CS 695: Virtualization and Cloud Computing

Lecture 14: Semi-structured data stores: Bigtable

Mythili Vutukuru

IIT Bombay

Spring 2021

Google’s Bigtable

- Semi-structured data store: does not support full relational database model, simpler data format
  - Borrows ideas from parallel and in-memory databases
- Tables of rows, columns (strings). Each cell also has timestamp. Maps row key, column key and timestamp to a value (array of bytes)

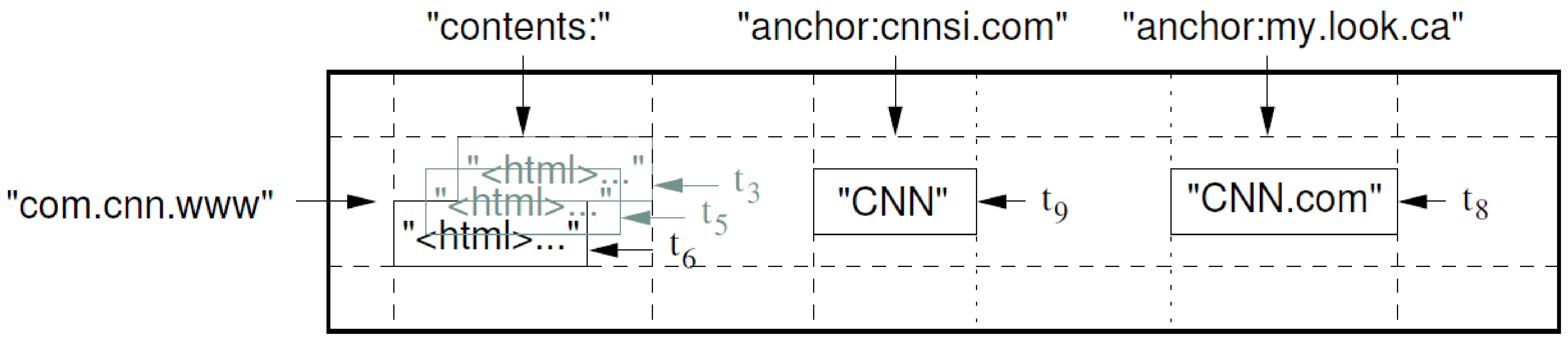
$$(\text{row:string}, \text{column:string}, \text{time:int64}) \rightarrow \text{string}$$

- Widely used by many systems at Google
  - Both batch processing and real time applications
  - Clients can control whether data on disk or in memory
- High availability, scalability, performance

Bigtable: A Distributed Storage System for Structured Data
Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows
Tushar Chandra, Andrew Fikes, Robert E. Gruber

Data model

- Rows: atomic unit of reading and writing data
  - Rows sorted alphabetically (users should pick row names suitably)
  - Group of adjacent rows is called a tablet (unit of distributing data to machines)
- Columns: grouped into small no. of families
  - All columns in a family store similar type of data, treated similarly
  - Family is granularity for specifying access control, locality (disk vs memory) etc.
- Timestamp of a cell acts as a version number and is provided by clients
  - Last N versions stored
- Example Webtable shown below



Bigtable API

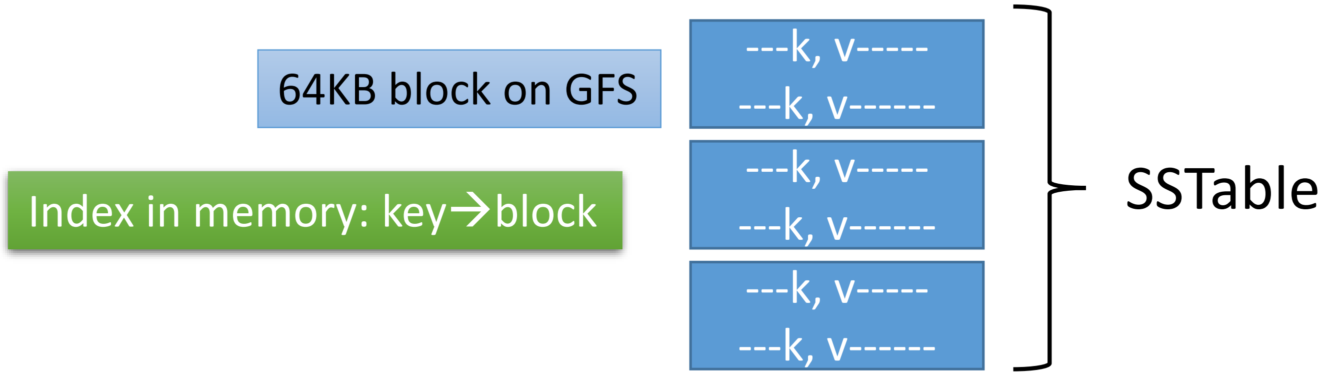
- Bigtable's API allows users to create tables and manipulate various cells
- Examples of reading and writing tables

```
// Open the table
Table *T = OpenOrDie("/bigtable/web/webtable");

// Write a new anchor and delete an old anchor
RowMutation r1(T, "com.cnn.www");
r1.Set("anchor:www.c-span.org", "CNN");
r1.Delete("anchor:www.abc.com");
Operation op;
Apply(&op, &r1);
```

```
Scanner scanner(T);
ScanStream *stream;
stream = scanner.FetchColumnFamily("anchor");
stream->SetReturnAllVersions();
scanner.Lookup("com.cnn.www");
for (; !stream->Done(); stream->Next()) {
  printf("%s %s %lld %s\n",
         scanner.RowName(),
         stream->ColumnName(),
         stream->MicroTimestamp(),
         stream->Value());
}
```

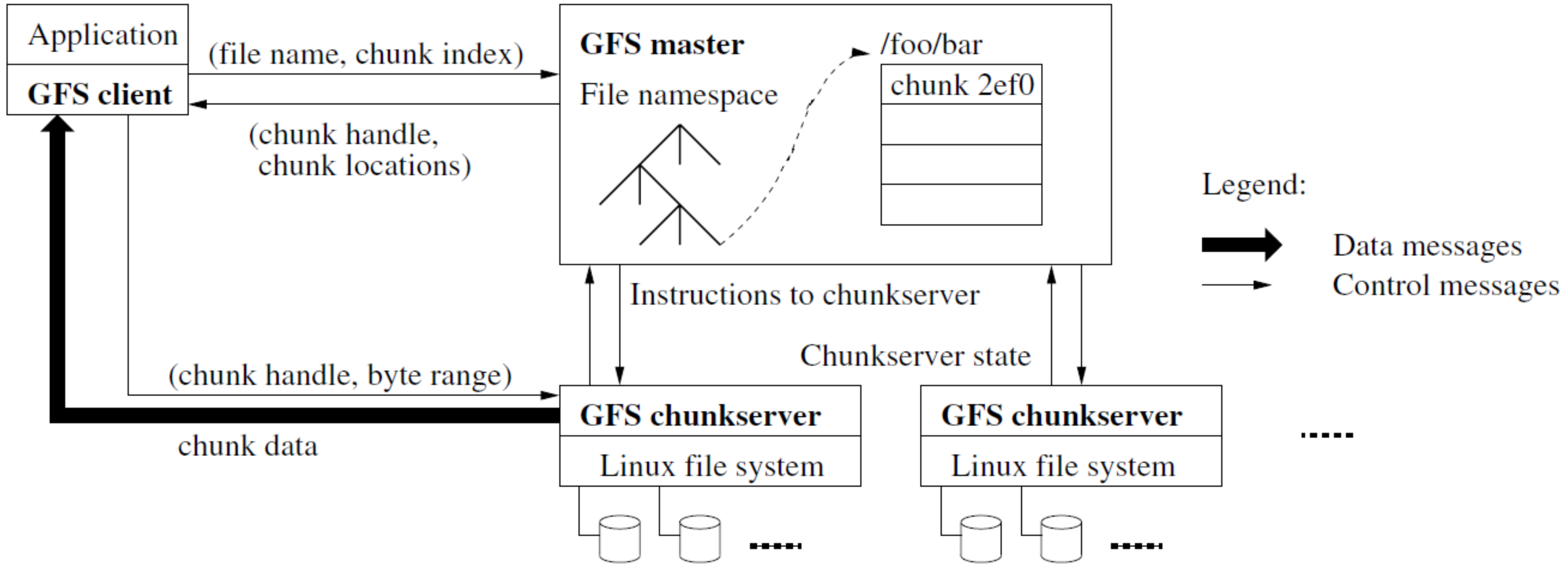
Building blocks (1)



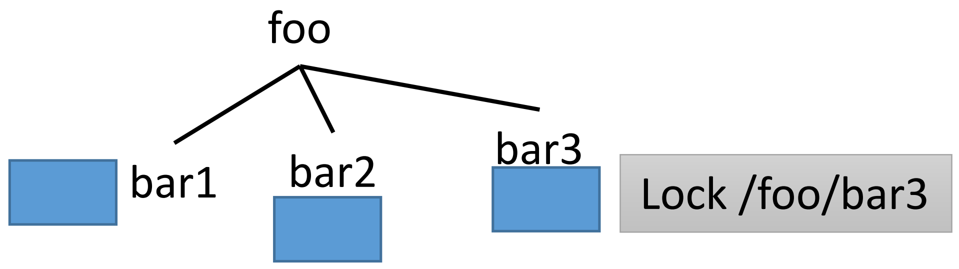
- Leverages two existing systems of Google: SSTable and Chubby
- SSTable: file format used to store Bigtable data internally
  - Immutable map of key-value pairs, stored on a sequence of blocks (64KB) on disk. Blocks stored on GFS (Google File System)
  - Can lookup a value using key, or iterate over all key-value pairs
  - Index of SSTable maps keys to disk blocks, loaded into memory when SSTable opened
  - Lookup only needs single disk access: lookup key in index to find block location, then access disk block
  - More efficient than storing files in other formats on regular filesystems

Building blocks (2)

- Google File System: distributed file system on commodity hardware
  - Designed to efficiently store a small number of large files (not POSIX API)
  - GFS cluster has one master and multiple chunk servers (Linux machines)
  - File divided into fixed size chunks, chunks replicated at multiple chunk servers
  - Chunks stored at chunk servers on local disk, identified by a unique handle
  - Master stores chunk handle → chunk server mapping



Building blocks (3)



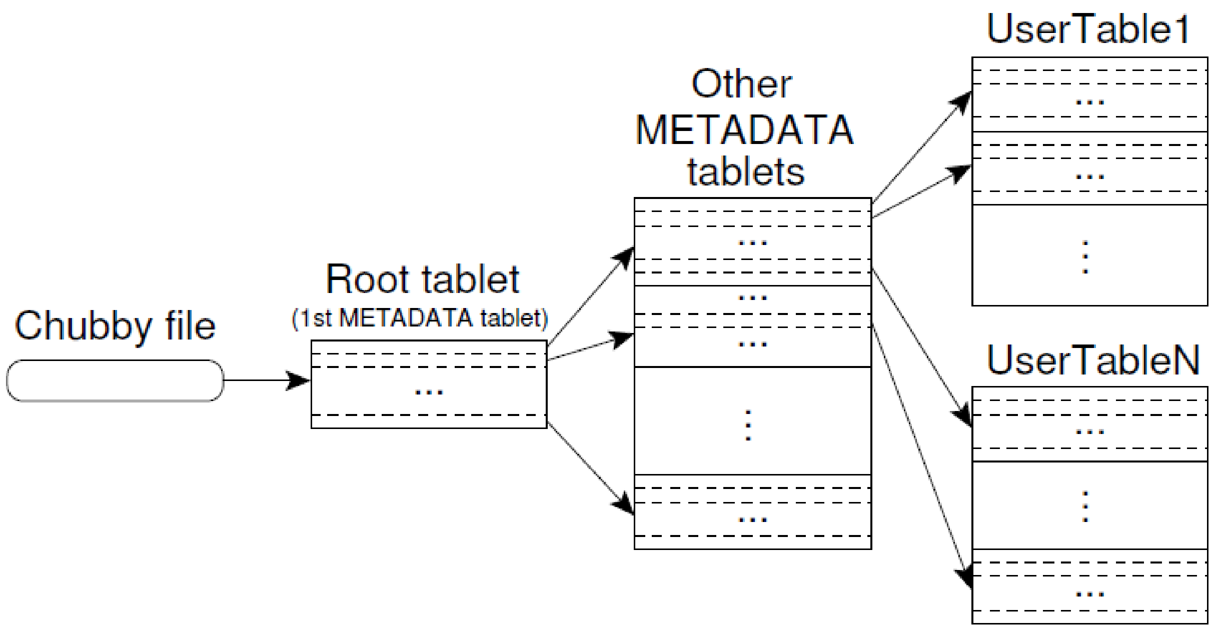
- Chubby: distributed lock service / distributed directory service
  - Exposes a namespace of directories and files
  - Users of chubby can acquire a lock on some directory/file and read/write its contents
- How is it useful?
  - Suppose an application is built to run at one node, needs to migrate to multiple nodes for fault tolerance. One way is to change app logic to run on multiple replicas, with some leader election/consensus algorithm among replicas – hard work!
  - Easier way: ask all replicas to lock a certain file using Chubby, whoever gets the lock is master. Master shares data with replicas via Chubby files
- Internally, Chubby implements Paxos across its 5 replicas, in order to consistently maintain replicated state of the lock namespace
  - Chubby runs Paxos, so that other apps do not have to reimplement the logic
- Clients maintain sessions with Chubby servers and send requests to acquire/release locks
  - If a session breaks (client fails), all locks held by a client are released.

Bigtable Implementation

- Components of the system:
  - Tablet servers: store few hundred tablets (group of rows of a bigtable)
  - Master: distributes tablets to tablet servers, load balancing, handles failures of tablet servers, creation/updation of table schema.
  - Clients: read/write tablets at tablet servers
- How does master track tablet servers?
  - Every tablet servers create its own unique file on Chubby
  - Master periodically looks at these files to find list of healthy tablet servers, assigns each tablet to one tablet server
  - If tablet server fails, Chubby connection lost, file is deleted, master reassigns its tablets to other servers.
  - If master fails, it scans list of tablet servers, queries them for tablets, and reconstructs tablet assignment

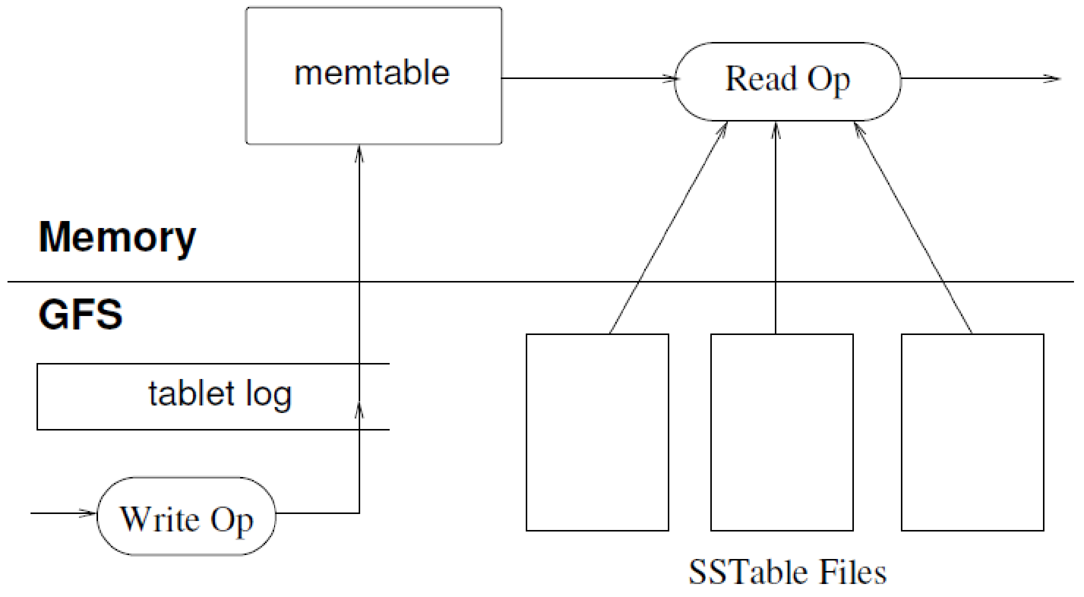
Locating tablets

- Tablet locations stored in metadata tablets
- Locations of all metadata tablets stored in a root tablet
- Root tablet published in Chubby by master
- Clients cache locations of tablets: no need to query master
  - No need to traverse the metadata tables
- Assume 1KB location information, 128 MB tablets
  - Each tablet stores 128K = 2^17 locations
  - Maximum filesize = 2^34 tablets = 2^61 bytes



Inside a tablet server (1)

- Persistent data is stored on GFS
    - GFS handles replication, so tablet can be just stored at one tablet server
- Each tablet server has multiple immutable SSTables and commit logs in GFS and a memtable in memory
    - List of all SSTables and commit logs of tablet server stored in metadata in Chubby
- Write operation:
    - Changes first written to a commit log on GFS
    - Recently committed writes are in memtable in memory
    - Old data is not deleted from SSTable, only deletion record written
- Read operation reads from merged view of memtable and SSTables
    - Latest value of a row is chosen from merged view
    - Easy to merge as all tables are sorted



Inside a tablet server (2)

- Why immutable SSTable? Simplifies design considerably
  - Can be accessed concurrently without locks during reads and writes
  - Only memtable needs concurrency control
- Compactions
  - Once enough data accumulates, memtable compacted into SSTable and stored in GFS
  - Periodically, multiple SSTables merged to create fewer SSTables (handling deletions)
- Separate locality groups created for each column family
  - Column families stored together in SSTables
  - SSTable of a column group can be specified to be on disk or in-memory
  - Compression format can be specified for a column group
- Caching at tablet server to improve performance
  - Recently read key-value pairs from SSTable are cached
  - Recently read blocks of SSTable are cached

Summary

- Key ideas:
  - Modified data format (semi structured data) for ease of optimization
    - Simpler than RDBMS, more complex than key-value stores
  - Decentralized master-worker design for scalability: tablet servers directly serve data to clients, master only maintains metadata. System performance scales by adding more tablet servers
  - Reuse existing functionality: SSTable in GFS for distributed storage, Chubby for coordination/consensus
  - Store data as immutable, fixed size SSTables (not general purpose files) for efficiency