BigTable: A Distributed Storage System for Structured Data

Course Information
DCS HPC - CSE449
Summer 2022

Group Information
Tanvir Rahman - 19101268
Group - 5

Instead of fighting other woman for a seat at the table,

DEMAND A BIGGER TABLE

What is BigTable?

- A NoSQL database services for large analytical and operational workloads
- Designed to scale large size data
- Fault-tolerant, flexible and high performance solution
- Self-managed database system
- Used in Web Indexing, Personalized Search, Google Map-Earth-Analytics

Related Works

- Apache Cassandra
- Apache HBase
- Apache Ignite
- Amazon SimpleDB
- Couchbase Server

MOTIVATION

Requirement

- Need of storage expansion
- Low level storage optimization

Performance

- Wide scalability and applicability
- High performance and availability

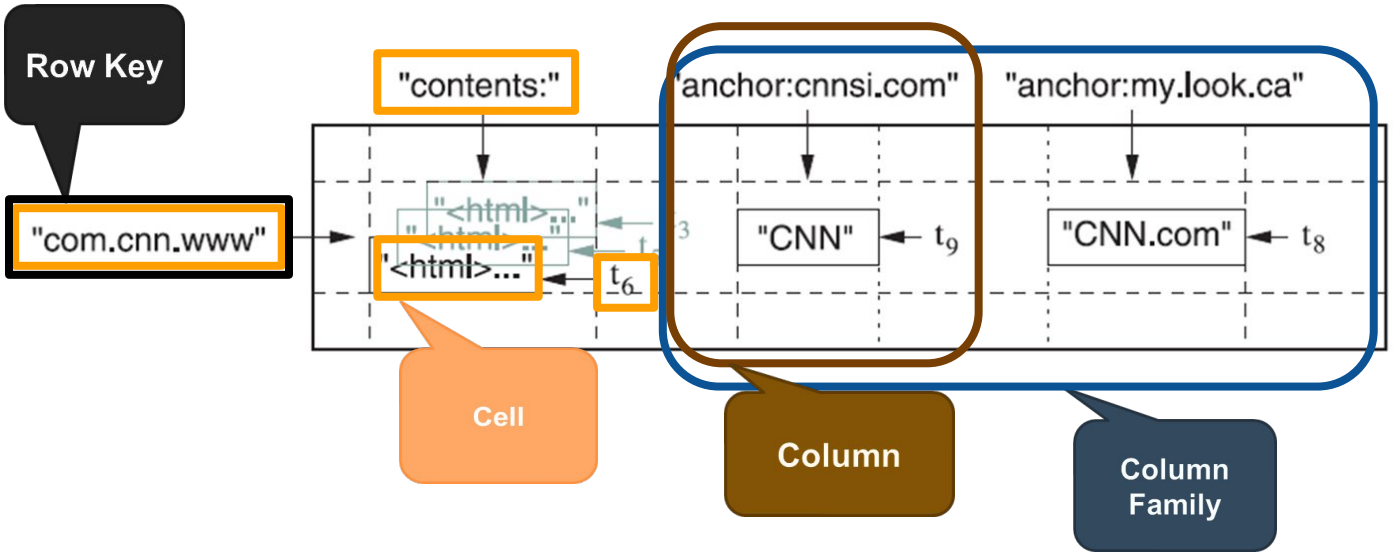
Cost

- Cost of commercial databases can be reduced
- Internal system building cost can be decreased

DATA MODEL

- ➔ Row and Column keys known as Tablets and Column Families
- ➔ Uses load balancing and units of distribution for Tablets
- ➔ Timestamps are 64-bit integers
- ➔ Does not support relational data model completely
- ➔ Allows for dynamic control over data layout and format
- ➔ Allows clients to manage locality of their data through a particular schema

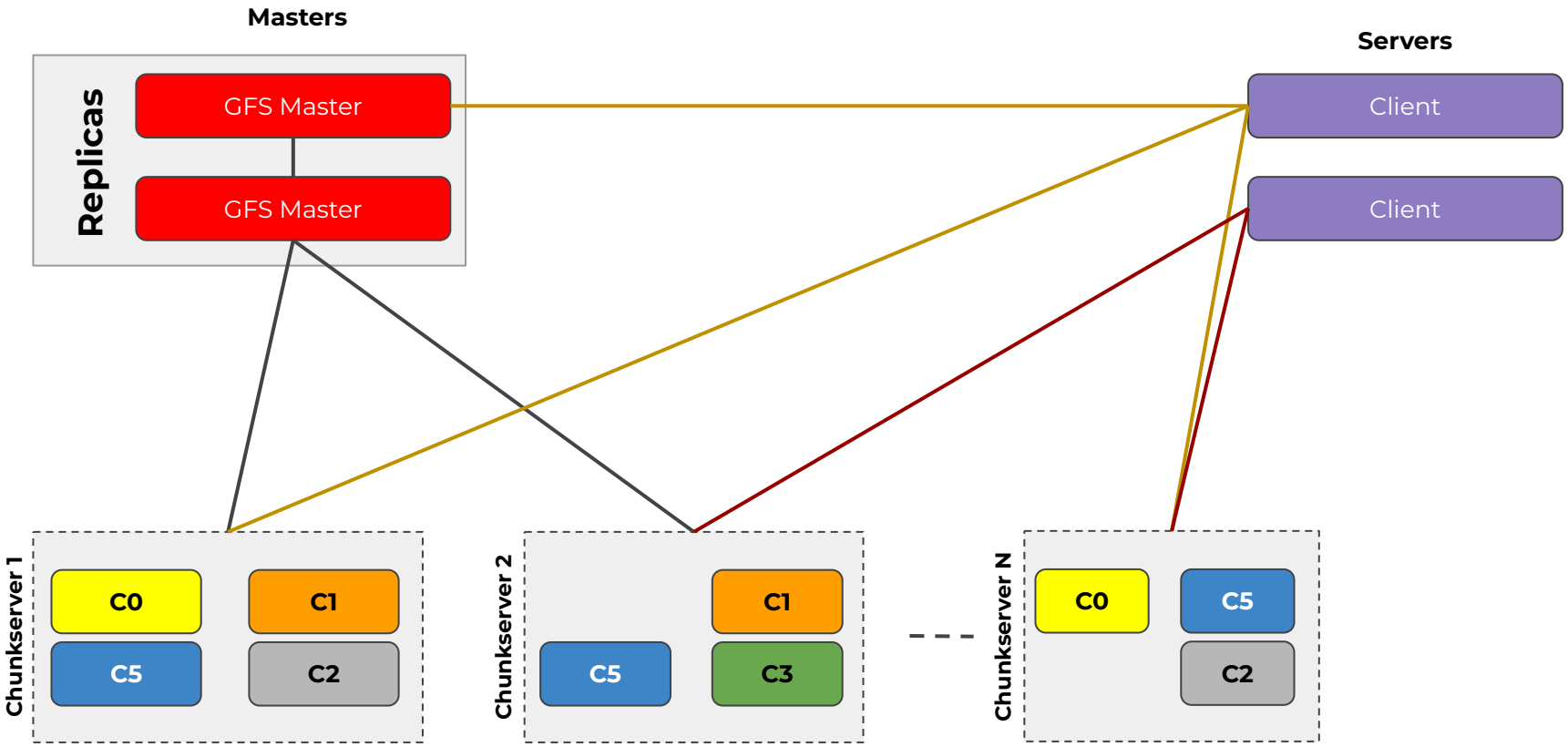
DATA MODEL (cont.)



BUILDING BLOCKS

Name	Services	Processes
Google File System (GFS)	Raw Storage	Conserves persistent state
Lock Service	Manages distributed locks	Does master selection and bootstrapping for locations
MapReduce	Easy processing for large-scale data	Uses frequently to read/write BigTable data
Scheduler	Job scheduling for machines	Schedules BigTable serving jobs

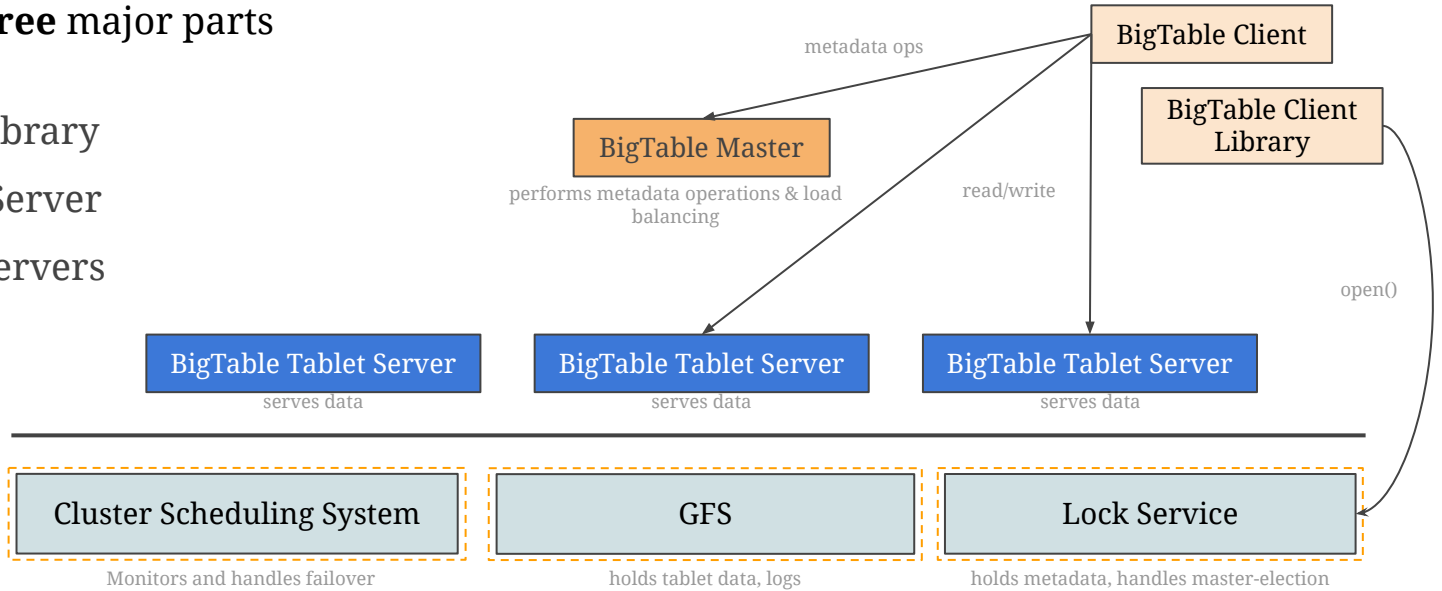
GOOGLE FILE SYSTEM



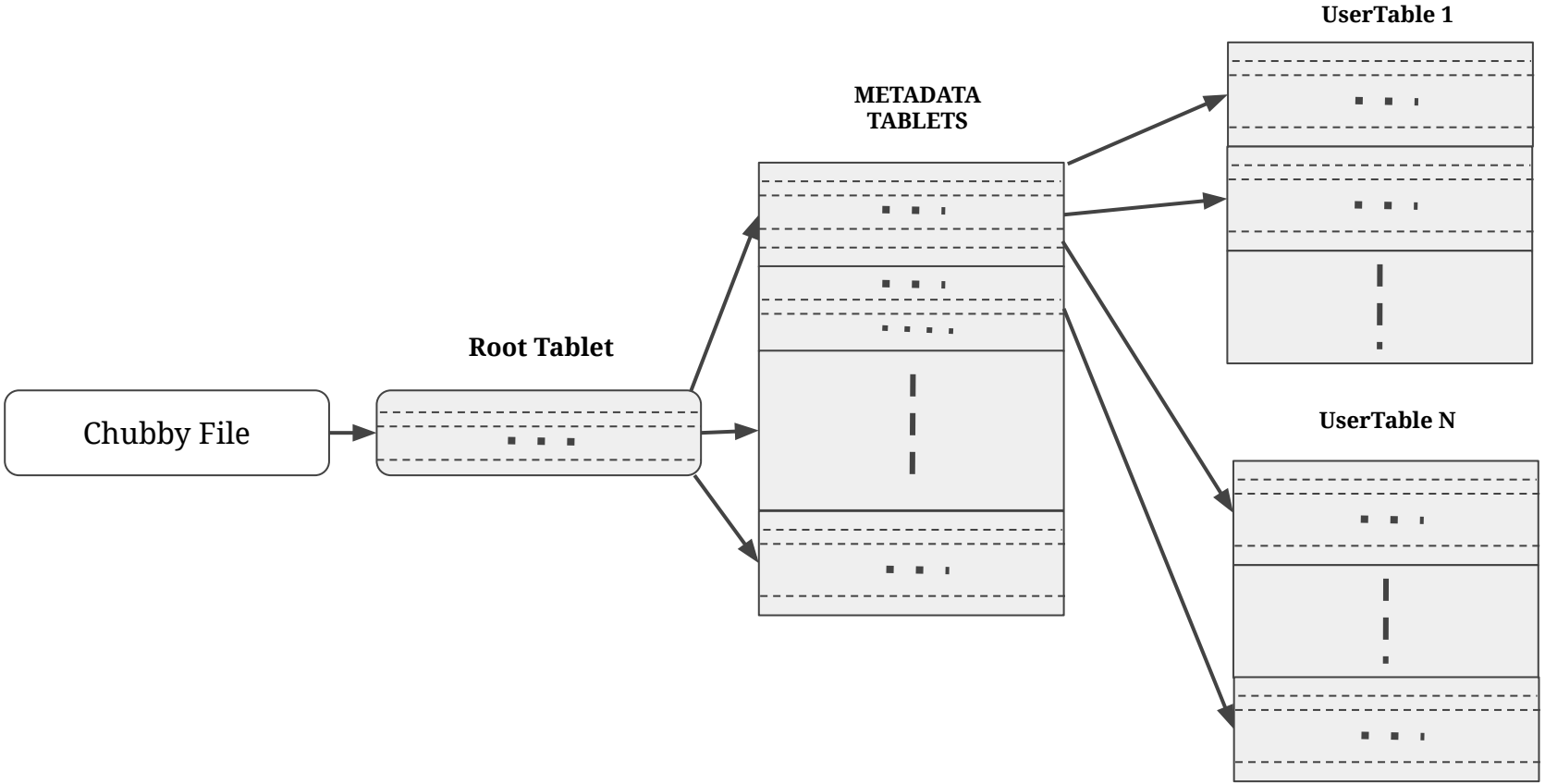
IMPLEMENTATION

Consists of **three** major parts

1. Client Library
2. Master Server
3. Tablet Servers



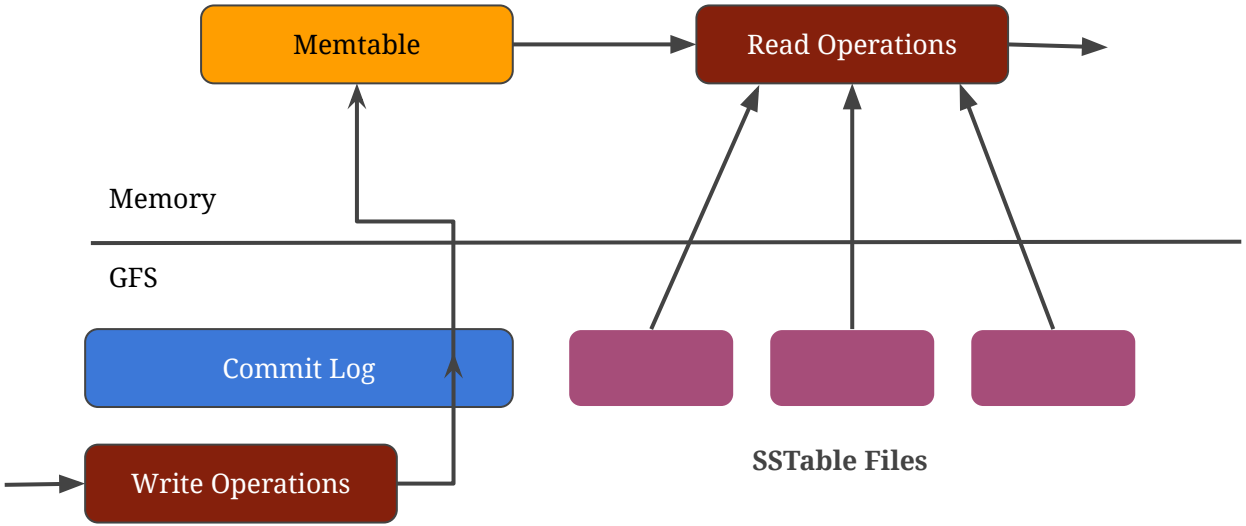
CLIENT FINDS TABLET



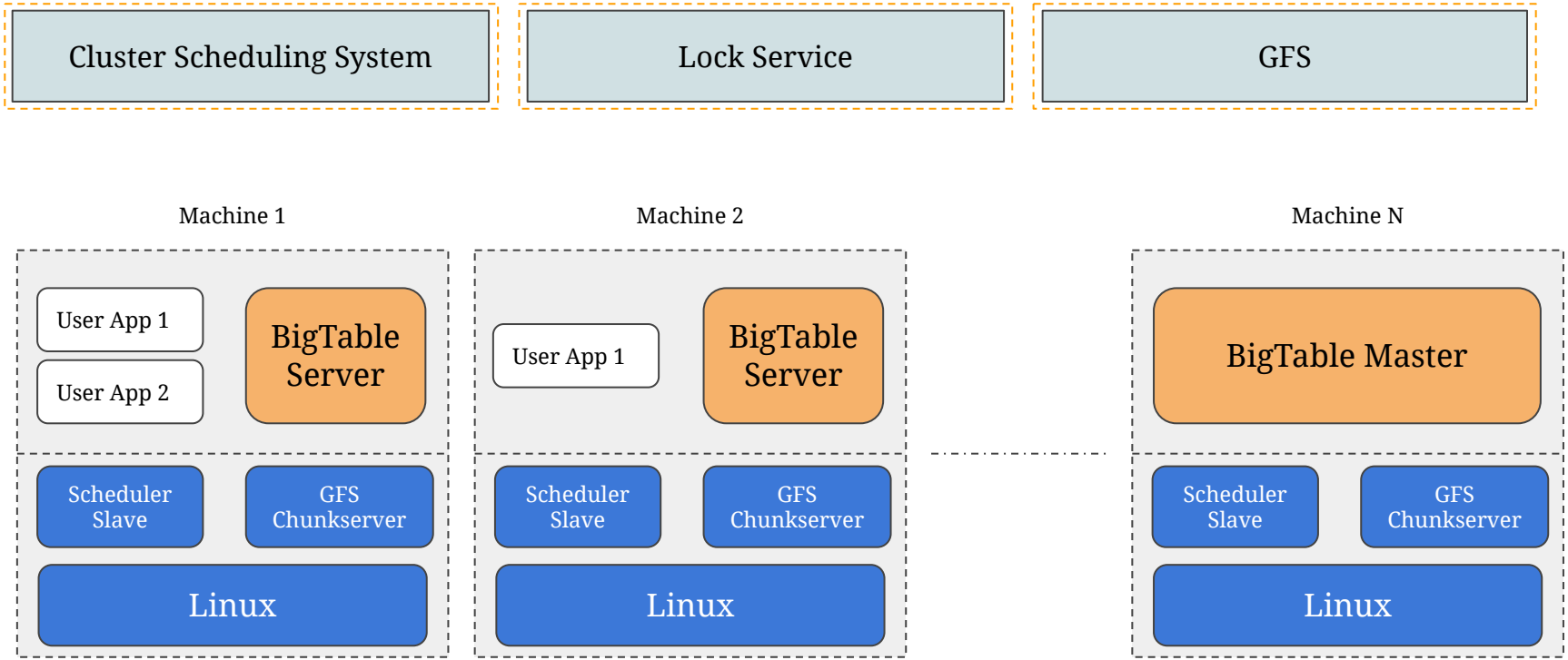
TABLET SERVER

- ➔ Manages a set of tablets
- ➔ Handles read and write operations
- ➔ Splits tablets if grown too large
- ➔ Aims for ~100MB to 200MB of data per tablet
- ➔ Responsible for ~100 tablets
- ➔ Fine-grainer load balancing

TABLET SERVING



GOOGLE CLUSTER



CLIENT API

- ➔ Function for tables, columns families to create or delete
- ➔ Function for changing cluster, table, column family metadata
- ➔ Supports for single row transactions
- ➔ Allows cells to be used as integer counters
- ➔ Clients can execute scripts in server side

UPDATE ROW

```
// 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);
```

Mutating the row

Storing new item under column key

Deleting an item under column key

Atomic the mutation

ITERATE TABLE

```
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());
}
```

Access column family

Specify return version

Specify row key

Iterate over rows

COMPACTIONS

Tablet state is represented as a collection of SSTable files that have been compressed and are immutable.

Minor Compaction

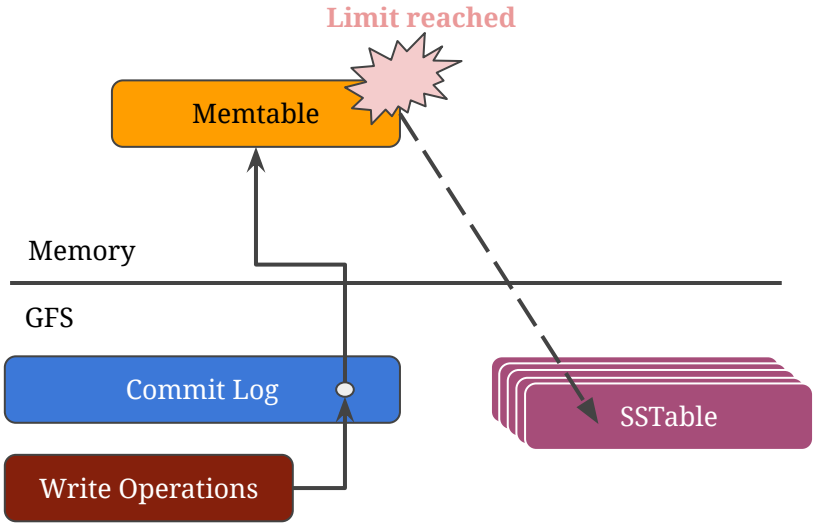
Memtables freeze, new ones get created, and writes content to SSTables that are kept in GFS whenever the in-memory state exceeds limit.

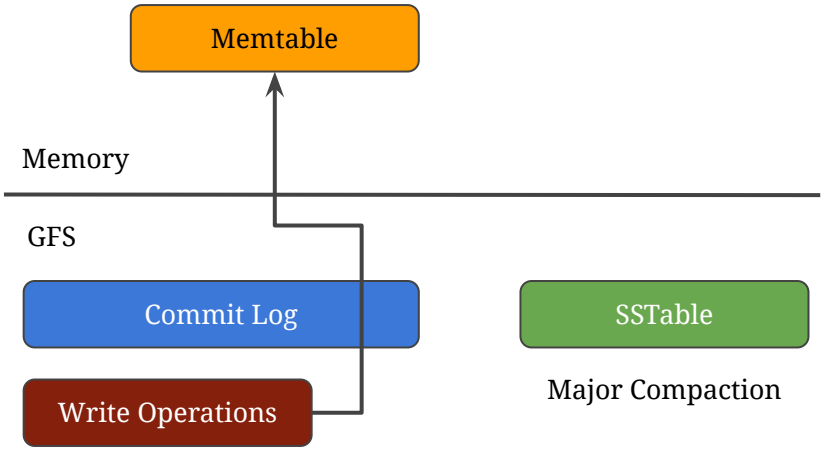
Major Compaction

Occasionally consolidates associated SSTables into a new version of SSTable on GFS. And, from the deletions process storage gets recovered.

*GFS = Google File System

COMPACTIONS (cont.)





REFINEMENTS

- ➔ Group multiple column families together
- ➔ Compress locality groups using Bentley and McIlroy scheme
- ➔ Bloom filters allows to ask existence of data into a specific row/column pair
- ➔ Caching for read performance

Scan Cache

A higher level cache that stores key-value pairs which are returned by the SSTable interface to the tablet server code

Block Cache

A lower level cache that stores SSTable blocks which are read from GFS.

SINGLE TABLET SERVER

Number of 1000-byte values are read/write per second.

Experiment	Table Server Count			
	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

FUTURE PLANS

- ➔ Multi row transaction support
- ➔ Performance enhancing for large cells
- ➔ Resource fairness, performance, isolation, prioritization across different clients
- ➔ More expressive data manipulation
- ➔ Support advanced indexing like secondary indices

CONCLUSIONS

- ➔ Provides high performance storage system
- ➔ Data model applicable to broad range of clients
- ➔ Advantages of building own storage system
- ➔ Able to handle such a wide array of requirements and workloads

REFERENCES

1. Chang F., Dean J., Ghemawat S., Hsieh W.C., Deborah A. Wallach, Burrows M., Chandra T., Fikes A., & Gruber R.E. (2006). Bigtable: A Distributed Storage System for Structured Data. In 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (pp. 205–218).
2. Joen K. (2012). Bigtable: A Distributed Storage System for Structured Data. Web-scale Data Management. https://slideplayer.com/slide/2749665.
3. Pirzadeh P., Ayyalasomayajula V. (2017). Bigtable: A Distributed Storage System for Structured Data. https://slidetodoc.com/bigtable-a-distributed-storage-system-for-structured-data-3.
4. Habibi B., Habibi S., Forouzandeh S. (2016). Bigtable: A Distributed Storage System for Structured Data. https://slideplayer.com/slide/7934078.
5. Fawad A. (2015). BigTable and Google File System (GFS). https://slideplayer.com/slide/6230378.

THANK YOU