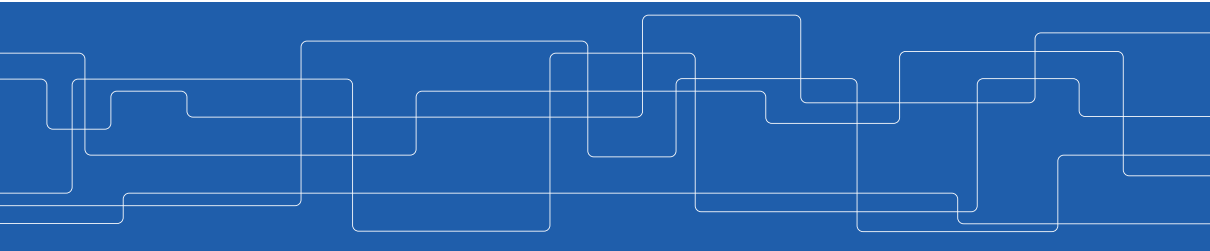
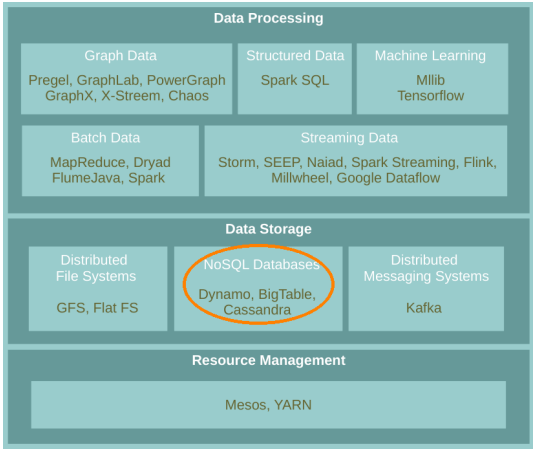
NoSQL Databases

Amir H. Payberah
payberah@kth.se
2025-09-01



Where Are We?


Data Processing
Graph Data
Pregel, GraphLab, PowerGraph
GraphX, X-Streem, Chaos
Structured Data
Spark SQL
Machine Learning
Mllib
Tensorflow
Batch Data
MapReduce, Dryad
FlumeJava, Spark
Streaming Data
Storm, SEEP, Naiad, Spark Streaming, Flink,
Millwheel, Google Dataflow
Data Storage
Distributed
File Systems
GFS, Flat FS
NoSQL Databases
Dynamo, BigTable,
Cassandra
Distributed
Messaging Systems
Kafka
Resource Management
Mesos, YARN

Database and Database Management System

- Database: an organized collection of data.
- Database Management System (DBMS): a software to capture and analyze data.

SQL vs. NoSQL Databases

Relational SQL Databases

- The dominant technology for storing structured data in web and business applications.
- SQL is good
  - Rich language and toolset
  - Easy to use and integrate
  - Many vendors
- They promise: ACID

ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.

ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.
- Consistency
  - A database is in a consistent state before and after a transaction.

ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.
- Consistency
  - A database is in a consistent state before and after a transaction.
- Isolation
  - Transactions can not see uncommitted changes in the database.

ACID Properties

- Atomicity
  - All included statements in a transaction are either executed or the whole transaction is aborted without affecting the database.
- Consistency
  - A database is in a consistent state before and after a transaction.
- Isolation
  - Transactions can not see uncommitted changes in the database.
- Durability
  - Changes are written to a disk before a database commits a transaction so that committed data cannot be lost through a power failure.

SQL Databases Challenges

- Web-based applications caused spikes.
  - Internet-scale data size
  - High read-write rates
  - Frequent schema changes

SQL Databases Challenges

- Web-based applications caused spikes.
  - Internet-scale data size
  - High read-write rates
  - Frequent schema changes
- RDBMS were not designed to be distributed.

NoSQL

- Avoids:
  - Overhead of ACID properties
  - Complexity of SQL query
- Provides:
  - Scalablity
  - Easy and frequent changes to DB
  - Large data volumes

Availability vs. Consistency

Availability

- Replicating data to improve the availability of data.

Availability

- Replicating data to improve the availability of data.
- Data replication
  - Storing data in more than one site or node

Consistency

- Strong consistency
  - After an update completes, any subsequent access will return the updated value.

Consistency

- Strong consistency
  - After an update completes, any subsequent access will return the updated value.

- Eventual consistency
  - Does not guarantee that subsequent accesses will return the updated value.
  - Inconsistency window.
  - If no new updates are made to the object, eventually all accesses will return the last updated value.

Availability vs. Consistency

- The large-scale applications have to be reliable: consistency, availability, partition tolerance

Availability vs. Consistency

- The large-scale applications have to be reliable: consistency, availability, partition tolerance
- Achieving ACID properties on large-scale applications is cahllenging.

Availability vs. Consistency

- The large-scale applications have to be reliable: consistency, availability, partition tolerance
- Achieving ACID properties on large-scale applications is cahllenging.
- CAP theorem

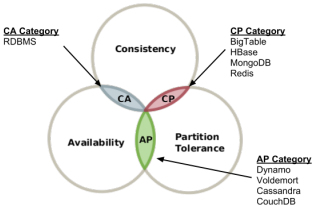
Go to www.menti.com, and use the code 5587 7038

- A Must for Distributed Databases?

1. Consistency
2. Availability
3. Partition Tolerance

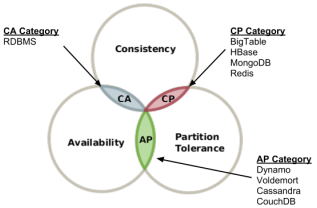
CAP Theorem

- Consistency
  - Consistent state of data after the execution of an operation.



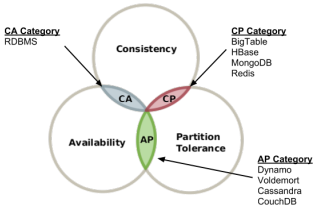
CAP Theorem

- Consistency
  - Consistent state of data after the execution of an operation.
- Availability
  - Clients can always read and write data.



CAP Theorem

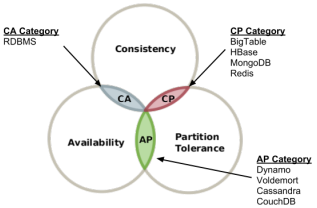
- Consistency
  - Consistent state of data after the execution of an operation.
- Availability
  - Clients can always read and write data.
- Partition Tolerance
  - Continue the operation in the presence of network partitions.



CAP Theorem

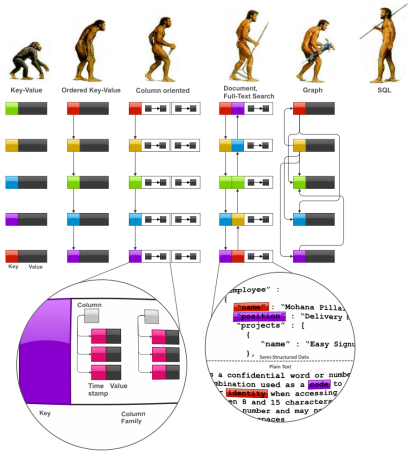
- Consistency
  - Consistent state of data after the execution of an operation.
- Availability
  - Clients can always read and write data.
- Partition Tolerance
  - Continue the operation in the presence of network partitions.

- You can choose only two!



NoSQL Data Models

NoSQL Data Models



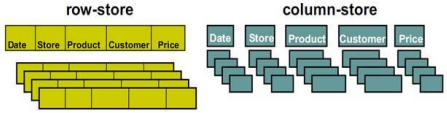
[http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques]

Key-Value Data Model

- Collection of key/value pairs.
- Ordered Key-Value: processing over key ranges.
- Dynamo, Scalaris, Voldemort, Riak, ...

Column-Oriented Data Model

- 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.
- BigTable, Hbase, Cassandra, ...



Document Data Model

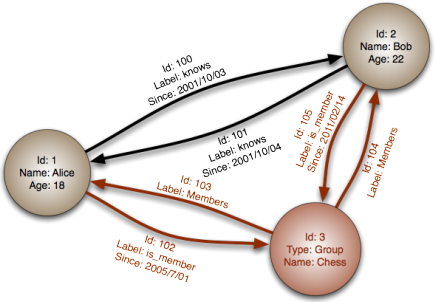
- Similar to a column-oriented store, but values can have complex documents.
- Flexible schema (XML, YAML, JSON, and BSON).
- CouchDB, MongoDB, ...

```
{
    FirstName: "Bob",
    Address: "5 Oak St.",
    Hobby: "sailing"
 }

 {
   FirstName: "Jonathan",
   Address: "15 Wanamassa Point Road",
   Children: [
        {Name: "Michael", Age: 10},
        {Name: "Jennifer", Age: 8},
   ]
  }
```

Graph Data Model

- Uses graph structures with nodes, edges, and properties to represent and store data.
- Neo4J, InfoGrid, ...

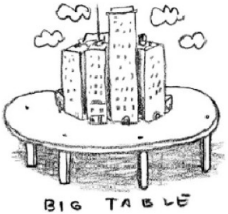


[http://en.wikipedia.org/wiki/Graph_database]

BigTable

BigTable

- Lots of (semi-)structured data at Google.
  - URLs, per-user data, geographical locations, ...
- Distributed multi-level map
- CAP: strong consistency and partition tolerance



Data Model

Data Model (1/5)

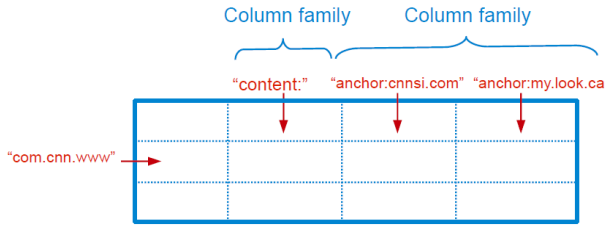
- Table
- Distributed multi-dimensional sparse map

Data Model (2/5)

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

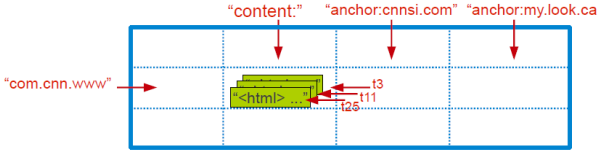
Data Model (3/5)

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



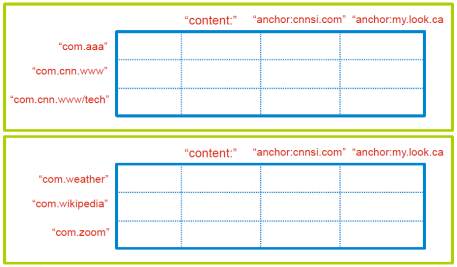
Data Model (4/5)

- Timestamp
- Each column value may contain multiple versions.



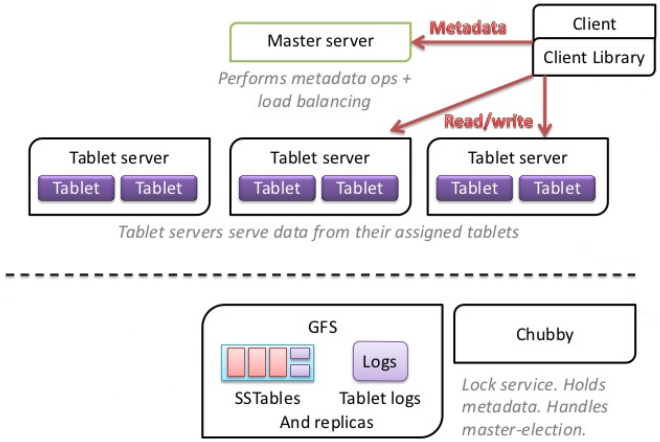
Data Model (5/5)

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



System Architecture

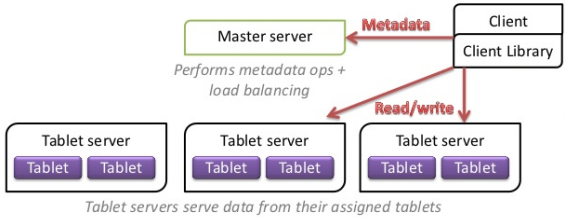
BigTable System Structure



[https://www.slideshare.net/GrishaWeintraub/cap-28353551]

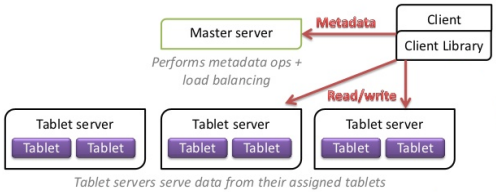
Main Components

- Master
- Tablet server
- Client library



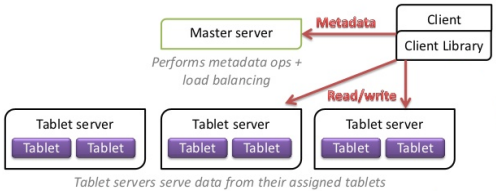
Master

- Assigns tablets to tablet server.



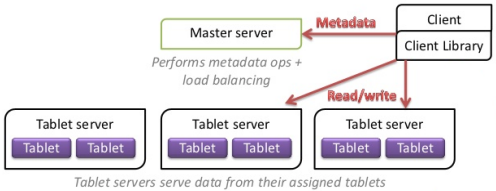
Master

- Assigns tablets to tablet server.
- Balances tablet server load.



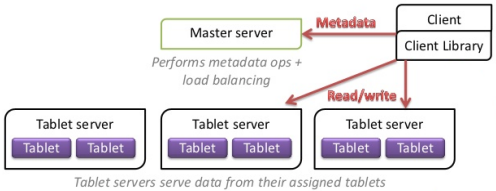
Master

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



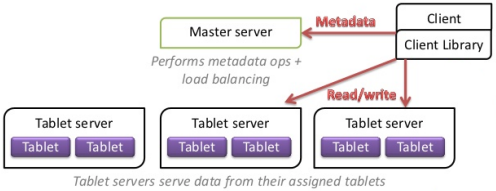
Master

- Assigns tablets to tablet server.
- Balances tablet server load.
- Garbage collection of unneeded files in GFS.
- Handles schema changes, e.g., table and column family creations



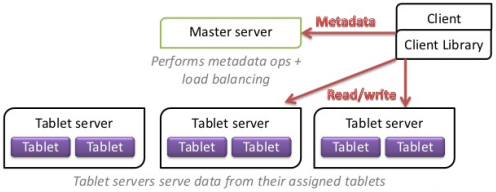
Tablet Server

- Can be added or removed dynamically.



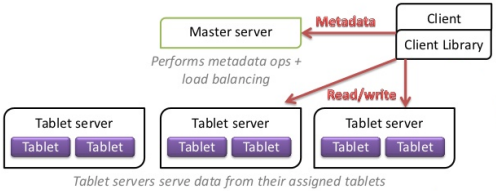
Tablet Server

- Can be added or removed dynamically.
- Each manages a set of tablets (typically 10-1000 tablets/server).



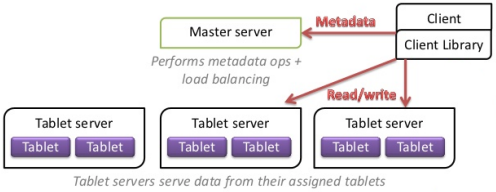
Tablet Server

- Can be added or removed dynamically.
- Each manages a set of tablets (typically 10-1000 tablets/server).
- Handles read/write requests to tablets.



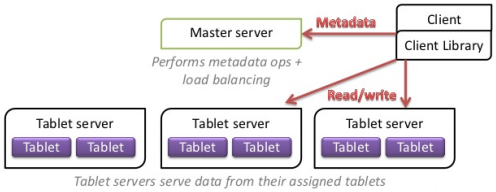
Tablet Server

- 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.



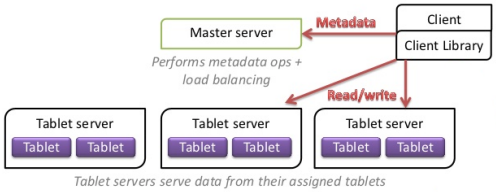
Client Library

- Library that is linked into every client.



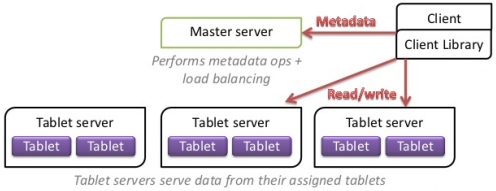
Client Library

- Library that is linked into every client.
- Client data does not move though the master.



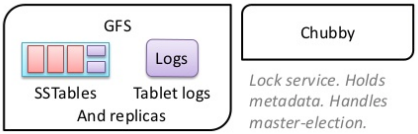
Client Library

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



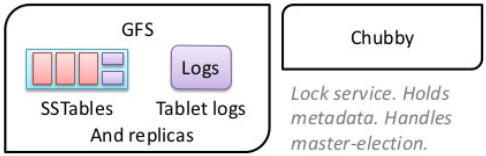
Building Blocks

- The building blocks for the BigTable are:
  - Google File System (GFS)
  - Chubby
  - SSTable



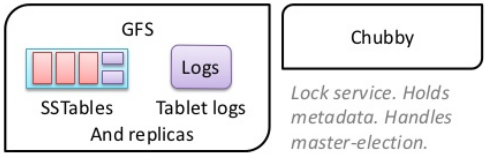
Google File System (GFS)

- Large-scale distributed file system.
- Store log and data files.



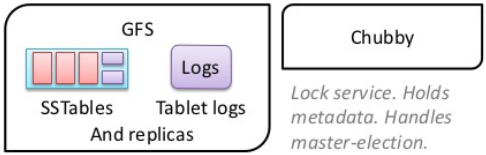
Chubby Lock Service

- Ensure there is only one active master.



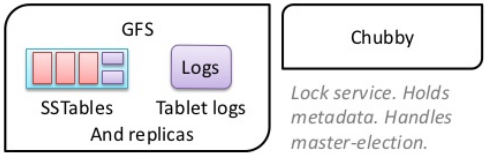
Chubby Lock Service

- Ensure there is only one active master.
- Store bootstrap location of BigTable data.



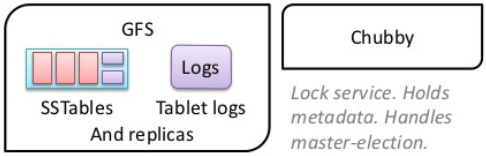
Chubby Lock Service

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



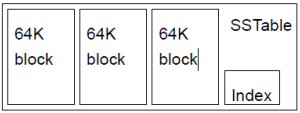
Chubby Lock Service

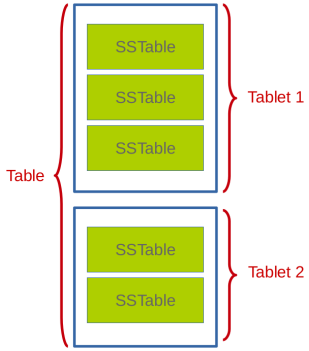
- Ensure there is only one active master.
- Store bootstrap location of BigTable data.
- Discover tablet servers.
- Store BigTable schema information and access control lists.



SSTable

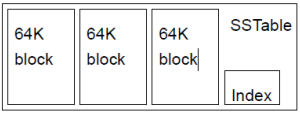
- SSTable file format used internally to store BigTable data.

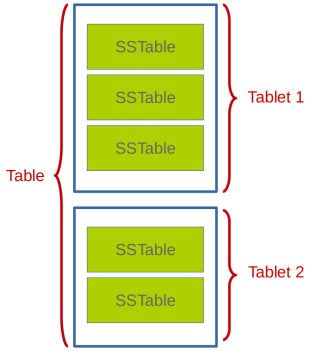




SSTable

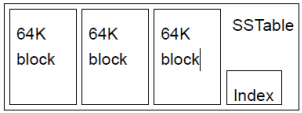
- SSTable file format used internally to store BigTable data.
- Chunks of data plus a block index.

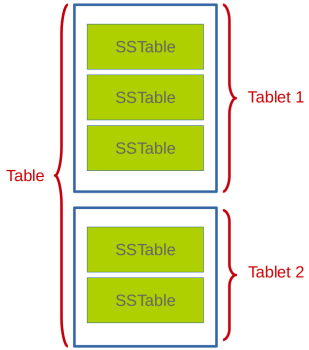




SSTable

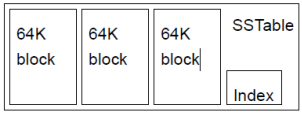
- SSTable file format used internally to store BigTable data.
- Chunks of data plus a block index.
- Immutable, sorted file of key-value pairs.

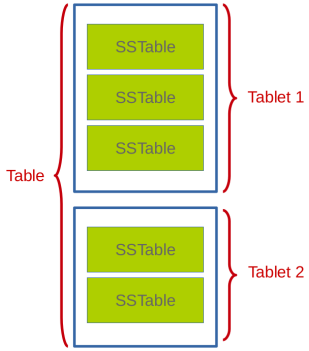




SSTable

- SSTable file format used internally to store BigTable data.
- Chunks of data plus a block index.
- Immutable, sorted file of key-value pairs.
- Each SSTable is stored in a GFS file.





Go to www.menti.com, and use the code 7595 3158

- Who takes care of data replication in BigTable?

1. The Master
2. GFS
3. The Chubby
4. Table Servers

Tablet Assignment

- 1 tablet $\rightarrow$ 1 tablet server.

Tablet Assignment

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

Tablet Assignment

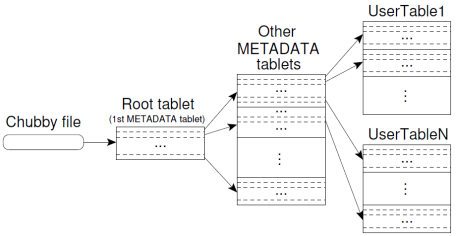
- 1 tablet $\rightarrow$ 1 tablet server.
- Master uses Chubby to keep tracks of live tablet serves and unassigned tablets.
  - When a tablet server starts, it creates and acquires an exclusive lock in Chubby.
- Master detects the status of the lock of each tablet server by checking periodically.

Tablet Assignment

- 1 tablet $\rightarrow$ 1 tablet server.
- Master uses Chubby to keep tracks of 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.

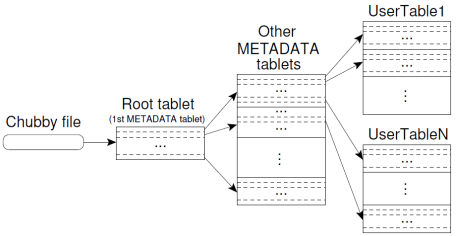
Finding a Tablet

- Three-level hierarchy.



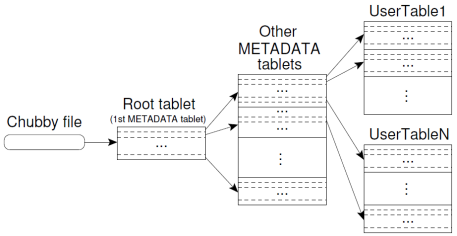
Finding a Tablet

- Three-level hierarchy.
- The first level is a file stored in Chubby that contains the location of the root tablet.



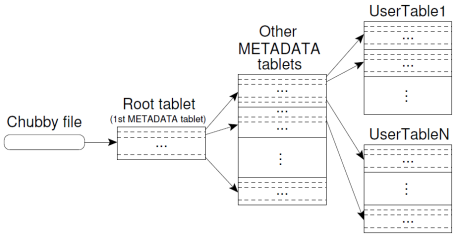
Finding a Tablet

- Three-level hierarchy.
- The first level is a file stored in Chubby that contains the location of the root tablet.
- Root tablet contains location of all tablets in a special METADATA table.



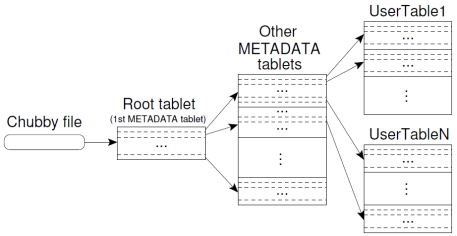
Finding a Tablet

- Three-level hierarchy.
- The first level is a file stored in Chubby that contains the location of the root tablet.
- 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.
- The first level is a file stored in Chubby that contains the location of the root tablet.
- 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.



Loading Tablets

- To load a tablet, a tablet server does the following:

Loading Tablets

- To load a tablet, a tablet server does the following:
- Finds locaton of tablet through its METADATA.
  - Metadata for a tablet includes list of SSTables and set of redo points.

Loading Tablets

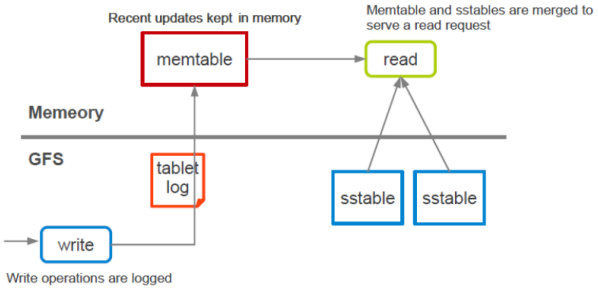
- To load a tablet, a tablet server does the following:
- Finds locaton of tablet through its METADATA.
  - Metadata for a tablet includes list of SSTables and set of redo points.
- Read SSTables index blocks into memory.

Loading Tablets

- To load a tablet, a tablet server does the following:
- Finds locaton of tablet through its METADATA.
  - Metadata for a tablet includes list of SSTables and set of redo points.
- Read SSTables index blocks into memory.
- Read the commit log since the redo point and reconstructs the memtable.

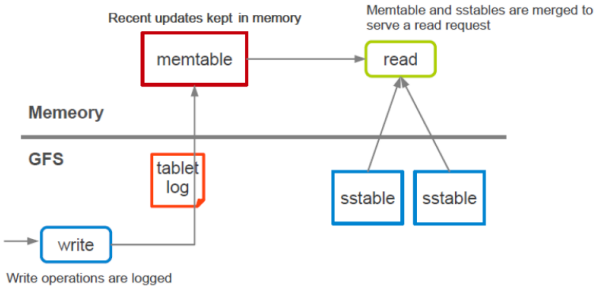
Tablet Serving (1/2)

- Updates committed to a commit log.



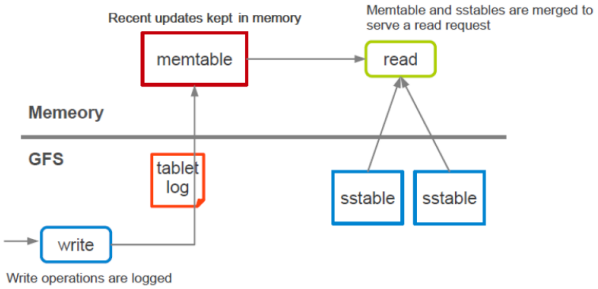
Tablet Serving (1/2)

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



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.



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.
- Trade-off with availability
  - If a tablet server fails, its portion of data is temporarily unavailable until a new server is assigned.

BigTable vs. HBase

BigTable	HBase
GFS	HDFS
Tablet Server	Region Server
SSTable	StoreFile
Memtable	MemStore
Chubby	ZooKeeper

Cassandra

Cassandra

- A column-oriented database
- It was created for Facebook and was later open sourced
- CAP: availability and partition tolerance

Borrowed From BigTable

- Data model: column oriented
  - Keyspaces (similar to the schema in a relational database), tables, and columns.

Borrowed From BigTable

- Data model: column oriented
  - Keyspaces (similar to the schema in a relational database), tables, and columns.
- SSTable disk storage
  - Append-only commit log
  - Memtable (buffering and sorting)
  - Immutable sstable files

Data Partitioning (1/2)

- Key/value, where values are stored as objects.
- If size of data exceeds the capacity of a single machine: partitioning

Data Partitioning (1/2)

- Key/value, where values are stored as objects.
- If size of data exceeds the capacity of a single machine: partitioning
- Consistent hashing for partitioning.

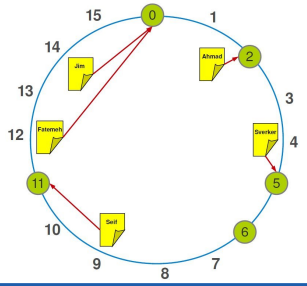
Data Partitioning (2/2)

- Consistent hashing.
- Hash both data and node ids using the same hash function in a same id space.
- `partition = hash(d) mod n`, `d`: data, `n`: the size of the id space

Data Partitioning (2/2)

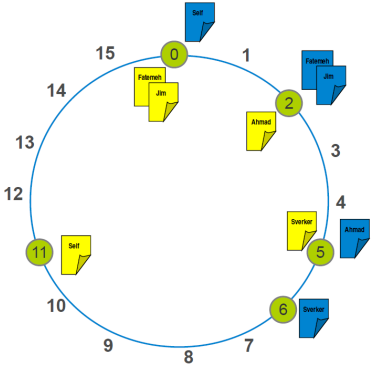
- Consistent hashing.
- Hash both data and node ids using the same hash function in a same id space.
- `partition = hash(d) mod n`, `d`: data, `n`: the size of the id space

```
id space = [0, 15], n = 16
hash("Fatemeh") = 12
hash("Ahmad") = 2
hash("Seif") = 9
hash("Jim") = 14
hash("Sverker") = 4
```



Replication

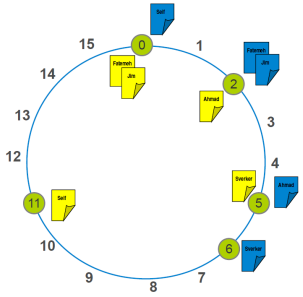
- To achieve high availability and durability, data should be replicated on multiple nodes.



Go to www.menti.com, and use the code 8650 0544

- `hash("A") = 15` and `hash("B") = 5`?

1. 0 and 5
2. 15 and 5
3. 0 and 6



NoSQL Through a Data Feminism Lens (1/2)

- Who holds power?

NoSQL Through a Data Feminism Lens (1/2)

- Who holds power?
  - Corporate-managed (Google, AWS, Microsoft).
  - Users depend on vendors for infrastructure, pricing, and availability.

NoSQL Through a Data Feminism Lens (1/2)

- Who holds power?
  - Corporate-managed (Google, AWS, Microsoft).
  - Users depend on vendors for infrastructure, pricing, and availability.
- Who gets excluded?

NoSQL Through a Data Feminism Lens (1/2)

- Who holds power?
  - Corporate-managed (Google, AWS, Microsoft).
  - Users depend on vendors for infrastructure, pricing, and availability.
- Who gets excluded?
  - High costs, proprietary APIs, and limited offline capabilities exclude small orgs, NGOs, and low-connectivity regions.

NoSQL Through a Data Feminism Lens (2/2)

- Who bears the environmental cost?

NoSQL Through a Data Feminism Lens (2/2)

- Who bears the environmental cost?
  - Always-on clusters = high energy and water use
  - Carbon costs externalized to hosting regions

NoSQL Through a Data Feminism Lens (2/2)

- Who bears the environmental cost?
  - Always-on clusters = high energy and water use
  - Carbon costs externalized to hosting regions
- Whose labor is invisible?

NoSQL Through a Data Feminism Lens (2/2)

- Who bears the environmental cost?
  - Always-on clusters = high energy and water use
  - Carbon costs externalized to hosting regions
- Whose labor is invisible?
  - Hardware built from mined minerals under unsafe conditions
  - Outsourced teams handle maintenance and data prep.

Questions

- If BigTable or Cassandra were reimagined for equity, inclusion, and sustainability, what would change in their architecture, governance and replication strategy?

Possible Answers

- Architecture

Possible Answers

- Architecture
  - Allow flexible representation (e.g., non-binary genders, multilingual text).
  - Context-aware sharding, i.e., data sharding designed to respect community.
  - Built-in mechanisms for deletion.

Possible Answers

- Architecture
  - Allow flexible representation (e.g., non-binary genders, multilingual text).
  - Context-aware sharding, i.e., data sharding designed to respect community.
  - Built-in mechanisms for deletion.
- Governance

Possible Answers

- Architecture
  - Allow flexible representation (e.g., non-binary genders, multilingual text).
  - Context-aware sharding, i.e., data sharding designed to respect community.
  - Built-in mechanisms for deletion.

- Governance
  - Shared authority on table design, retention, and access policies.
  - Transparent query auditing, i.e., logs visible to users and communities, not just operators.
  - Publish energy and carbon use alongside system metrics.

Possible Answers

- Architecture
  - Allow flexible representation (e.g., non-binary genders, multilingual text).
  - Context-aware sharding, i.e., data sharding designed to respect community.
  - Built-in mechanisms for deletion.
- Governance
  - Shared authority on table design, retention, and access policies.
  - Transparent query auditing, i.e., logs visible to users and communities, not just operators.
  - Publish energy and carbon use alongside system metrics.
- Replication Strategy

Possible Answers

- Architecture
  - Allow flexible representation (e.g., non-binary genders, multilingual text).
  - Context-aware sharding, i.e., data sharding designed to respect community.
  - Built-in mechanisms for deletion.
- Governance
  - Shared authority on table design, retention, and access policies.
  - Transparent query auditing, i.e., logs visible to users and communities, not just operators.
  - Publish energy and carbon use alongside system metrics.
- Replication Strategy
  - Stricter consistency for sensitive health data, lighter for public archives.
  - Adapt replication based on renewable energy availability.

Feminist-Aligned Alternative Systems

- CouchDB
  - Decentralized and offline-first
  - Local replicas support underserved regions and data sovereignty

Feminist-Aligned Alternative Systems

- CouchDB
  - Decentralized and offline-first
  - Local replicas support underserved regions and data sovereignty
- OrbitDB
  - Peer-to-peer, IPFS-based
  - No central authority, runs in browsers or low-power devices

Feminist-Aligned Alternative Systems

- CouchDB
  - Decentralized and offline-first
  - Local replicas support underserved regions and data sovereignty
- OrbitDB
  - Peer-to-peer, IPFS-based
  - No central authority, runs in browsers or low-power devices
- GunDB
  - Decentralized graph database with user-owned, encrypted data
  - Works offline-first

Questions

- What trade-offs might we face when choosing decentralized or community-controlled NoSQL over BigTable or Cassandra?

Possible Answers

- Performance vs. Autonomy: lower throughput/latency than hyperscale systems, but more local control.

Possible Answers

- Performance vs. Autonomy: lower throughput/latency than hyperscale systems, but more local control.
- Simplicity vs. Flexibility: community systems may lack advanced features or tooling, but are easier to adapt to local needs.

Possible Answers

- Performance vs. Autonomy: lower throughput/latency than hyperscale systems, but more local control.
- Simplicity vs. Flexibility: community systems may lack advanced features or tooling, but are easier to adapt to local needs.
- Resilience vs. Cost: fewer replicas $->$ lower energy/cost, but higher risk of data loss.

Possible Answers

- Performance vs. Autonomy: lower throughput/latency than hyperscale systems, but more local control.
- Simplicity vs. Flexibility: community systems may lack advanced features or tooling, but are easier to adapt to local needs.
- Resilience vs. Cost: fewer replicas $->$ lower energy/cost, but higher risk of data loss.
- Global Reach vs. Sovereignty: corporate clouds offer worldwide availability, while community NoSQL prioritizes jurisdictional/local storage.

Possible Answers

- Performance vs. Autonomy: lower throughput/latency than hyperscale systems, but more local control.
- Simplicity vs. Flexibility: community systems may lack advanced features or tooling, but are easier to adapt to local needs.
- Resilience vs. Cost: fewer replicas $->$ lower energy/cost, but higher risk of data loss.
- Global Reach vs. Sovereignty: corporate clouds offer worldwide availability, while community NoSQL prioritizes jurisdictional/local storage.
- Support vs. Self-reliance: bigTech = professional 24/7 support; community systems rely on shared governance and volunteer labor.

Summary

Summary

- NoSQL data models: key-value, column-oriented, document-oriented, graph-based
- CAP (Consistency vs. Availability)

Summary

- NoSQL data models: key-value, column-oriented, document-oriented, graph-based
- CAP (Consistency vs. Availability)
- BigTable
  - Column-oriented
  - Basic components: GFS, SSTable, Chubby

Summary

- NoSQL data models: key-value, column-oriented, document-oriented, graph-based
- CAP (Consistency vs. Availability)
- BigTable
  - Column-oriented
  - Basic components: GFS, SSTable, Chubby
- Cassandra
  - Column-oriented
  - Consistency hashing

Summary

- NoSQL data models: key-value, column-oriented, document-oriented, graph-based
- CAP (Consistency vs. Availability)
- BigTable
  - Column-oriented
  - Basic components: GFS, SSTable, Chubby
- Cassandra
  - Column-oriented
  - Consistency hashing
- Alternative systems, e.g., CouchDB, OrbitDB, GunDB

References

- F. Chang et al., Bigtable: A distributed storage system for structured data, ACM Transactions on Computer Systems (TOCS) 26.2, 2008.
- A. Lakshman et al., Cassandra: a decentralized structured storage system, ACM SIGOPS Operating Systems Review 44.2, 2010.
- J.C. Anderson et al., CouchDB: The Definitive Guide, O'Reilly Media, 2010.
- OrbitDB: https://orbitdb.org
- GunDB: https://gun.eco

Questions?