Data-Intensive Distributed Computing
CS 431/631 451/651 (Winter 2021)

Part 8a: Mutable State (1/2)

Ali Abedi
Mutable state

From sequential reads and append only writes

To random reads and writes
Why not RDBMS?

- Does not scale out → expensive
- Does not support semi-structured data
NoSQL
(Not only SQL)

1. Horizontal scaling
2. Replicate/distribute data over many servers
3. Simple call interface
4. Weaker concurrency model than ACID
5. Flexible schemas

But, don’t blindly follow the hype... Often, MySQL is what you really need!
SQL vs. NoSQL

**SQL**
- Vertically scalable
- Fixed schema
- Mature
- ~ TB
- Strong consistency

**NoSQL**
- Horizontally scalable
- Dynamic schema
- Emerging
- ~ PB
- Eventual consistency
(Major) Types of NoSQL databases

Key Value
- Example: Riak, Tokyo Cabinet, Redis server, Memcached, Riak

Document-Based
- Example: MongoDB, CouchDB, OrientDB, RavenDB

Column-Based
- Example: BigTable, Cassandra, Hbase, HyperTable

Graph-Based
- Example: Neo4J, InfoGrid, Infinite Graph, Flock DB

Source: guru99.com/nosql-tutorial.html
3 DATABASE ADMINs
WALKED INTO
A NOSQL BAR...

A LITTLE WHILE LATER
THEY WALKED OUT BECAUSE
THEY COULDN'T FIND A TABLE
# KEY-VALUE STORE

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>AAA,BBB,CCC</td>
</tr>
<tr>
<td>K2</td>
<td>AAA,BBB</td>
</tr>
<tr>
<td>K3</td>
<td>AAA,DDD</td>
</tr>
<tr>
<td>K4</td>
<td>AAA,2,01/01/2015</td>
</tr>
<tr>
<td>K5</td>
<td>3,ZZZ,5623</td>
</tr>
</tbody>
</table>
Three Core Ideas

Partitioning (sharding)
To increase scalability and to decrease latency

Consistency?
Replication
To increase robustness (availability) and to increase throughput

Caching
To reduce latency
Chord distributed protocol

Stoica et al. (2001). Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. SIGCOMM. 
And other resources ...
Routing: Which machine holds the key?

Each machine holds pointers to predecessor and successor

Send request to any node, gets routed to correct one in $O(n)$ hops

Can we do better?
Routing: Which machine holds the key?

Each machine holds pointers to predecessor and successor

+ “finger table”
(+2, +4, +8, ...)

Send request to any node, gets routed to correct one in $O(\log n)$ hops
Routing: Which machine holds the key?

\[ h = 2^n - 1 \]
New machine joins: What happens?

How do we rebuild the predecessor, successor, finger tables?

\( h = 2^n - 1 \)
Machine fails: What happens?

Solution: Replication

\[ h = 2^n - 1 \]
Bigtable Applications

Gmail
Google’s web crawl
Google Earth
Google Analytics
Data source and data sink for MapReduce

HBase is the open-source implementation...
Data Model

A table in Bigtable is a sparse, distributed, persistent multidimensional sorted map

Map indexed by a row key, column key, and a timestamp
(row:string, column:string, time:int64) → uninterpreted byte array

Supports lookups, inserts, deletes
Single row transactions only
Rows and Columns

Rows maintained in sorted lexicographic order
Applications can exploit this property for efficient row scans
Row ranges dynamically partitioned into tablets

Columns grouped into column families
  Column key = family:qualifier
  Column families provide locality hints
  Unbounded number of columns

At the end of the day, it’s all key-value pairs!
<table>
<thead>
<tr>
<th>row, column family, column qualifier, timestamp</th>
<th>value</th>
</tr>
</thead>
</table>

**Key-Values**
Okay, so how do we build it?

<table>
<thead>
<tr>
<th>In Memory</th>
<th>On Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutability Easy</td>
<td>Mutability Hard</td>
</tr>
<tr>
<td>Small</td>
<td>Big</td>
</tr>
</tbody>
</table>
Log Structured Merge Trees

What happens when we run out of memory?
Log Structured Merge Trees

Writes → MemStore → Reads

Memory

Disk

Flush to disk

Store

Immutable, indexed, persistent, key-value pairs

What happens to the read path?
Log Structured Merge Trees

Writes → MemStore → Merge → Reads

Memory → Disk

Flush to disk

 Immutable, indexed, persistent, key-value pairs

What happens as more writes happen?
Log Structured Merge Trees

- **Writes** to **MemStore**
- **Memory** and **Disk**
- **Flush to disk**
- **Merge**
- **Reads**

Immutable, indexed, persistent, key-value pairs

**What happens to the read path?**
Log Structured Merge Trees

Writes  →  MemStore  →  Merge  →  Reads

Memory
Disk

Flush to disk

Store  Store  Store  Store

Immutable, indexed, persistent, key-value pairs

What’s the next issue?
Log Structured Merge Trees

- **Writes** → **MemStore** → **Merge** → **Reads**

  - **Memory**
  - **Disk**

  **Flush to disk**

  **Compaction!**

  **Store** → **Store** → **Store**

  Immutable, indexed, persistent, key-value pairs
Log Structured Merge Trees

- **MemStore**
- **Store**

- **Memory**
- **Disk**

- **Flush to disk**

- **Merge**

**Writes** → MemStore → Merge → Reads

Immutable, indexed, persistent, key-value pairs
Log Structured Merge Trees

Writes to MemStore are then remembered for persistence in the Write-Ahead Log (WAL) and eventually merged into the Store. The Store provides immutable, indexed, persistent, key-value pairs.

One final component...

Flush to disk
Log Structured Merge Trees
The complete picture...

Writes → MemStore → Merge → Reads

Memory
Disk

Flush to disk

Logging for persistence

WAL
Store
Store
Store

Immutable, indexed, persistent, key-value pairs

Compaction!
Log Structured Merge Trees
The complete picture...

Okay, now how do we build a distributed version?
HBase: Bigtable building blocks

HDFS
GFS
HFile
SSTable
Tablet
Region
Regions Server
Tablet Server
Chubby
Zookeeper
SSTable

Persistent, ordered immutable map from keys to values

Stored in GFS: replication “for free”

Supported operations:
- Look up value associated with key
- Iterate key/value pairs within a key range
Region Tablet

Dynamically partitioned range of rows
Comprised of multiple SSTables

SSTable  SSTable  SSTable  SSTable
Region Server

Tablet Server

MemStore

Writes

Reads

Memory

Disk

Flush to disk

Logging for persistence

WAL

SSTable

SSTable

SSTable

Immutable, indexed, persistent, key-value pairs

Compaction!
Table

Comprised of multiple tablets
SSTables can be shared between tablets

Tablet
aardvark - base

Tablet
basic - database

SSTable

SSTable

SSTable

SSTable

SSTable

SSTable
Each tablet is assigned to one tablet server at a time
Exclusively handles read and write requests to that tablet

What happens when a tablet grow too big?
What happens when a tablet server fails?

We need a lock service!
HBase Bigtable building blocks

HDFS
GFS
HFile
SSTable
Tablet
Region
Regions Server
Tablet Server
Chubby
Zookeeper
Architecture

Client library

Bigtable master HMaster
Tablet servers

Regions Servers
Bigtable Master

Roles and responsibilities:
- Assigns tablets to tablet servers
- Detects addition and removal of tablet servers
- Balances tablet server load
- Handles garbage collection
- Handles schema changes

Tablet structure changes:
- Table creation/deletion (master initiated)
- Tablet merging (master initiated)
- Tablet splitting (tablet server initiated)
HBase