Bigtable: A Distributed Storage System for Structured Data

By Fay Chang, et al. OSDI 2006

Presented by Xiang Gao

2014-11-05

Outline

- Motivation
- Data Model
- APIs
- Building Blocks
- Implementation
- Refinement
- Evaluation

Outline

- Motivation
- Data Model
- APIs
- Building Blocks
- Implementation
- Refinement
- Evaluation

Google’s Motivation

- Lots of data
  - Web contents, satellite data, user data, email, etc.
  - Different projects/applications
  - Billions of users
  - Many incoming requests
- Storage for structured data
- No commercial system big enough
- Low-level storage optimization help performance significantly

Bigtable

- Distributed multi-level map
- Fault-tolerant, persistent
- Scalable
  - Thousands of servers
  - Terabytes of in-memory data
  - Petabyte of disk-based data
  - Millions of reads/writes per second, efficient scans
- Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to load imbalance

Outline

- Motivation
- Data Model
- APIs
- Building Blocks
- Implementation
- Refinement
- Evaluation

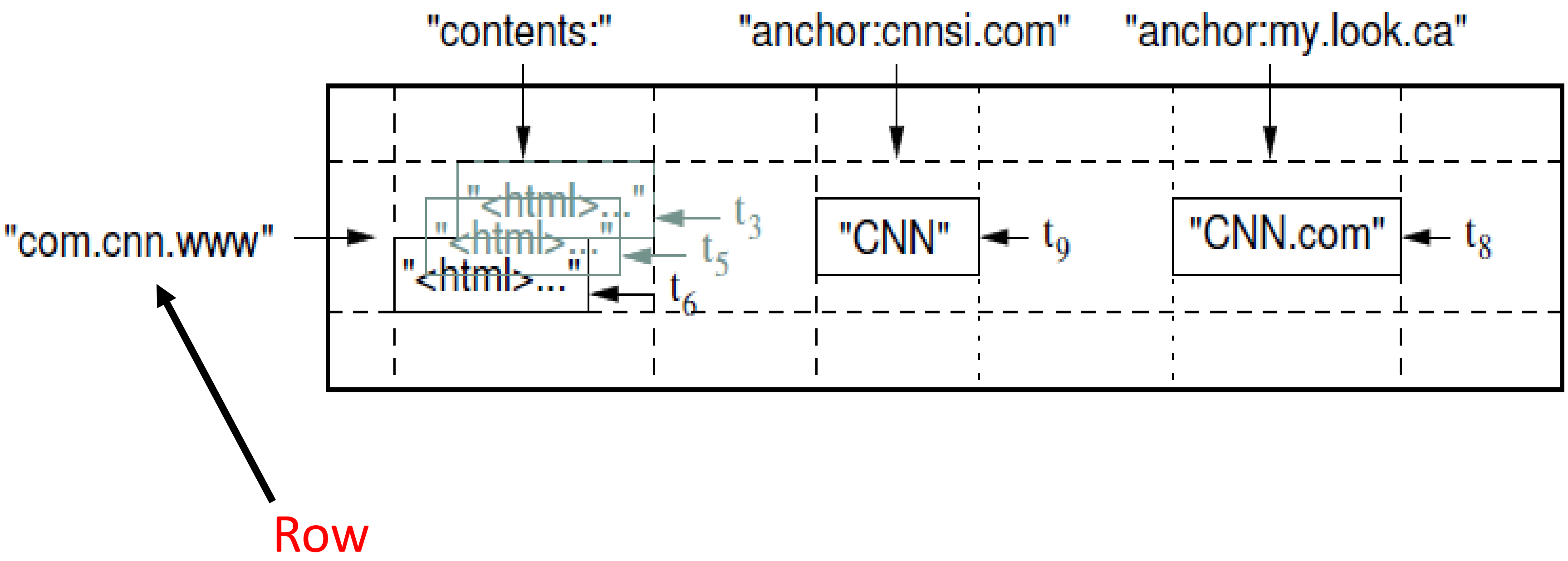
Data Model

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

(row, column, timestamp) -> cell contents

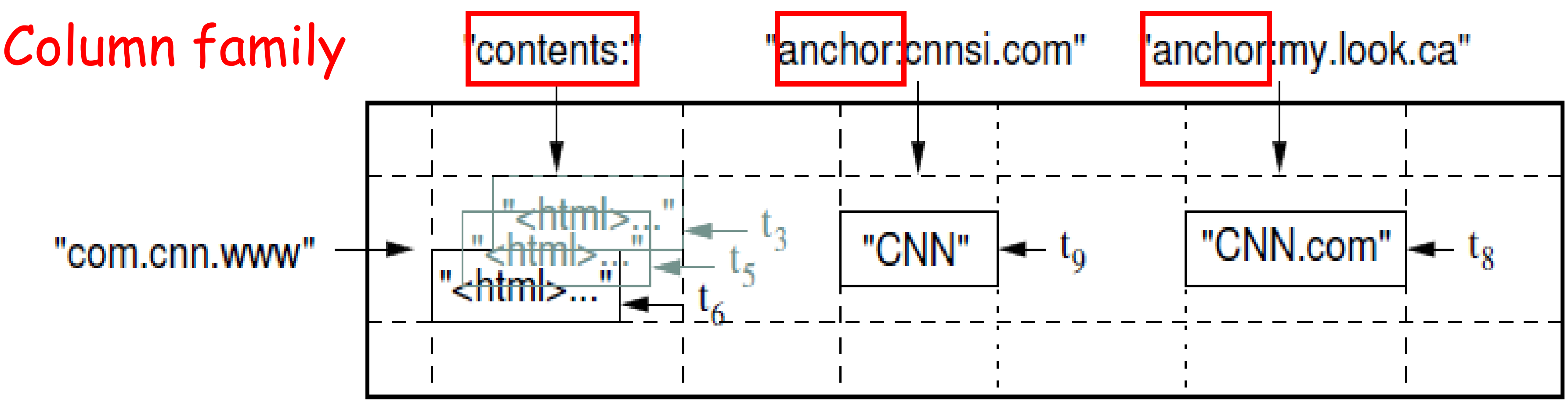
Data Model

- Rows
  - Arbitrary string
  - Access to data in a row is atomic
  - Ordered lexicographically



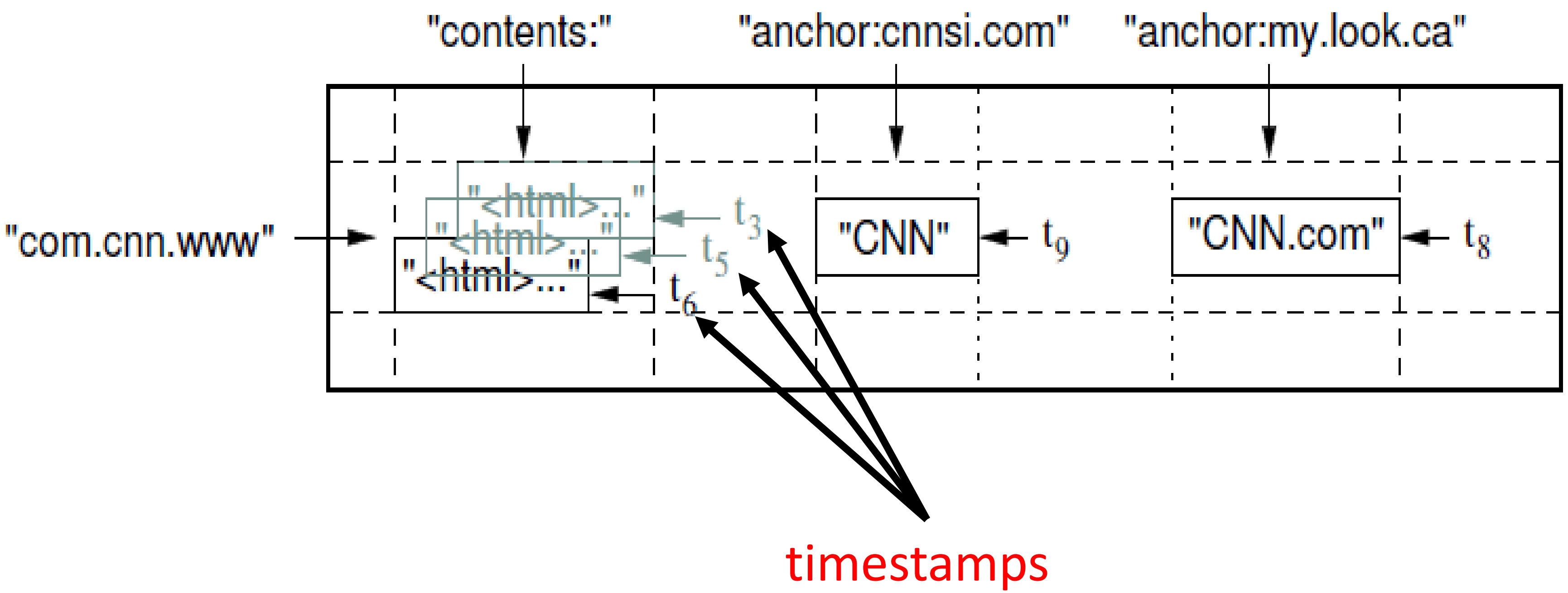
Data Model

- Column
  - Two-level name structure:
    - family: qualifier
  - Column Family is the unit of access control



Data Model

- Timestamps
  - Store different versions of data in a cell
  - Lookup options
    - Return most recent n versions
    - Return new enough versions



Data Model

- The row range for a table is dynamically partitioned
- Each row range is called a tablet
- Tablet is the unit for distribution and load balancing

Outline

- Motivation
- Data Model
- APIs
- Building Blocks
- Implementation
- Refinement
- Evaluation

APIs

- Metadata operations
  - Create/delete tables, column families, etc.
- Writes
  - Set(): write cells in a row
  - DeleteCells(): delete cells in a row
  - DeleteRow(): delete all cells in a row
- Reads
  - Scanner: read arbitrary cells in a bigtable
    - Each row read is atomic
    - Can restrict returned rows to a particular range
    - Can ask for just data from 1 row, all rows, etc.
    - Can ask for all columns, just certain column families, or specific columns

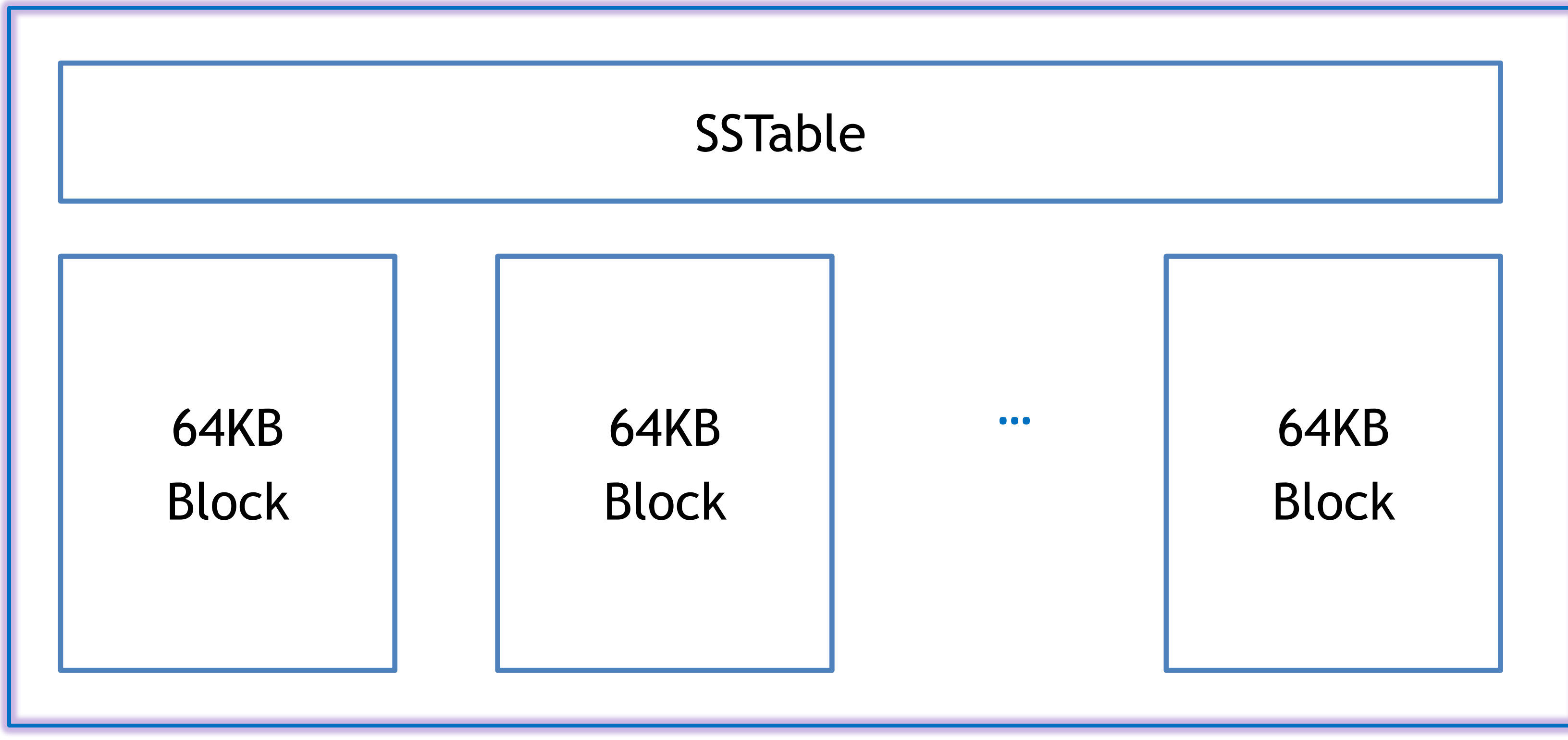
Outline

- Motivation
- Data Model
- APIs
- Building Blocks
- Implementation
- Refinement
- Evaluation

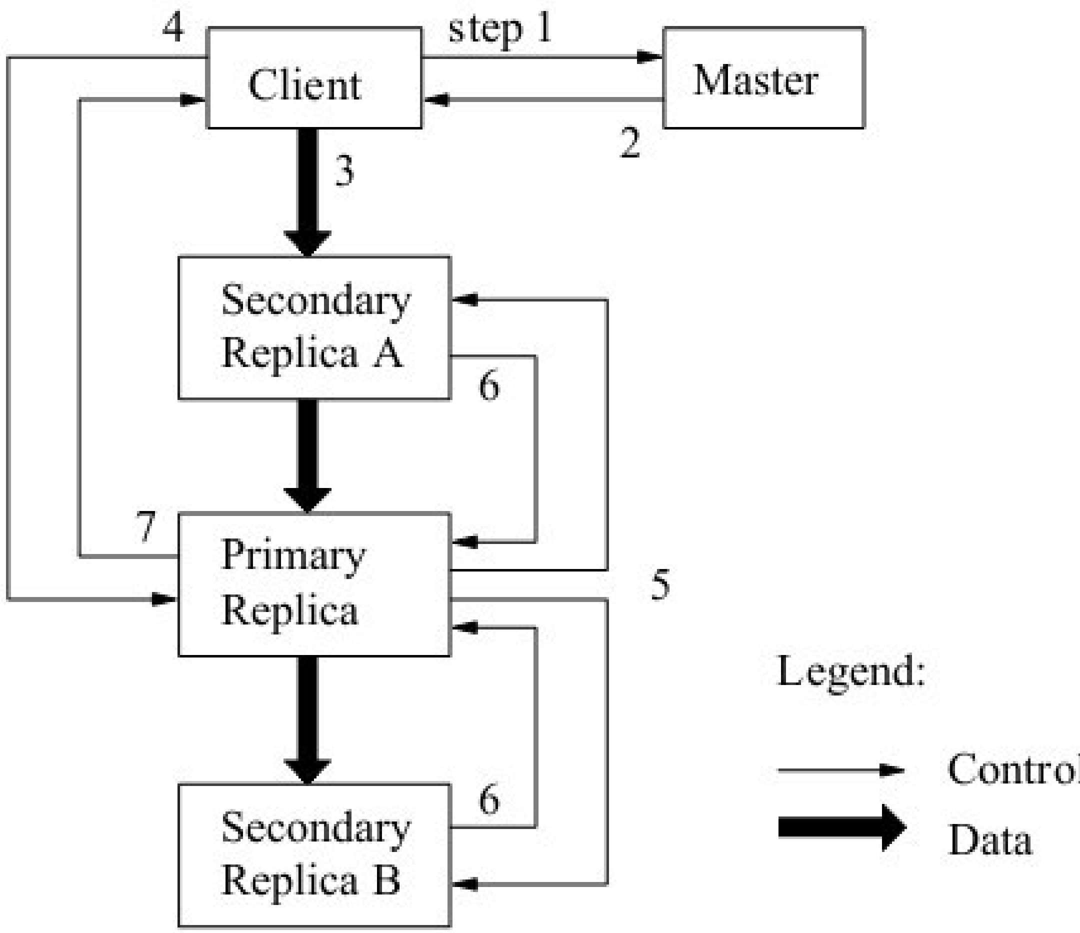
Building Blocks

- Bigtable uses the distributed Google File System (GFS) to store log and data files
- The Google SSTable file format is used internally to store Bigtable data
- An SSTable provides a persistent , ordered immutable map from keys to values
  - Each SSTable contains a sequence of blocks
  - A block index (stored at the end of SSTable) is used to locate blocks
  - The index is loaded into memory when the SSTable is open

Tablet and SSTables



GFS write flow



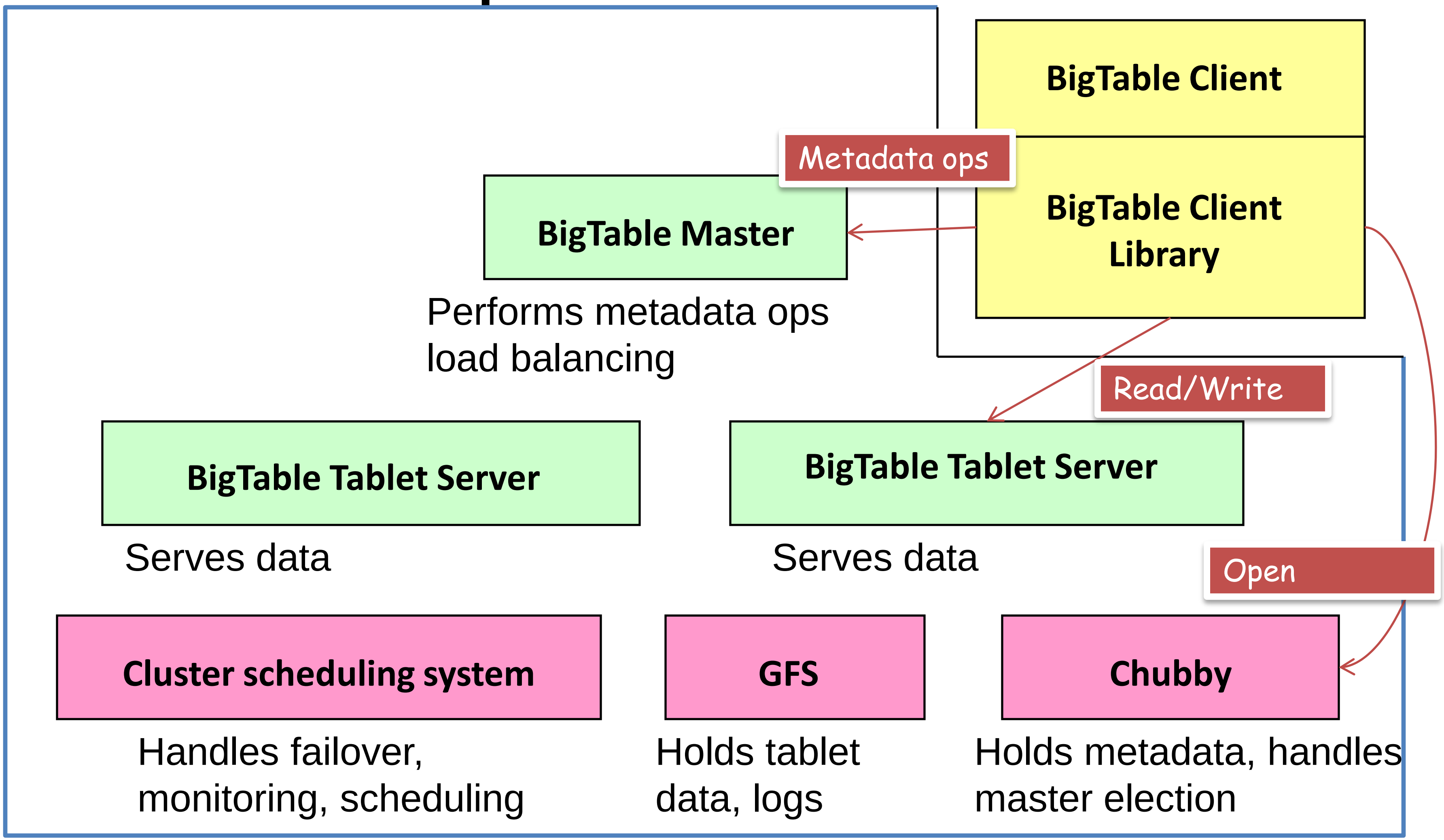
Outline

- Motivation
- Data Model
- APIs
- Building Blocks
- Implementation
- Refinement
- Evaluation

Implementation

- Single-master distributed system
- Three major components
  - Library that linked into every client
  - One master server
    - Assigning tablets to tablet servers
    - Detecting addition and expiration of tablet servers
    - Balancing tablet-server load
    - Garbage collection
    - Metadata Operations
  - Many tablet servers
    - Tablet servers handle read and write requests to its table
    - Splits tablets that have grown too large

Implementation



Master Operation

- Upon start up the master needs to discover the current tablet assignment.
  - Obtains unique master lock in Chubby
    - Prevents concurrent master instantiations
  - Scans servers directory in Chubby for live servers
  - Communicates with every live tablet server
    - Discover all tablets
  - Scans METADATA table to learn the set of tablets
    - Unassigned tablets are marked for assignment

Master Operation

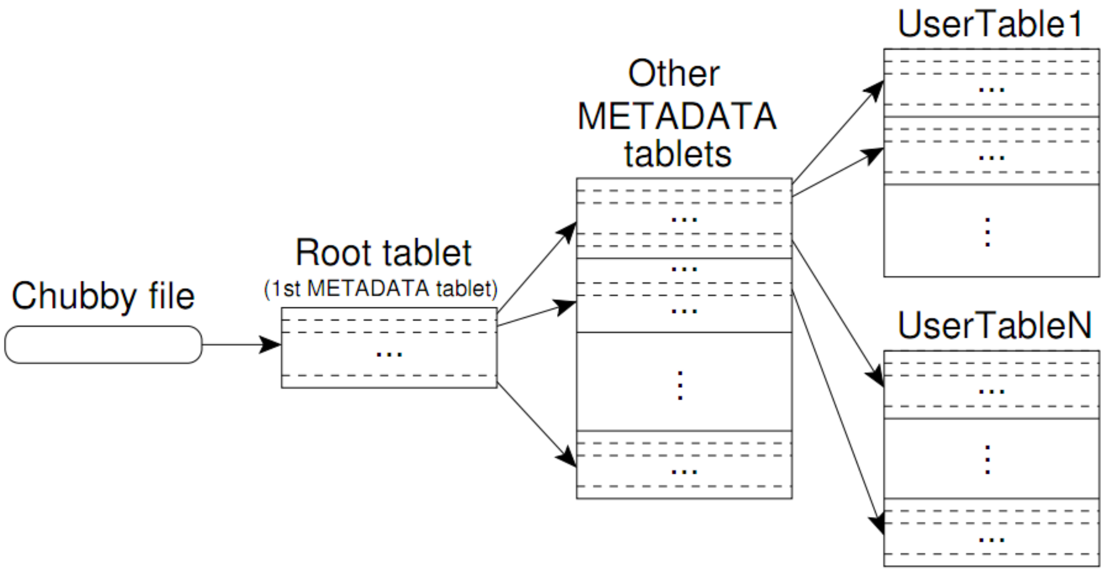
- Detect tablet server failures/resumption
- Master periodically asks each tablet server for the status of its lock
- Tablet server lost its lock or master cannot contact tablet server:
  - Master attempts to acquire exclusive lock on the server's file in the servers directory
  - If master acquires the lock then the tablets assigned to the tablet server are assigned to others
- If master loses its Chubby session then it kills itself
  - Election will be triggered

Implementation

- Each Tablets is assigned to one tablet server.
  - Tablet holds contiguous range of rows
    - Clients can often choose row keys to achieve locality
  - Aim for 100MB to 200MB of data per tablet
    - Automatically split tablets that have grown too large
- Tablet server is responsible for 10-1000 tablets
- Clients communicate directly with tablet servers

Tablet Locating

- Given a row, how do clients find the location of the tablet whose ro



METADATA: Key: table id + end row, Data: location

Tablet Locating

- A 3-level hierarchy analogous to that of a B+ tree to store tablet location information :
  - A file stored in chubby contains location of the root tablet
  - Root tablet contains location of *Metadata tablets*
    - The root tablet never splits
  - Each meta-data tablet contains the locations of a set of user tablets
- Client library aggressively caches tablet locations

Tablet Server

- When a tablet server starts, it creates and acquires exclusive lock on, a uniquely-named file in a specific Chubby directory
  - Call this servers directory
- A tablet server stops serving its tablets if it loses its exclusive lock
  - This may happen if there is a network connection failure that causes the tablet server to lose its Chubby session

Tablet Server

- A tablet server will attempt to reacquire an exclusive lock on its file as long as the file still exists
- If the file no longer exists then the tablet server will never be able to serve again
  - Kills itself
  - At some point it can restart; it goes to a pool of unassigned tablet servers

Tablet Server Failover

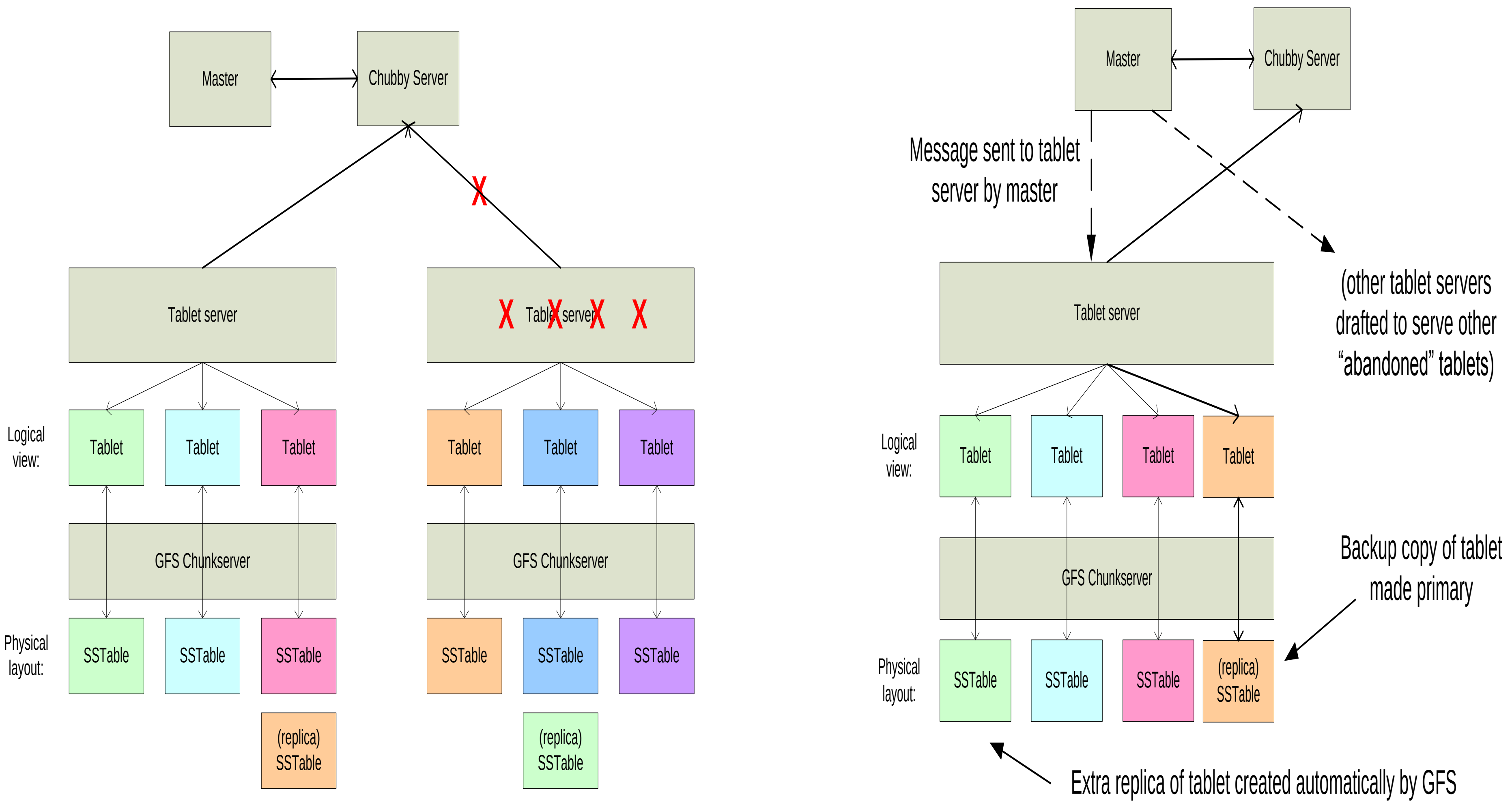
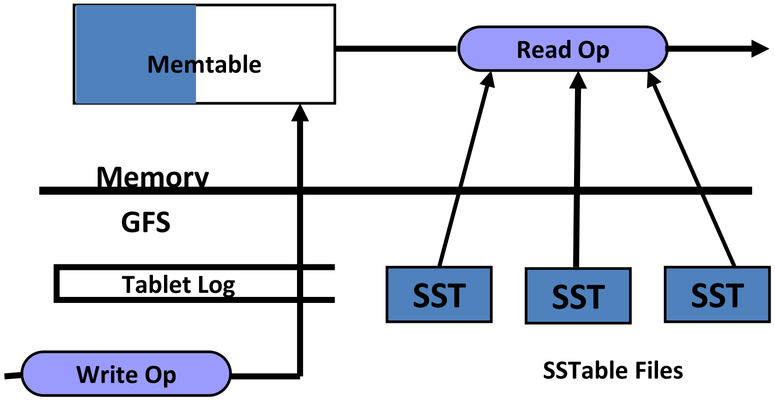


Image source: http://itsitrc.blogspot.ca/2013/02/bigtable-used-by-google.html

Tablet Server

- Commit log stores the updates that are made to the data
- Recent updates are stored in memtable (sorted buffer)
- Older updates are stored in SStable files



Compactions

- Minor compaction
  - At the threshold
  - Reduce memory usage
  - Reduce log traffic on restart
- Merging compaction
  - Periodically
  - Reduce number of SSTables
  - Good place to apply policy “keep only N versions”
- Major compaction
  - Results in only one SSTable
  - No deletion records, only live data

Refinements

- Locality groups
  - Clients can group multiple column families together into a *locality group.*
- Compression
  - Compression applied to each SSTable block separately
  - Uses *Bentley and McIlroy's* scheme and *fast compression* algorithm
- Caching for read performance
  - Scan Cache and Block Cache
- Bloom filters
  - Reduce the number of disk accesses

Outline

- Motivation
- Data Model
- APIs
- Building Blocks
- Implementation
- Refinement
- Evaluation

Evaluation

- N tablet servers (N=1,50,250,500)
- Round-trip time less than 1ms
- R row-keys ~ 1GB data per tablet server
- 1000-byte values one time
- 6 operations

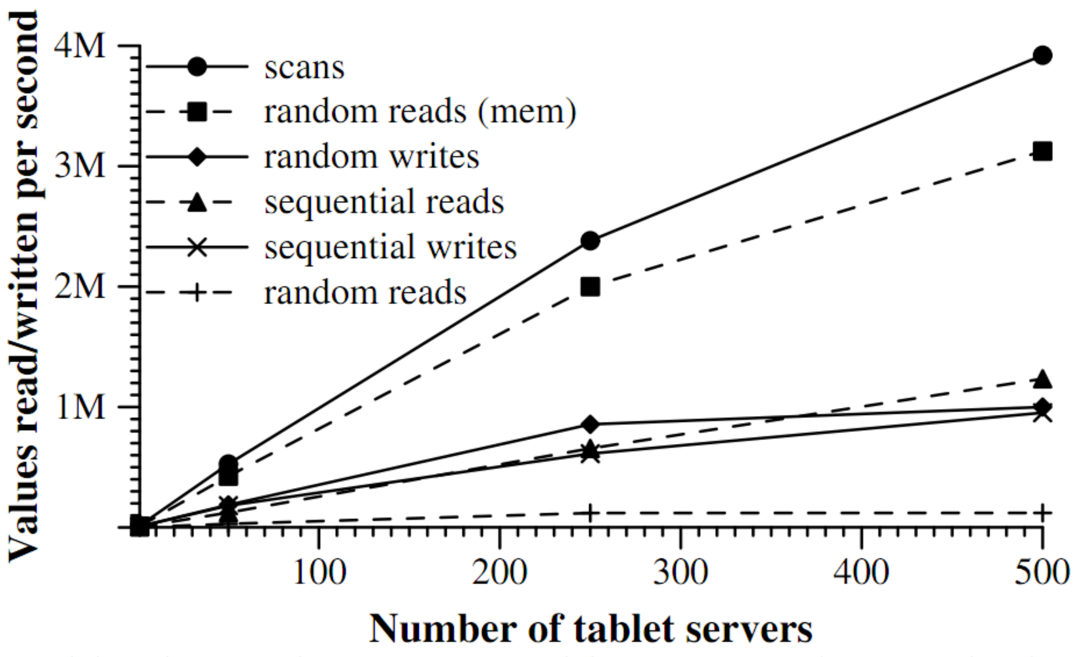
Evaluation

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

Performance - Scaling

- 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?



Comments

- The authors claim a very low failure rate, whereas they also mentioned the vulnerability in lessons due to many types of failures. Extend in this direction.
- The API does not support standard SQL query, it could be obstacle for applications, such as secondary index.

Thanks !