LevelDB (Bigtable)

Authors: Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

Presenter: Xiling Li

Northwestern | McCORMICK SCHOOL OF ENGINEERING

Motivation

- Large amount of data needed to be stored
- Data may be used to many different purposes
- Data may be stored in different machines

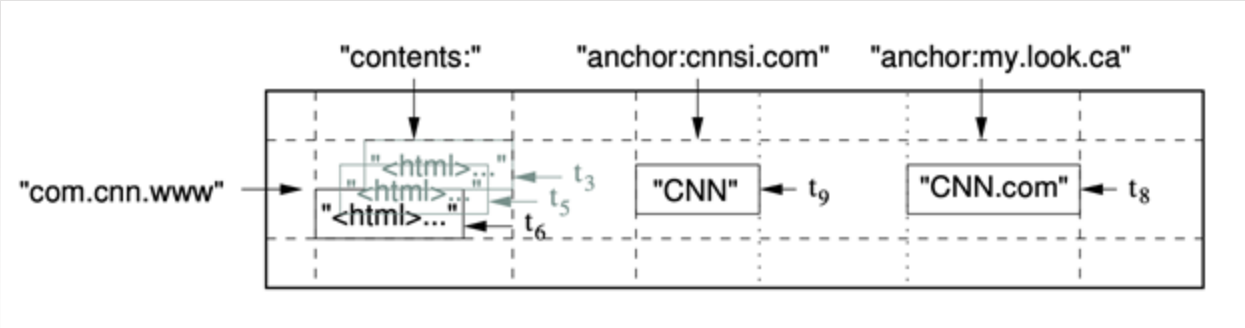
What is LevelDB?

- Distributed storage system for structured data
- Key: (Row, Col, timestamp) ; Value: Arbitrary string

Why LevelDB?

- Flexibility
  - For applications w/ various latency demands and data sizes
- Scalability
  - Petabyte-level data/thousand-level machines
- High performance/High availability
  - Thanks to Google building blocks

Data Model



- Webtable Example:
  - Row: reversed URL
  - Col: Contents, Anchor
  - Timestamp: when is fetched

APIs

```
// 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);
```

Figure 2: Writing to Bigtable.

```
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());
}
```

Figure 3: Reading from Bigtable.

- C++ interfaces
- Write: RowMutation(Table, RowKey)
- Read: Scanner (iterate cols of a row)
- Single Row Transaction
  - Atomic read-modify-write fashion

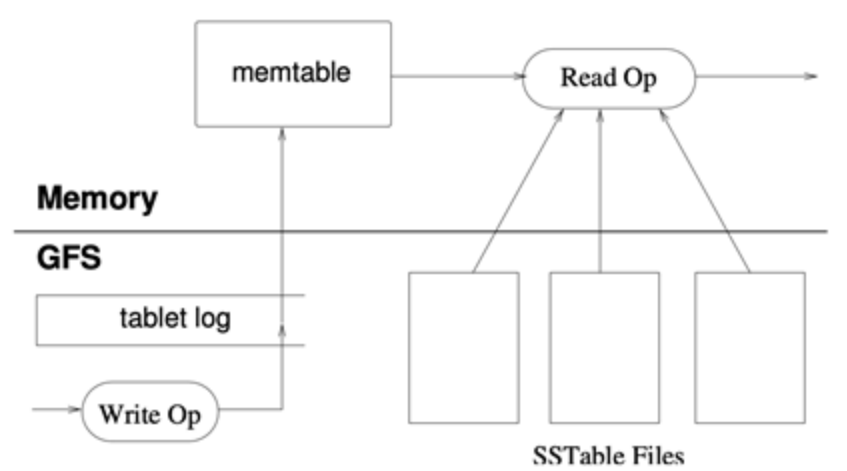
Building Blocks

- Google File System (GFS)
  - Distributed log/data files storage
  - Managing scheduling, recourses, failure and machine status.
- SSTable file format
  - Persistent, ordered immutable map (key -> value)
  - Quick lookup/key range scan w/ block index loaded into memory
- Chubby
  - Highly available and persistent distributed lock service
  - One active master server at any time

Architecture

- Tablet
  - Contains data w/ row range
- Tablet server
  - Handle read/write requests to a set of tablets
- Master server
  - Manage tablets (assignment, load balancing, garbage collection)
- Clients
  - Directly communicate with tablet servers

Tablet in detail



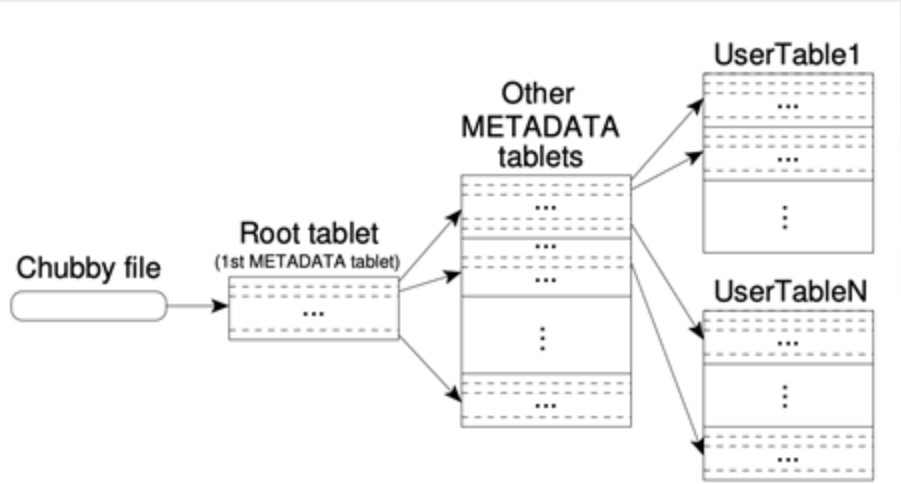
Analogous to LSM-tree structure

- Memtable (Sorted)
  - New changes are stored in RAM
- SSTable (Sorted)
  - Persistently store old data
- METADATA table
  - Lists of SSTables and redo points
  - Recovery for memtable

Read in the merged fashion

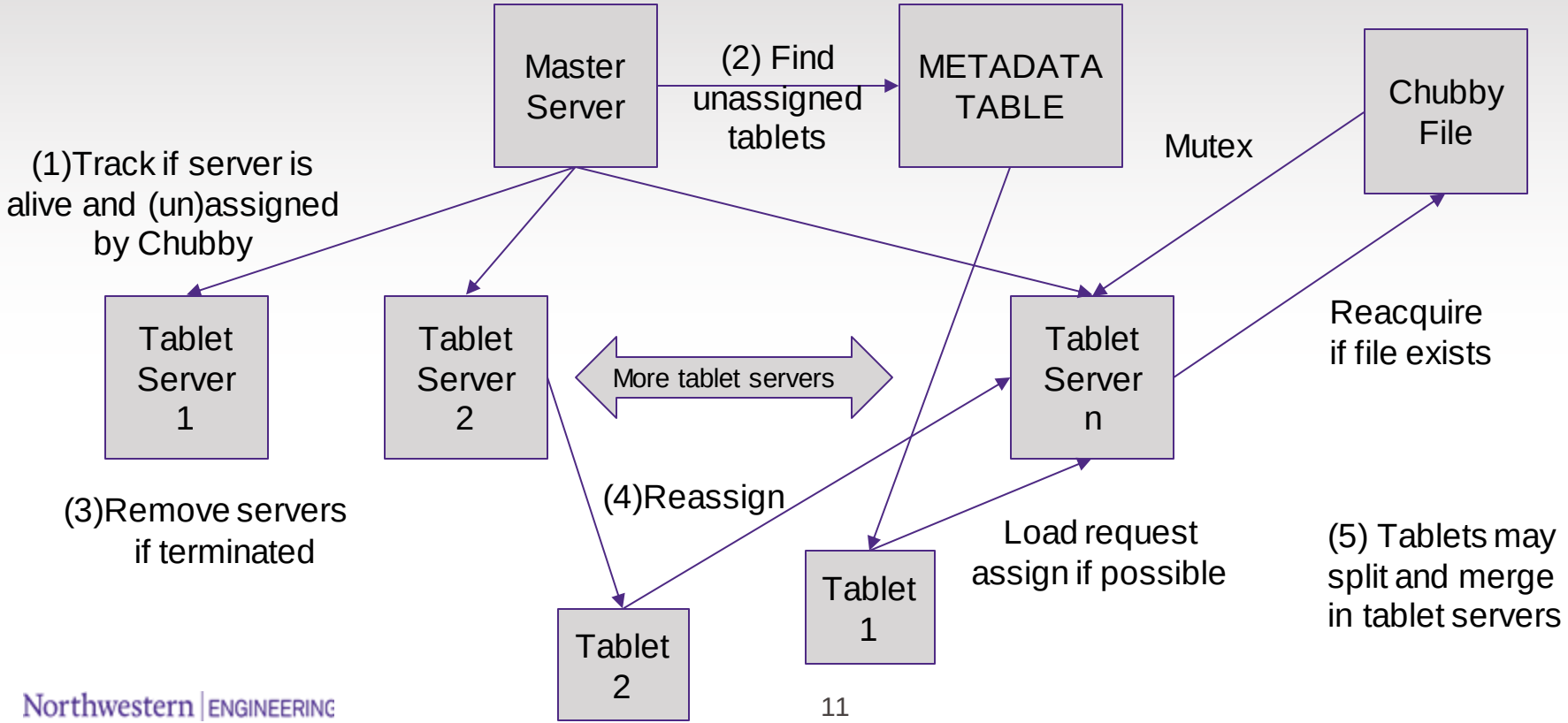
Talk about more write in compaction

Tablet Location



- Three-level hierarchy
  - Root tablet in Chubby
    - Loc of all tablets (METADATA tab)
  - METADATA tablets
    - Loc of user tablets
  - User tablet
    - Encode table identifier and row key

How assignment works?



Compaction

- Minor compaction (memtable size > a threshold):
  - Create a new memtable
  - Convert old one to SSTable and write to GFS
- Merging compaction
  - Periodically merge in the background to new SSTable
  - Discard old memtable and fetched SSTables after compaction
- Major compaction
  - Rewrite all SSTables in merging fashion into one new SSTable
  - Get more space for deleted data if storage is sensitive

Improvements

- Locality group
  - Group (in-memory) multiple col families to make read faster
- Compression
  - Option to compress SSTables for a locality group
- Caching for read performance
  - Scan cache for k/v pairs from SSTable to tablet server
  - Block cache for SSTable blocks read from GFS
- Bloom filters
  - Reduce # of access by creating BF for SSTables in a locality group

Improvements - Cont'd

- Commit-log implementation
  - Maintain a single physical log file
- Faster tablet recovery
  - Additional minor compaction for moving around tablets
- SSTable's Immutability
  - Efficient concurrency control

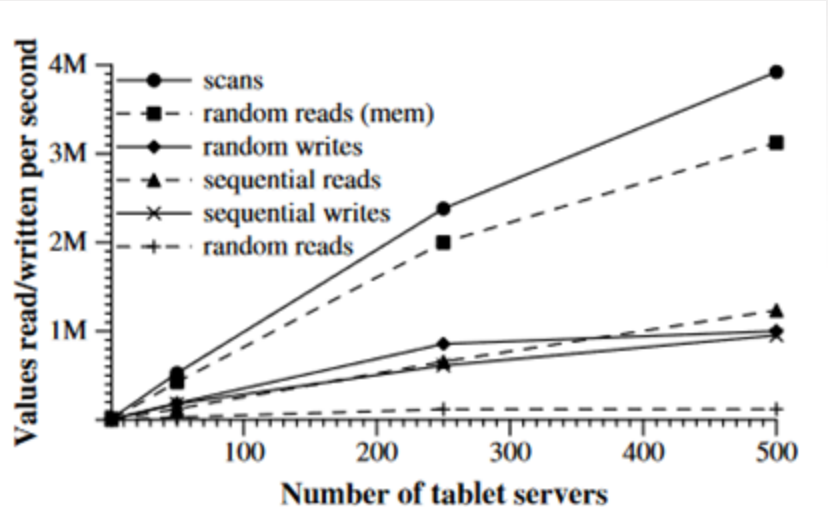
Experimental results

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

A Bigtable cluster with N tablet servers

- Random read from disk is slowest among all exps
- Read w/ in-mem locality group is one order of magnitude faster than read from disk.
- Sequential read is also more efficient than random read, while write does not vary significantly
- Scan is even faster than random read in memory
- Fun Face: write is faster than read in either sequential or random fashion (analogous to LSM-tree - fast write and slow read)

Scalability



- As # of tablet servers increases, the trend is more significant
- Random read from mem has CPU bottleneck
- Drop from 1 to 50 caused by load imbalance of configs

Applications

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

- Demonstrate LevelDB works for applications w/ different demands!

Conclusion

- LevelDB is a distributed storage system for structured data with row, col, time as key and arbitrary string as value to support flexity for various applications, scalability to different size/# of machines with underlying high performance/availability building blocks

Thanks!

Northwestern | McCORMICK SCHOOL OF ENGINEERING