BigTable: A Distributed Storage System for Structured Data

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Motivation

- Lots of (semi-)structured data at Google.

Motivation

- Lots of (semi-)structured data at Google.
    - URLs: contents, crawl metadata, links, anchors

Motivation

- Lots of (semi-)structured data at Google.
  - URLs: contents, crawl metadata, links, anchors
  - Per-user data: user preferences settings, recent queries, search results

Motivation

- Lots of (semi-)structured data at Google.
  - URLs: contents, crawl metadata, links, anchors
  - Per-user data: user preferences settings, recent queries, search results
  - Geographical locations: physical entities, e.g., shops, restaurants, roads

Motivation

- Lots of (semi-)structured data at Google.
    - URLs: contents, crawl metadata, links, anchors
    - Per-user data: user preferences settings, recent queries, search results
    - Geographical locations: physical entities, e.g., shops, restaurants, roads

- Scale is large

Motivation

- Lots of (semi-)structured data at Google.
  - URLs: contents, crawl metadata, links, anchors
  - Per-user data: user preferences settings, recent queries, search results
  - Geographical locations: physical entities, e.g., shops, restaurants, roads

- Scale is large
  - Billions of URLs, many versions/page - 20KB/page

Motivation

- Lots of (semi-)structured data at Google.
  - URLs: contents, crawl metadata, links, anchors
  - Per-user data: user preferences settings, recent queries, search results
  - Geographical locations: physical entities, e.g., shops, restaurants, roads

- Scale is large
  - Billions of URLs, many versions/page - 20KB/page
  - Hundreds of millions of users, thousands of q/sec - Latency requirement

Motivation

- Lots of (semi-)structured data at Google.
  - URLs: contents, crawl metadata, links, anchors
  - Per-user data: user preferences settings, recent queries, search results
  - Geographical locations: physical entities, e.g., shops, restaurants, roads

- Scale is large
  - Billions of URLs, many versions/page - 20KB/page
  - Hundreds of millions of users, thousands of q/sec - Latency requirement
  - 100+TB of satellite image data

Goals

- Need to support:

Goals

- Need to support:
  - Very high read/write rates (millions of operations per second): Google Talk

Goals

- Need to support:
  - Very high read/write rates (millions of operations per second): Google Talk
  - Efficient scans over all or interesting subset of data.

Goals

- Need to support:
    - Very high read/write rates (millions of operations per second): Google Talk
    - Efficient scans over all or interesting subset of data.
    - Efficient joins of large 1-1 and 1-* datasets.

Goals

- Need to support:
  - Very high read/write rates (millions of operations per second): Google Talk
  - Efficient scans over all or interesting subset of data.
  - Efficient joins of large 1-1 and 1-* datasets.
- Often want to examine data changes over time.
  - Contents of web page over multiple crawls.

BigTable

- Distributed multi-level map

BigTable

- Distributed multi-level map
- Fault-tolerant, persistent

BigTable

- Distributed multi-level map
- Fault-tolerant, persistent
- Scalable
  - 1000s of servers
  - TB of in-memory data
  - Peta byte of disk based data
  - Millions of read/writes per second, efficient scans

BigTable

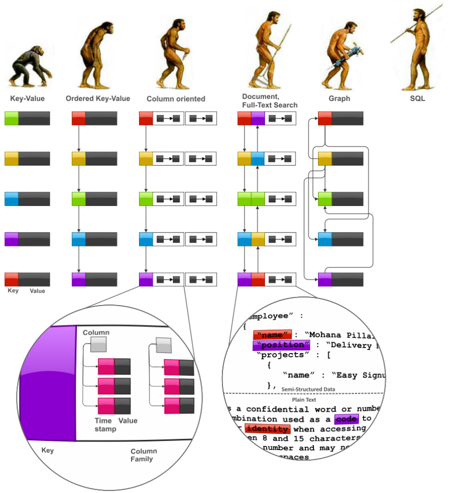
- Distributed multi-level map
- Fault-tolerant, persistent
- Scalable
  - 1000s of servers
  - TB of in-memory data
  - Peta byte of disk based data
  - Millions of read/writes per second, efficient scans
- Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to the load imbalance

BigTable

- Distributed multi-level map
- Fault-tolerant, persistent
- Scalable
  - 1000s of servers
  - TB of in-memory data
  - Peta byte of disk based data
  - Millions of read/writes per second, efficient scans
- Self-managing
  - Servers can be added/removed dynamically
  - Servers adjust to the load imbalance
- CAP: strong consistency and partition tolerance

Data Model

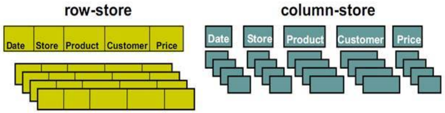
Reminder



[http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques]

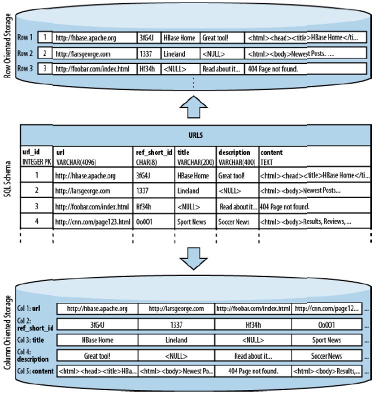
Column-Oriented Data Model (1/2)

- Similar to a key/value store, but the value can have multiple attributes (Columns).
- Column: a set of data values of a particular type.
- Store and process data by column instead of row.



Columns-Oriented Data Model (2/2)

- In many analytical databases queries, few attributes are needed.
- Column values are stored contiguously on disk: reduces I/O.



Row Oriented Storage

Row 1	1	http://hbase.apache.org	3fG4J	HBase Home	Great tool!	<html><head><title>HBase Home</ti...
Row 2	2	http://larsgeorge.com	1337	Lineland	<NULL>	<html><body>Newest Posts. ...
Row 3	3	http://foobar.com/index.html	Hf34h	<NULL>	Read about it...	404 Page not found.

SQL Schema

URLS

url_id INTEGER PK	url VARCHAR(4096)	ref_short_id CHAR(8)	title VARCHAR(200)	description VARCHAR(400)	content TEXT
1	http://hbase.apache.org	3fG4J	HBase Home	Great tool!	<html><head><title>HBase Home</ti...
2	http://larsgeorge.com	1337	Lineland	<NULL>	<html><body>Newest Posts...
3	http://foobar.com/index.html	Hf34h	<NULL>	Read about it...	404 Page not found.
4	http://cnn.com/page123.html	0o001	Sport News	Soccer News	<html><body>Results, Reviews, ...

Column Oriented Storage

Col 1: **url**	http://hbase.apache.org	http://larsgeorge.com	http://foobar.com/index.html	http://cnn.com/page12...	...
Col 2: **ref_short_id**	3fG4J	1337	Hf34h	0o001	...
Col 3: **title**	HBase Home	Lineland	<NULL>	Sport News	...
Col 4: **description**	Great tool!	<NULL>	Read about it...	Soccer News	...
Col 5: **content**	<html><head><title>HBa...	<html><body>Newest Po...	404 Page not found.	<html><body>Results,...	...

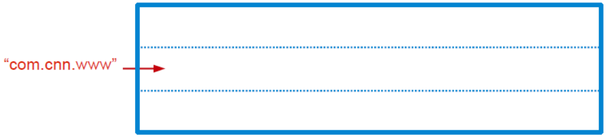
[Lars George, "Hbase: The Definitive Guide", O'Reilly, 2011]

BigTable Data Model (1/5)

- Table
- Distributed multi-dimensional sparse map

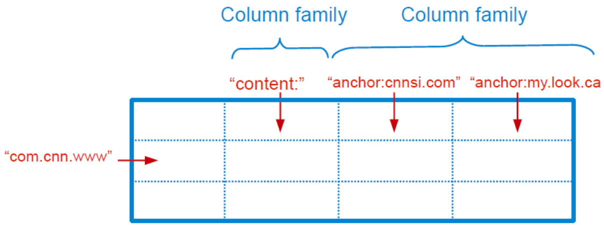
BigTable Data Model (2/5)

- Rows
- Every read or write in a row is atomic.
- Rows sorted in lexicographical order.



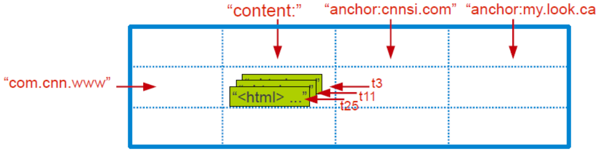
BigTable Data Model (3/5)

- Column
- The basic unit of data access.
- Column families: group of (the same type) column keys.
- Column key naming: family:qualifier



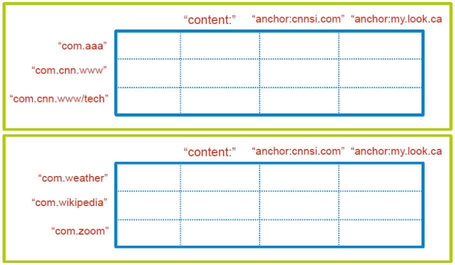
BigTable Data Model (4/5)

- Timestamp
- Each column value may contain multiple versions.



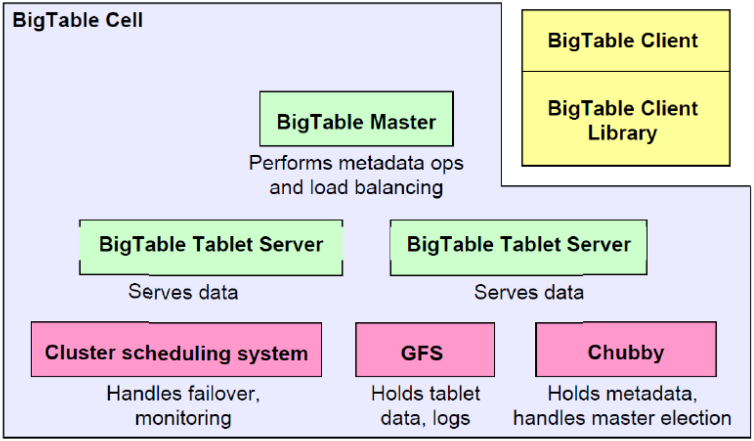
BigTable Data Model (5/5)

- Tablet: contiguous ranges of rows stored together.
- Tables are split by the system when they become too large.
- Auto-Sharding
- Each tablet is served by exactly one tablet server.



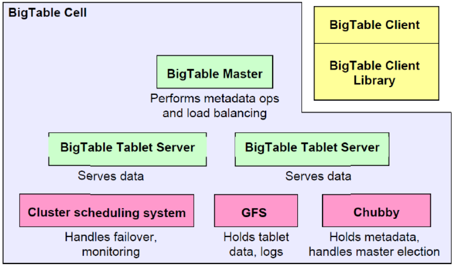
Building Blocks

BigTable Cell



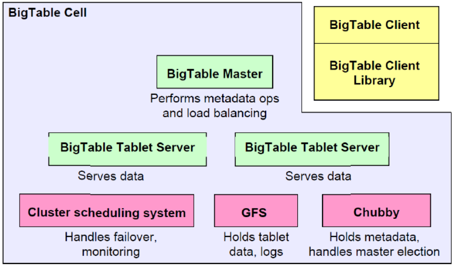
Main Components

- Master server
- Tablet server
- Client library



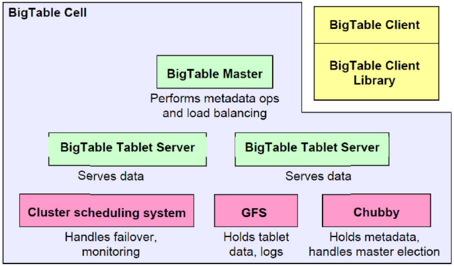
Master Server

- One master server.
- Assigns tablets to tablet server.
- Balances tablet server load.
- Garbage collection of unneeded files in GFS.



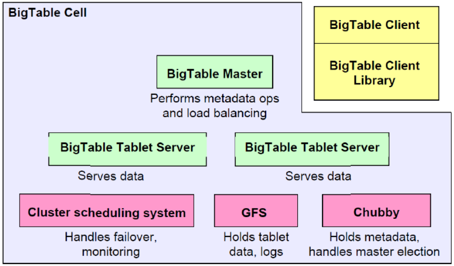
Tablet Server

- Many tablet servers.
- Can be added or removed dynamically.
- Each manages a set of tablets (typically 10-1000 tablets/server).
- Handles read/write requests to tablets.
- Splits tablets when too large.



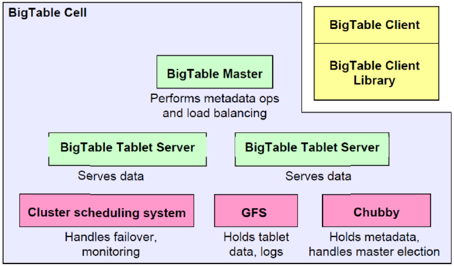
Client Library

- Library that is linked into every client.
- Client data does not move though the master.
- Clients communicate directly with tablet servers for reads/writes.



Building Blocks

- The building blocks for the BigTable are:
  - Google File System (GFS): raw storage
  - Chubby: distributed lock manager
  - Scheduler: schedules jobs onto machines



Google File System (GFS)

- Large-scale distributed file system.
- Master: responsible for metadata.
- Chunk servers: responsible for reading and writing large chunks of data.
- Chunks replicated on 3 machines, master responsible for ensuring replicas exist.

Chubby Lock Service (1/2)

- Name space consists of directories/files used as locks.
- Read/Write to a file are atomic.
- Consists of 5 active replicas: one is elected master and serves requests.
- Needs a majority of its replicas to be running for the service to be alive.
- Uses Paxos to keep its replicas consistent during failures.

Chubby Lock Service (2/2)

- Chubby is used to:
  - Ensure there is only one active master.

Chubby Lock Service (2/2)

- Chubby is used to:
  - Ensure there is only one active master.
  - Store bootstrap location of BigTable data.

Chubby Lock Service (2/2)

- Chubby is used to:
  - Ensure there is only one active master.
  - Store bootstrap location of BigTable data.
  - Discover tablet servers.

Chubby Lock Service (2/2)

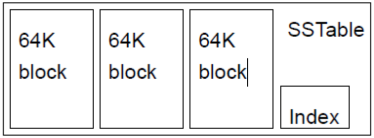
- Chubby is used to:
  - Ensure there is only one active master.
  - Store bootstrap location of BigTable data.
  - Discover tablet servers.
  - Store BigTable schema information.

Chubby Lock Service (2/2)

- Chubby is used to:
  - Ensure there is only one active master.
  - Store bootstrap location of BigTable data.
  - Discover tablet servers.
  - Store BigTable schema information.
  - Store access control lists.

SSTable

- Immutable, sorted file of key-value pairs.
- Chunks of data plus an index.
- Index of block ranges, not values.



Implementation

Tablet Assignment

- 1 tablet $\rightarrow$ 1 tablet server.

Tablet Assignment

- 1 tablet $\rightarrow$ 1 tablet server.
- Master uses Chubby to keep tracks of set of live tablet serves and unassigned tablets.
  - When a tablet server starts, it creates and acquires an exclusive lock in Chubby.

Tablet Assignment

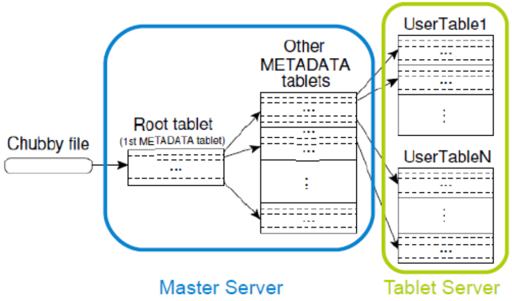
- 1 tablet $\rightarrow$ 1 tablet server.
- Master uses Chubby to keep tracks of set of live tablet serves and unassigned tablets.
    - When a tablet server starts, it creates and acquires an exclusive lock in Chubby.
- Master detects the status of the lock of each tablet server by checking periodically.

Tablet Assignment

- 1 tablet $\rightarrow$ 1 tablet server.
- Master uses Chubby to keep tracks of set of live tablet serves and unassigned tablets.
  - When a tablet server starts, it creates and acquires an exclusive lock in Chubby.
- Master detects the status of the lock of each tablet server by checking periodically.
- Master is responsible for finding when tablet server is no longer serving its tablets and reassigning those tablets as soon as possible.

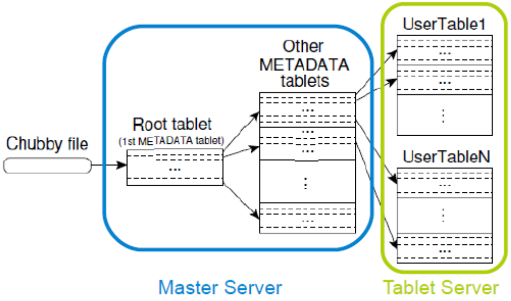
Finding a Tablet

- Three-level hierarchy.



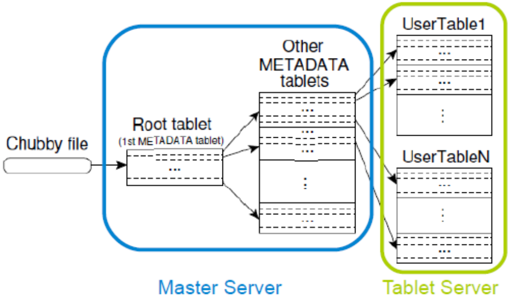
Finding a Tablet

- Three-level hierarchy.
- Root tablet contains location of all tablets in a special METADATA table.



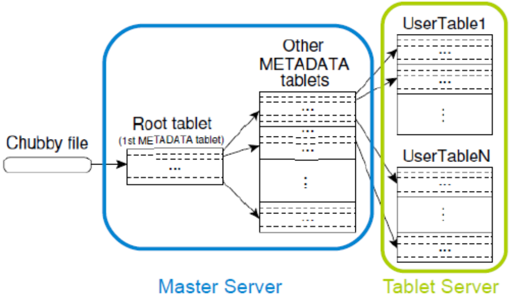
Finding a Tablet

- Three-level hierarchy.
- Root tablet contains location of all tablets in a special METADATA table.
- METADATA table contains location of each tablet under a row.



Finding a Tablet

- Three-level hierarchy.
- Root tablet contains location of all tablets in a special METADATA table.
- METADATA table contains location of each tablet under a row.
- The client library caches tablet locations.



Master Startup

- The master executes the following steps at startup:

Master Startup

- The master executes the following steps at startup:
  - Grabs a unique master lock in Chubby, which prevents concurrent master instantiations.

Master Startup

- The master executes the following steps at startup:
  - Grabs a unique master lock in Chubby, which prevents concurrent master instantiations.
  - Scans the servers directory in Chubby to find the live servers.

Master Startup

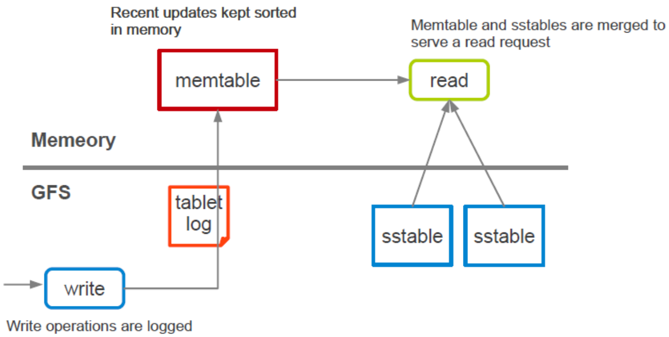
- The master executes the following steps at startup:
  - Grabs a unique master lock in Chubby, which prevents concurrent master instantiations.
  - Scans the servers directory in Chubby to find the live servers.
  - Communicates with every live tablet server to discover what tablets are already assigned to each server.

Master Startup

- The master executes the following steps at startup:
  - Grabs a unique master lock in Chubby, which prevents concurrent master instantiations.
  - Scans the servers directory in Chubby to find the live servers.
  - Communicates with every live tablet server to discover what tablets are already assigned to each server.
  - Scans the METADATA table to learn the set of tablets.

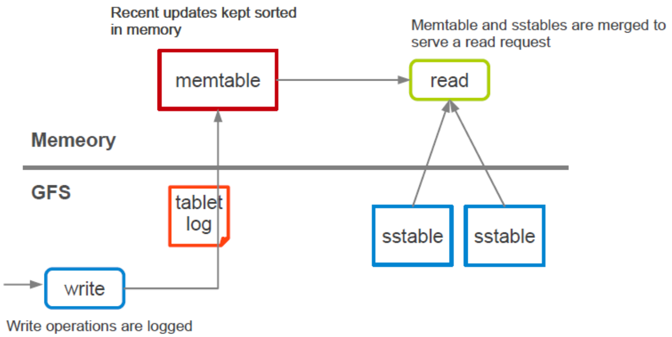
Tablet Serving (1/2)

- Updates committed to a commit log.


Recent updates kept sorted in memory
Memtable and sstables are merged to serve a read request
memtable
read
Memeory
GFS
tablet log
sstable
sstable
write
Write operations are logged

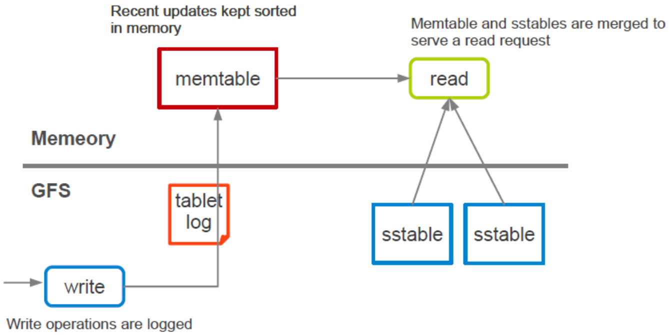
Tablet Serving (1/2)

- Updates committed to a commit log.
- Recently committed updates are stored in memory - memtable


Recent updates kept sorted in memory
Memtable and sstables are merged to serve a read request
memtable
read
Memeory
GFS
tablet log
sstable
sstable
write
Write operations are logged

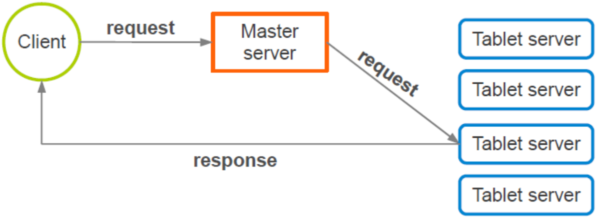
Tablet Serving (1/2)

- Updates committed to a commit log.
- Recently committed updates are stored in memory - memtable
- Older updates are stored in a sequence of SSTables.


Recent updates kept sorted in memory
Memtable and sstables are merged to serve a read request
memtable
read
Memeory
GFS
tablet log
sstable
sstable
write
Write operations are logged

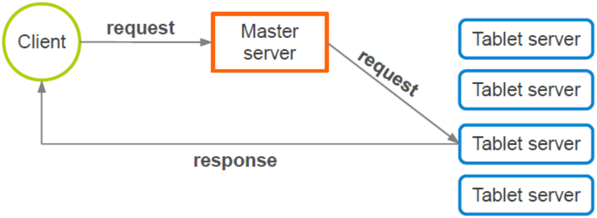
Tablet Serving (2/2)

- Strong consistency
  - Only one tablet server is responsible for a given piece of data.
  - Replication is handled on the GFS layer.



Tablet Serving (2/2)

- Strong consistency
  - Only one tablet server is responsible for a given piece of data.
  - Replication is handled on the GFS layer.
- Tradeoff with availability
  - If a tablet server fails, its portion of data is temporarily unavailable until a new server is assigned.



Compaction

- When in-memory is full

Compaction

- When in-memory is full

- Minor compaction
  - Convert the memtable into an SSTable.
  - Reduce memory usage and log traffic on restart.

Compaction

- When in-memory is full
- Minor compaction
  - Convert the memtable into an SSTable.
  - Reduce memory usage and log traffic on restart.
- Merging compaction
  - Reduces number of SSTables.
  - Reads the contents of a few SSTables and the memtable, and writes out a new SSTable.

Compaction

- When in-memory is full
- Minor compaction
  - Convert the memtable into an SSTable.
  - Reduce memory usage and log traffic on restart.
- Merging compaction
  - Reduces number of SSTables.
  - Reads the contents of a few SSTables and the memtable, and writes out a new SSTable.
- Major compaction
  - Merging compaction that results in only one SSTable.
  - No deleted records, only sensitive live data.

The Bigtable API

The Bigtable API

- Metadata operations
  - Create/delete tables, column families, change metadata

The Bigtable API

- Metadata operations
  - Create/delete tables, column families, change metadata

- Writes: single-row, atomic
  - write/delete cells in a row, delete all cells in a row

The Bigtable API

- Metadata operations
  - Create/delete tables, column families, change metadata
- Writes: single-row, atomic
  - write/delete cells in a row, delete all cells in a row
- Reads: read arbitrary cells in a Bigtable table
  - Each row read is atomic.
  - Can restrict returned rows to a particular range.
  - Can ask for just data from one row, all rows, etc.
  - Can ask for all columns, just certain column families, or specific columns.
  - Can ask for certain timestamps only.

Writing Example

```
// 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);
```

Reading Example

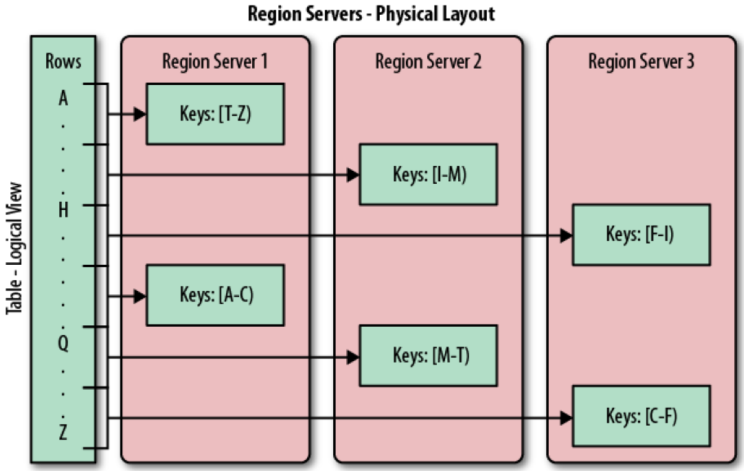
```
Scanner scanner(T);
scanner.Lookup("com.cnn.www");
ScanStream *stream;
stream = scanner.FetchColumnFamily("anchor");
stream->SetReturnAllVersions();

for (; !stream->Done(); stream->Next()) {
  printf("%s %s %lld %s\n",
    scanner.RowName(),
    stream->ColumnName(),
    stream->MicroTimestamp(),
    stream->Value());
}
```

APACHE HBASE

HBase

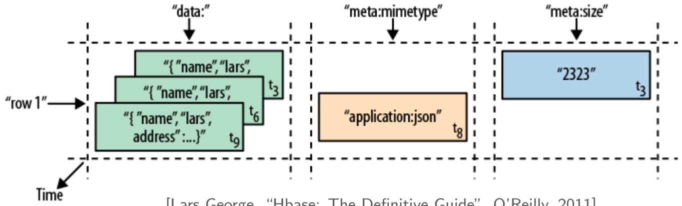
- Type of NoSQL database, based on Google Bigtable
- Column-oriented data store, built on top of HDFS
- CAP: strong consistency and partition tolerance

Region and Region Server



[Lars George, "Hbase: The Definitive Guide", O'Reilly, 2011]

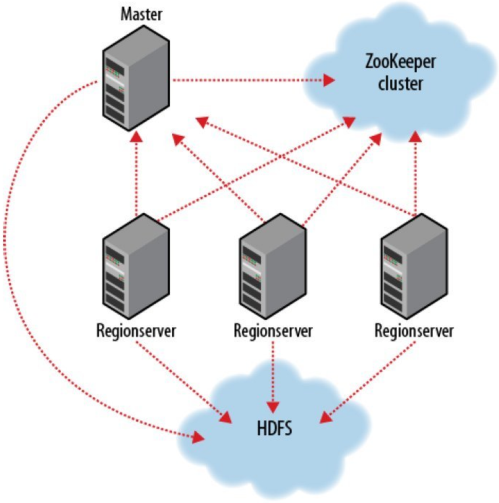
HBase Cell



[Lars George, "Hbase: The Definitive Guide", O'Reilly, 2011]

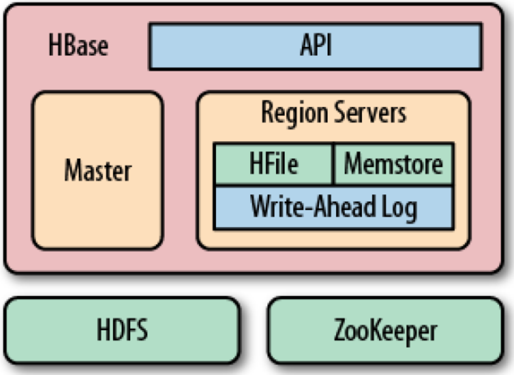
- (Table, RowKey, Family, Column, Timestamp) $\rightarrow$ Value

HBase Cluster



[Tom White, “Hadoop: The Definitive Guide”, O’Reilly, 2012]

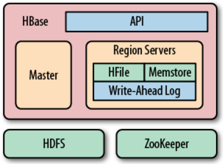
HBase Components



[Lars George, "Hbase: The Definitive Guide", O'Reilly, 2011]

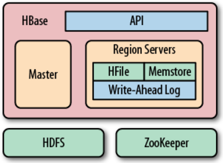
HBase Components - Region Server

- Responsible for all read and write requests for all regions they serve.
- Split regions that have exceeded the thresholds.
- Region servers are added or removed dynamically.



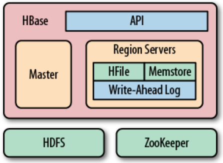
HBase Components - Master

- Responsible for managing regions and their locations.
    - Assigning regions to region servers (uses Zookeeper).
    - Handling load balancing of regions across region servers.
- Doesn't actually store or read data.
    - Clients communicate directly with region servers.
- Responsible for schema management and changes.
    - Adding/removing tables and column families.



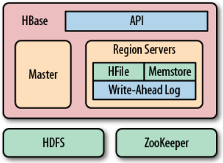
HBase Components - Zookeeper

- A coordinator service: not part of HBase
- Master uses Zookeeper for region assignment.
- Ensures that there is only one master running.
- Stores the bootstrap location for region discovery: a registry for region servers



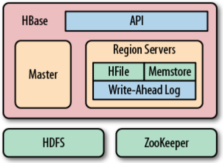
HBase Components - HFile

- The data is stored in HFiles.
- HFiles are immutable sequences of blocks and saved in HDFS.
- Block index is stored at the end of HFiles.
- Cannot remove key-values out of HFiles.
- Delete marker (tombstone marker) indicates the removed records.
  - Hides the marked data from reading clients.
- Updating key/value pairs: picking the latest timestamp.



HBase Components - WAL and memstore

- When data is added it is written to a log called Write Ahead Log (WAL) and is also stored in memory (memstore).
- When in-memory data exceeds maximum value it is flushed to an HFile.



HBase Installation and Shell

HBase Installation

- Three options
  - Local (Standalone) Mode
  - Pseudo-Distributed Mode
  - Fully-Distributed Mode

Installation - Local

- Uses local filesystem (not HDFS).
- Runs HBase and Zookeeper in the same JVM.

Installation - Pseudo-Distributed (1/3)

- Requires HDFS.
- Mimics Fully-Distributed but runs on just one host.
- Good for testing, debugging and prototyping, not for production.
- Configuration files:
  - `hbase-env.sh`
  - `hbase-site.xml`

Installation - Pseudo-Distributed (2/3)

- Specify environment variables in `hbase-env.sh`

```
export JAVA_HOME=/opt/jdk1.7.0_51
```

Installation - Pseudo-Distributed (3/3)

- Starts an HBase Master process, a ZooKeeper server, and a Region-Server process.
- Configure in `hbase-site.xml`

```
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>

<property>
  <name>hbase.rootdir</name>
  <value>hdfs://localhost:8020/hbase</value>
</property>
```

Start HBase and Test

- Start the HBase daemon.

```
start-hbase.sh
hbase shell
```

- Web-based management
  - Master host: `http://localhost:60010`
  - Region host: `http://localhost:60030`

HBase Shell

```
status

list

create 'Blog', {NAME=>'info'}, {NAME=>'content'}

# put 'table', 'row_id', 'family:column', 'value'
put 'Blog', 'Matt-001', 'info:title', 'Elephant'
put 'Blog', 'Matt-001', 'info:author', 'Matt'
put 'Blog', 'Matt-001', 'info:date', '2009.05.06'
put 'Blog', 'Matt-001', 'content:post', 'Do elephants like monkeys?'

# get 'table', 'row_id'
get 'Blog', 'Matt-001'
get 'Blog', 'Matt-001', {COLUMN=>['info:author','content:post']}

scan 'Blog'
```

Summary

Summary

- BigTable
- Column-oriented
- Main components: master, tablet server, client library
- Basic components: GFS, chubby, SSTable
- HBase

Questions?