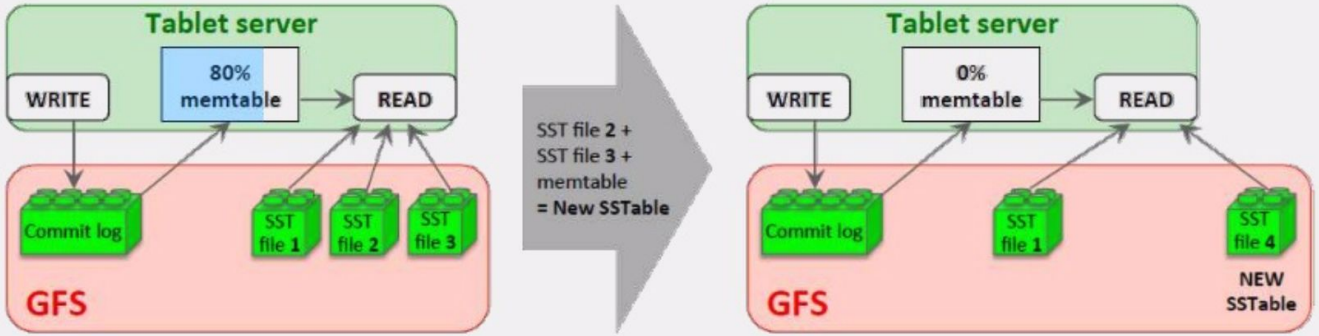


Compaction

2. Merging Compaction

– **Problem**: every minor compaction creates a **new SSTable** (→arbitrary number of SSTables !)

– **Solution**: periodic merging of a few **SSTables** and the **MEMtable**

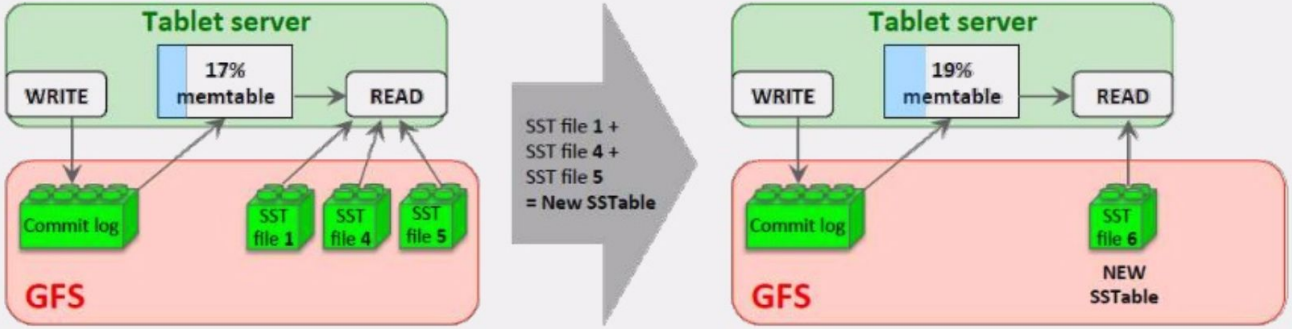


Compaction

3. **Major Compaction**

– It is a merging compaction that rewrites all SSTables into exactly **one SSTable that contains no deletion information or deleted data**

– BigTable cycles through all of it tablets and regularly applies major compaction to them (=reclaim resources used by deleted data in a timely fashion)



Refinements

- tablet-server-side write-through cache
    - o scan cache: high-level key/value
    - o cache blockcache: GFS block cache
    - o why no data cache in client library?
- SSTable per “locality group” – a set of column familieso excludes from reads unrelated columns
- SSTables can be compressed
    - o 10-1 reduction in space and thus improvement in logical disk bandwidth
    - o decompression presumably done on server-side? no network bandwidth benefits!
- bloom filter
    - o explain what a bloom filter is
    - o in-memory bloom filter filters out most lookups for non-existent rows/columns

Questions

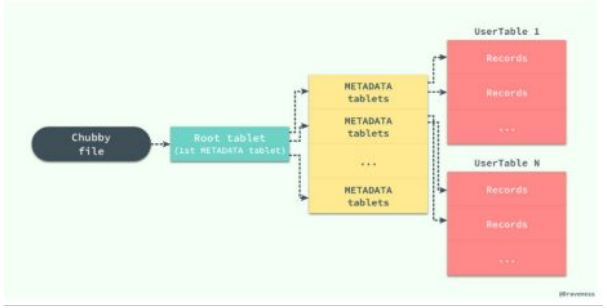
为什么Bigtable设计成Root、Meta、User三级结构，而不是两级或者四级结构？

假定文件的大小为y，文件每条记录的长度为x，整个系统的容量为C，meta tablet的层次为N，文件数为M。

则满足：

$$C = y^N/x^{(N-1)}$$
$$M = (y/x)^{(N-1)}=C/y$$
$$y = (C*x^{(N-1)})^{(1/N)}$$

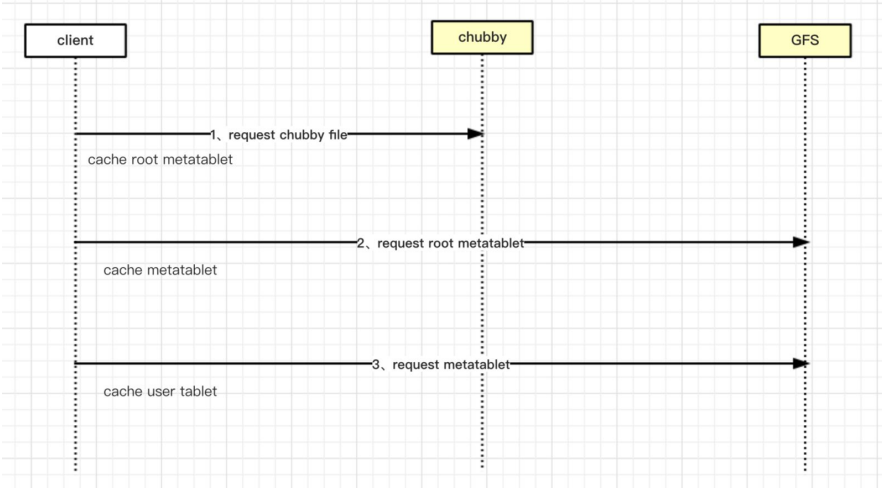
subjet to： C为PB级，文件数要考虑文件系统支持的文件数。

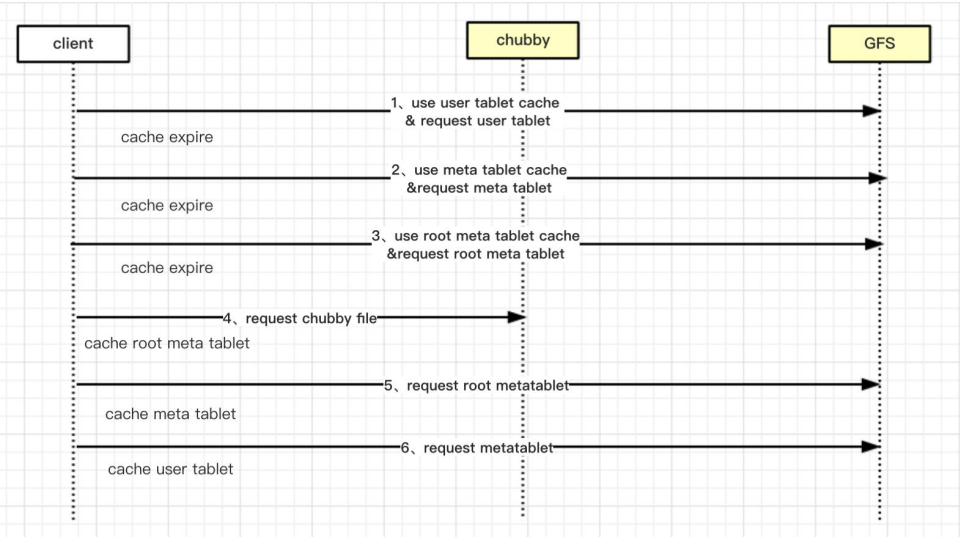


N	C	y	x	M
2	2048PB	2^36=64G	1KB	2^25=3400w
3	2048PB	2^27=128M	1KB	2^34=172亿
4	2048PB	2^23=8M	1KB	2^38=2800亿

Questions

读取某一行用户数据，最多需要几次请求？分别是什么？





Questions

如何保证同一个tablet不会被多台机器同 时服务？

1、master同一时刻只有一个。
2、master会维护那些tablet是已经分配，那些是没有 进行分配的。
3、master在探测那些tablet server是可以服 务的之后，负责下发指令。
4、对于master维护的tablet和tablet server的映射关系，同一 时刻只有一个 tablet server服务某个tablet。

Questions

如何设计SSTable的存储格式？

SSTable 中其实存储的不只是数据，其中 还保存了一些元数据、索引等信息，用于加速 读写操作的速度，虽然在 Bigtable 的论文中并没有 给出 SSTable 的数据格式，不过在 LevelDB 的实现中，我们可以发现 SSTable 是以这种格式存 储数据的。

Questions

如何使得tablet迁移过程停服务时间尽量短？(mimor compaction，减少读log)

1、master对需要迁移的tablet server发起指令，进行一次minor compaction。

2、需要迁移的tablet server停写，再进行一次minor compation。

3、新的tablet server就不需要读取log重建memtable。

Questions

如果tablet出现故障，需要将服 务迁移到其它机器，这个过程需要排序操作日志。如何 实现？

1、每个tablet服务器将本服 务器的所有tablet日志写入到一个文件。

2、对日志文件条目以 key<table, row name, log sequence number>进行排序。（局部特性）

3、每个日志文件切分 为64M大小文件，在各 tablet server进行并行排序。

4、某tablet server故障恢复的 时候，有master协同启动日志排序，然后故障机器 读取日志 进行重建。