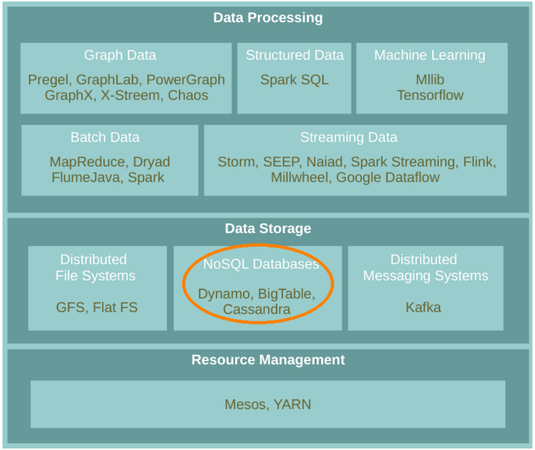
NoSQL Databases

Amir H. Payberah
payberah@kth.se
03/09/2018

The Course Web Page

`https://id2221kth.github.io`

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 that interacts with users, other applications, and the database itself to capture and analyze data.

Relational Databases Management Systems (RDMBSs)

- RDMBSs: 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.

RDBMS Challenges

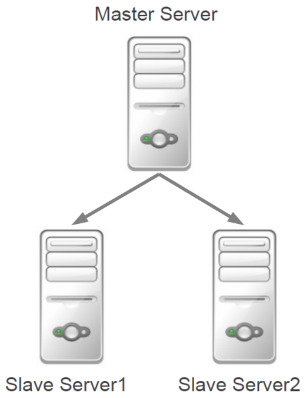
- Web-based applications caused spikes.
  - Internet-scale data size
  - High read-write rates
  - Frequent schema changes

Let's Scale RDBMSs

- RDBMS were not designed to be distributed.
- Possible solutions:
  - Replication
  - Sharding

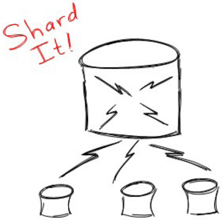
Let's Scale RDBMSs - Replication

- Master/Slave architecture
- Scales read operations

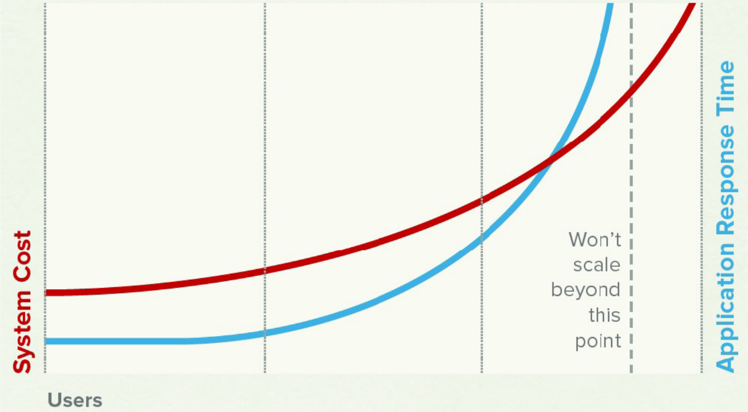


Let's Scale RDBMSs - Sharding

- Dividing the database across many machines.
- It scales read and write operations.
- Cannot execute transactions across shards (partitions).



Scaling RDBMSs is Expensive and Inefficient



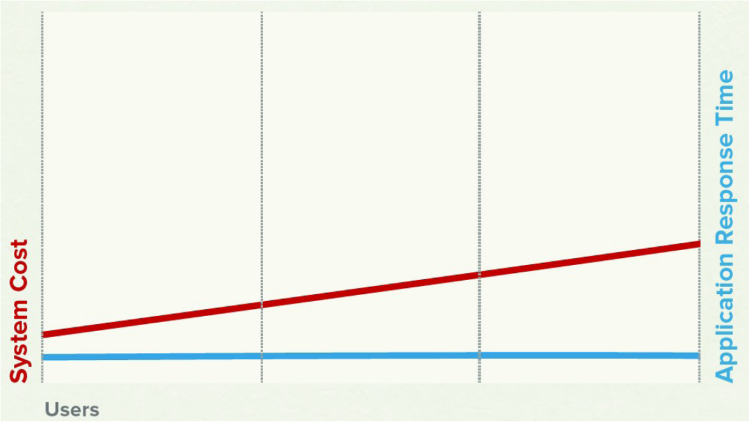
[http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQLWhitepaper.pdf]


Not Only SQL

NoSQL

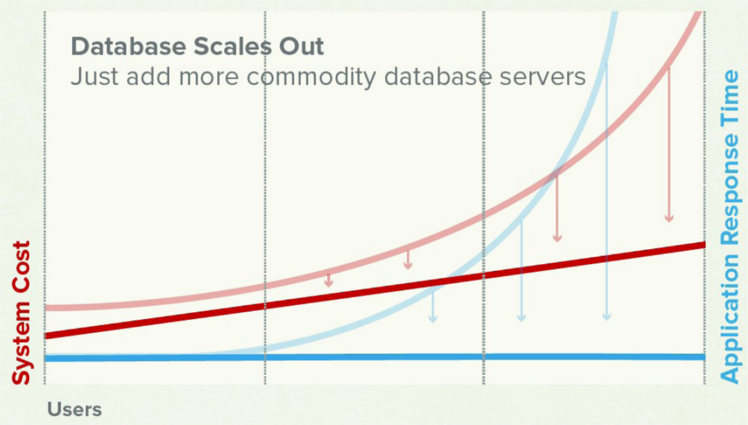
- Avoids:
  - Overhead of ACID properties
  - Complexity of SQL query
- Provides:
  - Scalablity
  - Easy and frequent changes to DB
  - Large data volumes

NoSQL Cost and Performance



[http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQLWhitepaper.pdf]

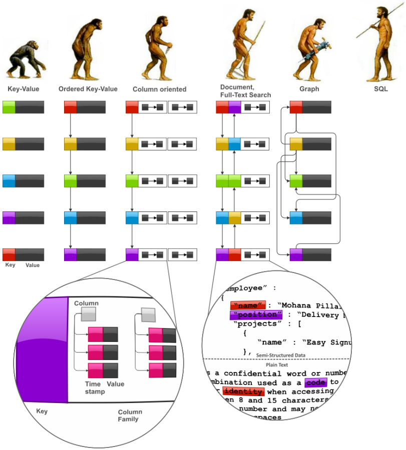
RDBMS vs. NoSQL



[http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQLWhitepaper.pdf]

NoSQL Data Models

NoSQL Data Models



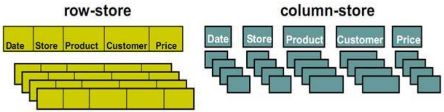
[http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques]

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, ...



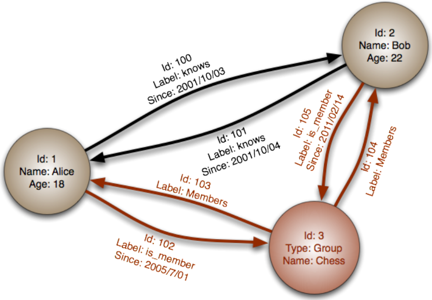
Document Data Model

- 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, ...

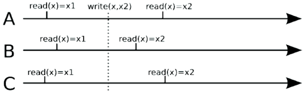


[http://en.wikipedia.org/wiki/Graph_database]

Consistency

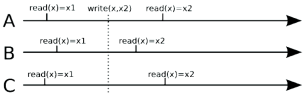
Consistency

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

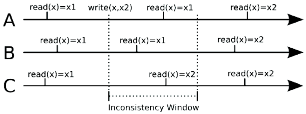


Consistency

- 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.



Quorum Model

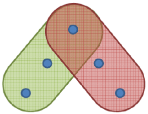
- N: the number of nodes to which a data item is replicated.
- R: the number of nodes a value has to be read from to be accepted.
- W: the number of nodes a new value has to be written to before the write operation is finished.

- To enforce strong consistency: $R + W > N$

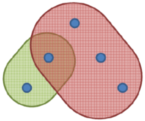
Quorum Model

- N: the number of nodes to which a data item is replicated.
- R: the number of nodes a value has to be read from to be accepted.
- W: the number of nodes a new value has to be written to before the write operation is finished.

- To enforce strong consistency: $R + W > N$



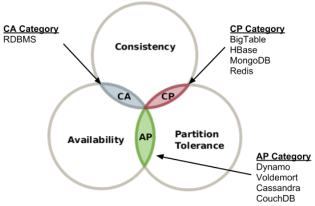
R = 3, W = 3, N = 5



R = 4, W = 2, N = 5

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!

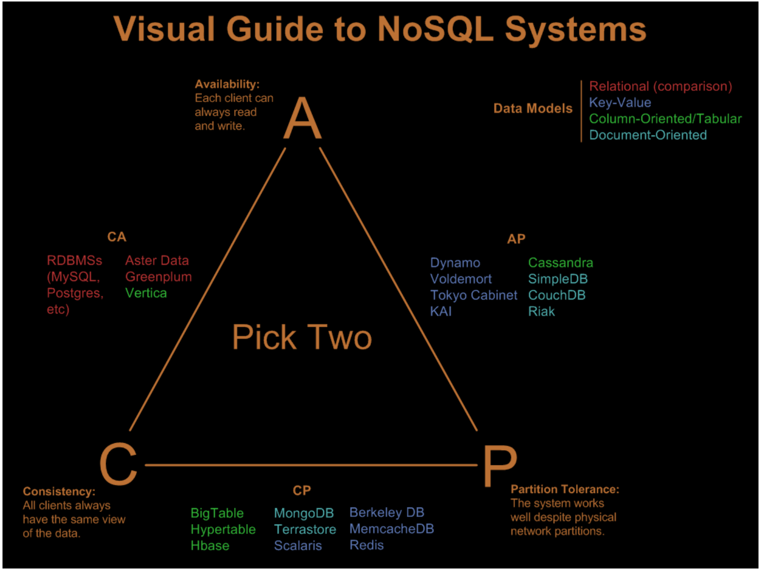


Consistency vs. Availability

- The large-scale applications have to be reliable: availability + partition tolerance
- These properties are difficult to achieve with ACID properties.
- The BASE approach forfeits the ACID properties of consistency and isolation in favor of availability and performance.

BASE Properties

- Basic Availability
  - Possibilities of faults but not a fault of the whole system.
- Soft-state
  - Copies of a data item may be inconsistent
- Eventually consistent
  - Copies becomes consistent at some later time if there are no more updates to that data item


Visual Guide to NoSQL Systems
Availability:
Each client can always read and write.
A
Data Models
Relational (comparison)
Key-Value
Column-Oriented/Tabular
Document-Oriented
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Pick Two
C
P
Consistency:
All clients always have the same view of the data.
CP
BigTable
Hypertable
Hbase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance:
The system works well despite physical network partitions.

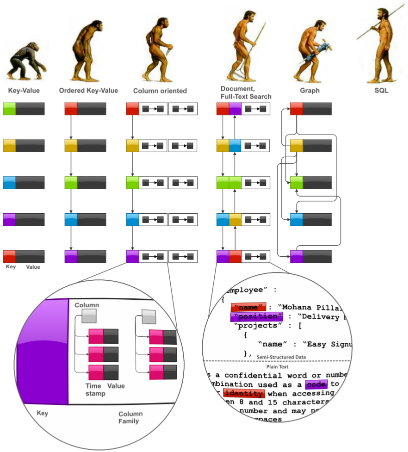
Dyanmo

Dynamo

- Distributed key/value storage system
- Scalable and Highly available
- CAP: it sacrifices strong consistency for availability: always writable

Data Model

Data Model



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

Partitioning

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

Partitioning

- Key/value, where values are stored as objects.
- If size of data exceeds the capacity of a single machine: partitioning
- Consistent hashing is one form of sharding (partitioning).

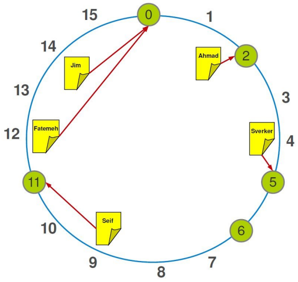
Consistent Hashing

- Hash both data and nodes using the same hash function in a same id space.
- `partition = hash(d) mod n`, `d`: data, `n`: number of nodes

Consistent Hashing

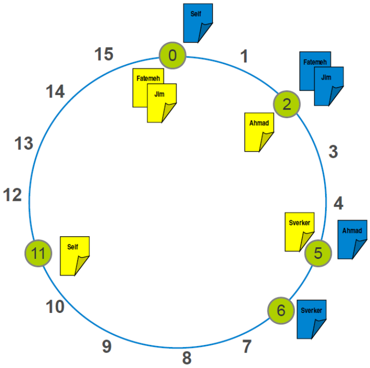
- Hash both data and nodes using the same hash function in a same id space.
- `partition = hash(d) mod n`, `d`: data, `n`: number of nodes

```
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 replicates on multiple nodes.



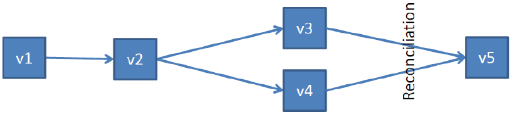
Data Consistency

Data Consistency

- Eventual consistency: updates are propagated asynchronously.
- Each update/modification of an item results in a new and immutable version of the data.
  - Multiple versions of an object may exist.
- Replicas eventually become consistent.

Data Versioning (1/2)

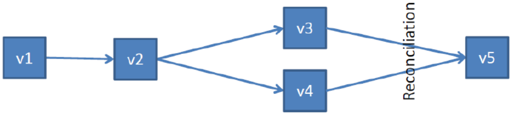
- Use vector clocks for capturing causality, in the form of (node, counter)
  - If causal: older version can be forgotten
  - If concurrent: conflict exists, requiring reconciliation



Data Versioning (1/2)

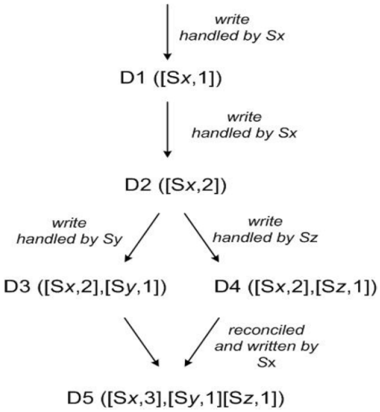
- Use vector clocks for capturing causality, in the form of (node, counter)
  - If causal: older version can be forgotten
  - If concurrent: conflict exists, requiring reconciliation

- Version branching can happen due to node/network failures.



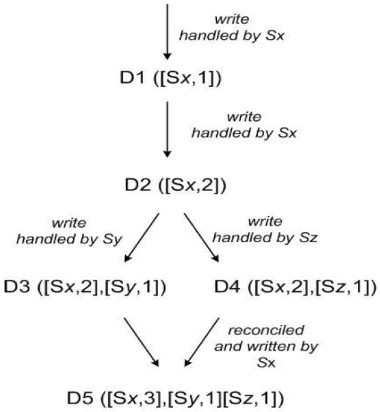
Data Versioning (2/2)

- Client C1 writes new object via Sx.



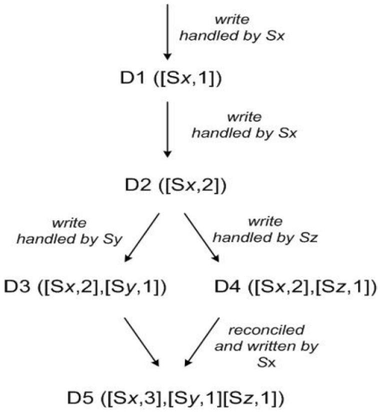
Data Versioning (2/2)

- Client C1 writes new object via Sx.
- C1 updates the object via Sx.



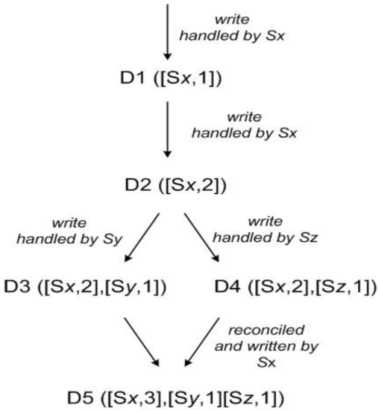
Data Versioning (2/2)

- Client C1 writes new object via Sx.
- C1 updates the object via Sx.
- C1 updates the object via Sy.



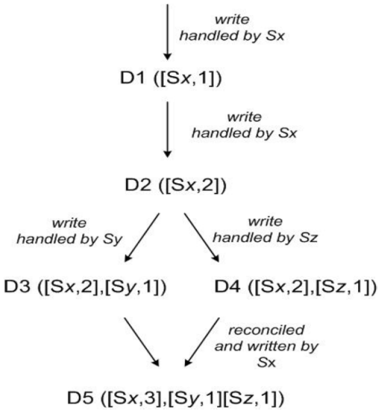
Data Versioning (2/2)

- Client C1 writes new object via Sx.
- C1 updates the object via Sx.
- C1 updates the object via Sy.
- C2 reads D2 and updates the object via Sz.



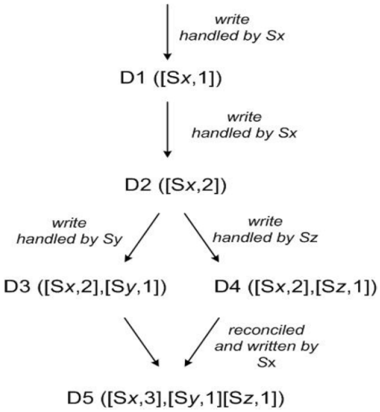
Data Versioning (2/2)

- Client C1 writes new object via Sx.
- C1 updates the object via Sx.
- C1 updates the object via Sy.
- C2 reads D2 and updates the object via Sz.
- C3 reads D3 and D4 via Sx.
  - The read context is a summary of the clocks of D3 and D4: [(Sx, 2), (Sy, 1), (Sz, 1)].



Data Versioning (2/2)

- Client C1 writes new object via Sx.
- C1 updates the object via Sx.
- C1 updates the object via Sy.
- C2 reads D2 and updates the object via Sz.
- C3 reads D3 and D4 via Sx.
  - The read context is a summary of the clocks of D3 and D4: [(Sx, 2), (Sy, 1), (Sz, 1)].
- Reconciliation



Dynamo API

Dynamo API

- `get(key)`
  - Return single object or list of objects with conflicting version and context.
- `put(key, context, object)`
  - Store object and context under key.
  - Context encodes system metadata, e.g., version number.

put Operation

- Coordinator generates new vector clock and writes the new version locally.
- Send to N nodes.
- Wait for response from W nodes.

get Operation

- Coordinator requests existing versions from N.
  - Wait for response from R nodes.
- If multiple versions, return all versions that are causally unrelated.
- Divergent versions are then reconciled.
- Reconciled version written back.

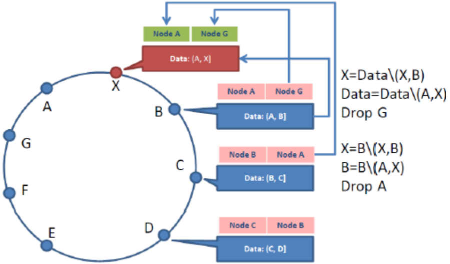
Membership Management

Membership Management

- Administrator explicitly adds and removes nodes.
- Gossiping to propagate membership changes.
  - Eventually consistent view.
  - $O(1)$ hop overlay.

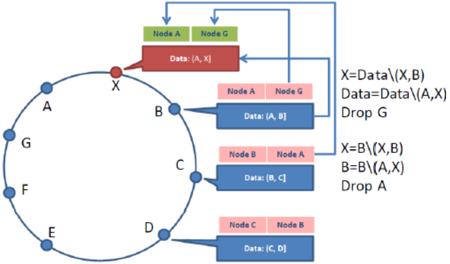
Adding and Removing Nodes

- A new node X added to system.
  - X is assigned key ranges w.r.t. its virtual servers.
  - For each key range, it transfers the data items.



Adding and Removing Nodes

- A new node X added to system.
  - X is assigned key ranges w.r.t. its virtual servers.
  - For each key range, it transfers the data items.



- Removing a node: reallocation of keys is a reverse process of adding nodes.

Failure Detection

- Passive failure detection.
  - Use pings only for detection from failed to alive.
- In the absence of client requests, node A doesn't need to know if node B is alive.

BigTable

Motivation

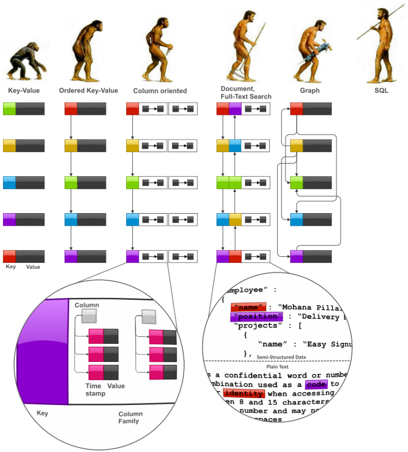
- Lots of (semi-)structured data at Google.
  - URLs, TextGreenper-user data, geographical locations, ...
- Big data
  - Billions of URLs, hundreds of millions of users, 100+TB of satellite image data, ...

BigTable

- Distributed multi-level map
- Fault-tolerant
- Scalable and self-managing
- CAP: strong consistency and partition tolerance

Data Model

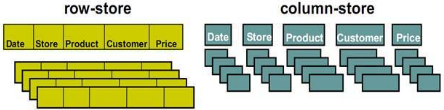
Data Model



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

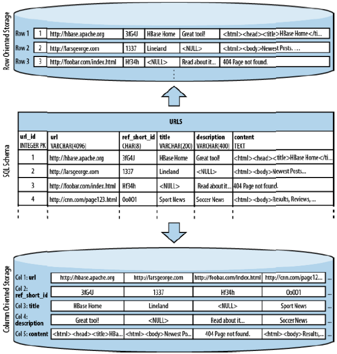
Column-Oriented Data Model (1/2)

- 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.



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

BigTable Data Model (1/5)

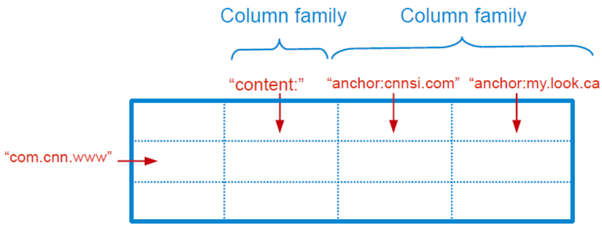
- Table
- Distributed multi-dimensional sparse map

BigTable Data Model (2/5)

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

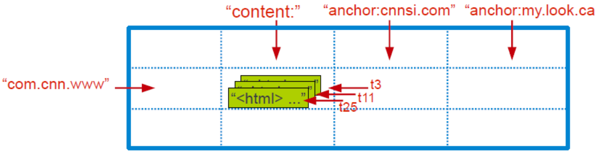
BigTable Data Model (3/5)

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



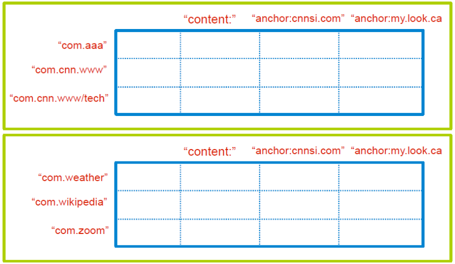
BigTable Data Model (4/5)

- Timestamp
- Each column value may contain multiple versions.



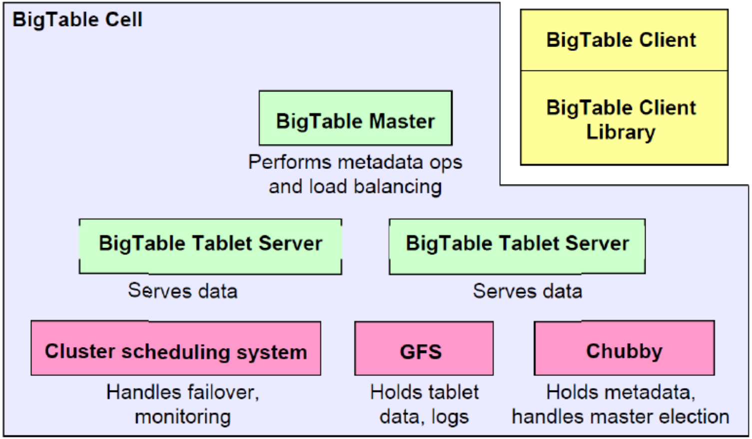
BigTable Data Model (5/5)

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



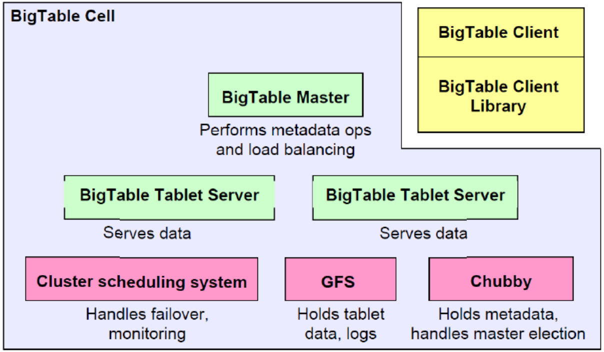
BigTable Architecture

BigTable Cell



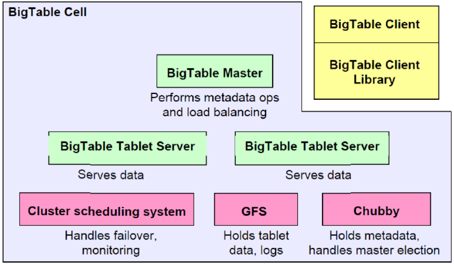
Main Components

- Master server
- Tablet server
- Client library



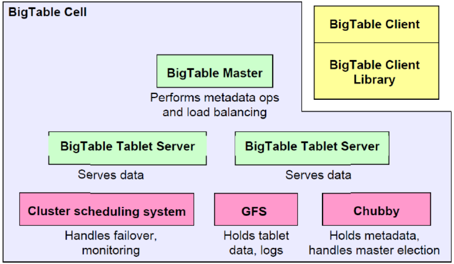
Master Server

- 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



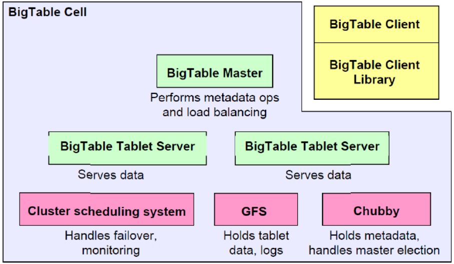
Tablet Server

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



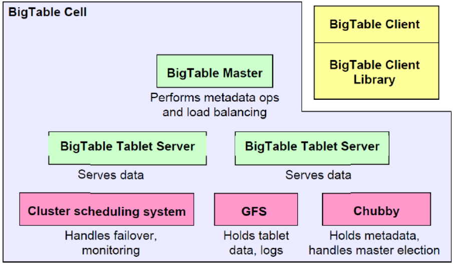
Client Library

- 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.
- Store log and data files.

Chubby Lock Service

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

Table Serving

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 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

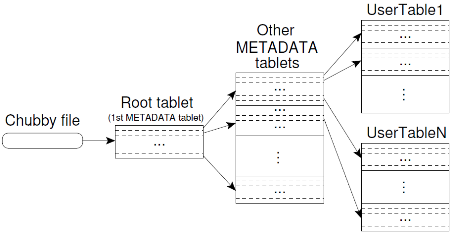
- 1 tablet $\rightarrow$ 1 tablet server.
- Master uses Chubby to keep tracks of live tablet serves and unassigned tablets.
  - When a tablet server starts, it creates and acquires an exclusive lock in Chubby.
- Master detects the status of the lock of each tablet server by checking periodically.

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.

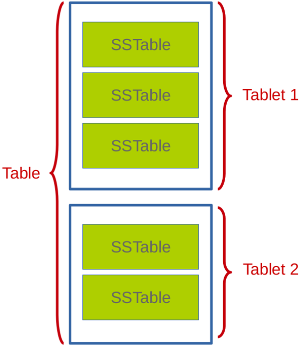
Finding a Tablet

- Three-level hierarchy.
- 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.



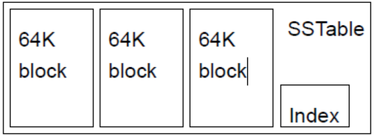
SSTable (1/2)

- SSTable file format used internally to store Bigtable data.
- Immutable, sorted file of key-value pairs.
- Each SSTable is stored in a GFS file.



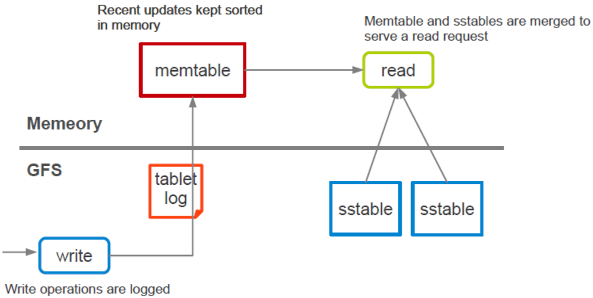
SSTable (2/2)

- Chunks of data plus a block index.
  - A block index is used to locate blocks.
  - The index is loaded into memory when the SSTable is opened.



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.

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.

Compaction

- Minor compaction
  - Convert the memtable into an SSTable.

Compaction

- Minor compaction
  - Convert the memtable into an SSTable.
- Merging compaction
  - Reads the contents of a few SSTables and the memtable, and writes out a new SSTable.

Compaction

- Minor compaction
  - Convert the memtable into an SSTable.
- Merging compaction
  - 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.

BigTable vs. HBase

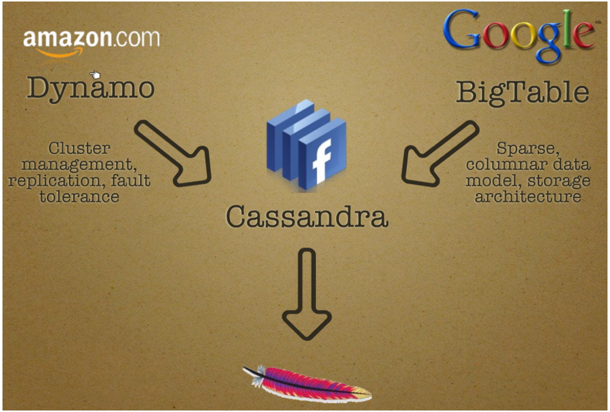
BigTable	HBase
GFS	HDFS
Tablet Server	Region Server
SSTable	StoreFile
Memtable	MemStore
Chubby	ZooKeeper

HBase Example

```
# Create the table "test", with the column family "cf"
create 'test', 'cf'

# Use describe to get the description of the "test" table
describe 'test'

# Put data in the "test" table
put 'test', 'row1', 'cf:a', 'value1'
put 'test', 'row2', 'cf:b', 'value2'
put 'test', 'row3', 'cf:c', 'value3'

# Scan the table for all data at once
scan 'test'

# To get a single row of data at a time, use the get command
get 'test', 'row1'
```

Cassandra

Cassandra


amazon.com
Dynamo
Cluster management, replication, fault tolerance
Google
BigTable
Sparse, columnar data model, storage architecture
Cassandra

From Dynamo

- Symmetric P2P architecture
- Gossip based discovery and error detection
- Distributed key-value store: partitioning and topology discovery
- Eventual consistency

From BigTable

- Sparse Column oriented sparse array
- SSTable disk storage
  - Append-only commit log
  - Memtable (buffering and sorting)
  - Immutable sstable files
  - Compaction

Cassandra Example

```
# Create a keyspace called "test"
# (a keyspace is similar to a database in the RDBMS)
create keyspace test
with replication = {'class': 'SimpleStrategy', 'replication_factor': 1};

# Print the list of keyspaces
describe keyspaces;

# Navigate to the "test" keyspace
use test

# Create the "words" table in the "test" keyspace
create table words (word text, count int, primary key (word));

# Insert a row
insert into words(word, count) values('hello', 5);

# Look at the table
select * from words;
```

Summary

Summary

- NoSQL data models: key-value, column-oriented, document-oriented, graph-based
- Sharding and consistent hashing
- ACID vs. BASE
- CAP (Consistency vs. Availability)

Summary

- Dynamo: key/value storage: put and get
- Data partitioning: consistent hashing
- Replication: several nodes, preference list
- Data versioning: vector clock, resolve conflict at read time by the application
- Membership management: join/leave by admin, gossip-based to update the nodes' views, ping to detect failure

Summary

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

References

- G. DeCandia et al., Dynamo: amazon's highly available key-value store, ACM SIGOPS operating systems review. Vol. 41. No. 6. ACM, 2007.
- 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.

Questions?