Cassandra - A Decentralized Structured Storage System

Presented by : Walid Budgaga

CS 655 – Advanced Topics in Distributed Systems

Computer Science Department

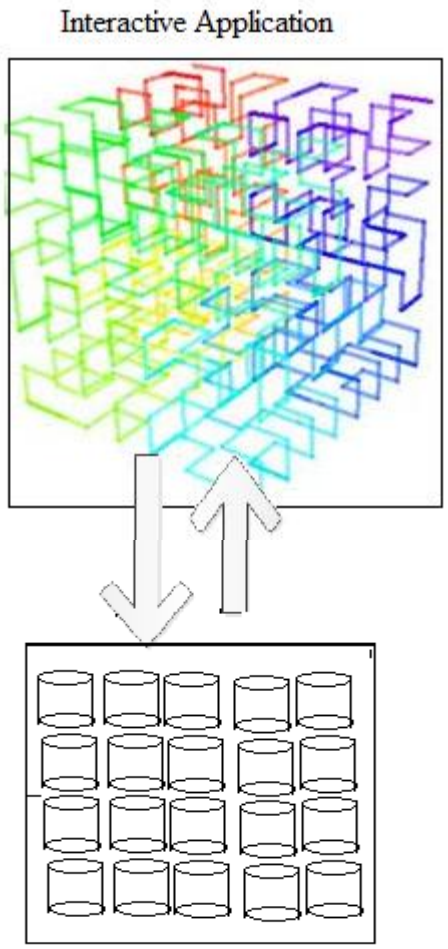
Colorado State University

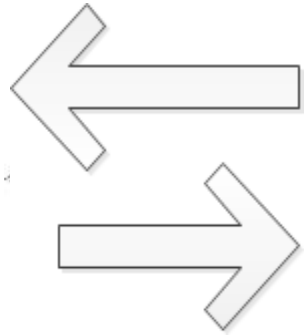
Outline

- Problem
- Solution Approaches
- Comparison
- Conclusion

Problem

Problem





- Increase the data => add DBs
- Increase the complexity
- Expensive
- Need expertise
- Deal with failed DBs
- Upgrade and repair DBs without downtime

Problem Description

Growing the global Internet use leads to having applications with millions of users.

The applications must:

- Support unpredictable number of concurrent users
  - Overnight, The number can grow from zero to a million users
- Efficiently manage big data that must be distributed across several nodes
  - 2012- Amazon sold 27 million items (306 items/s)
  - 2013- Facebook has 1.26 billion users
  - 2013- Flickr has 87 million users and stores 8 billion of phones
- Be reliable and always available

Problem Description

- **Incredible increasing amount of**
  - Data
  - Users
  - Commodity computers
- **Is driving the need for storage systems that**
  - Targets specific usage pattern
  - Easy to scale

Why not use RDBMs?

- Does not scale well
- Designed to run on one server and not on cluster of nodes
- Availability issue in case of Partitioning
- No support for different data models
- No need for RDBMs' full functionality
- Cost money

NoSQL Databases

The described problem is motivating more organizations to move to NoSQL databases that from beginning have been built to be

- Distributed
- Scale out
- Reliable
- Fault tolerance

NoSQL

The appeared NoSQL databases have Common Characteristic

- Schema-less
- Non-relational
- Ability to run on large cluster
- Horizontally scalable
- Distributed

NoSQL Databases

NoSQL databases use

- Partitioning
  - To exploit storage capabilities
  - To Balance load
- Replication
  - To increase performance
  - To provide high availability
  - To recover from failure
- Running on large cluster
  - To scale out

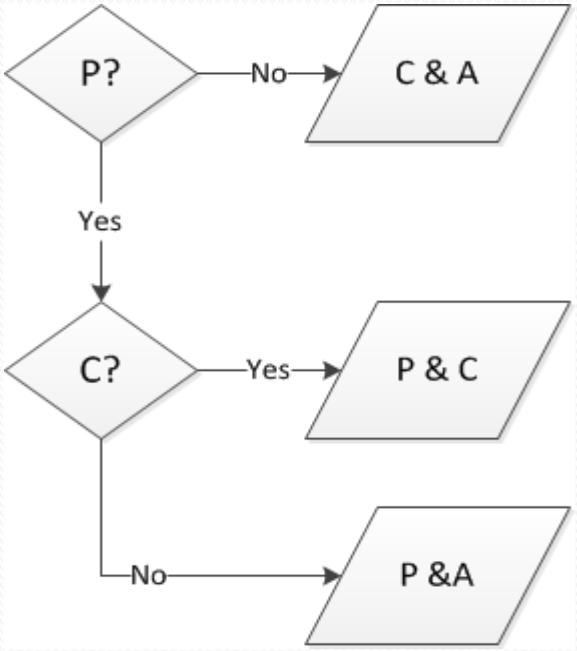
NoSQL Databases

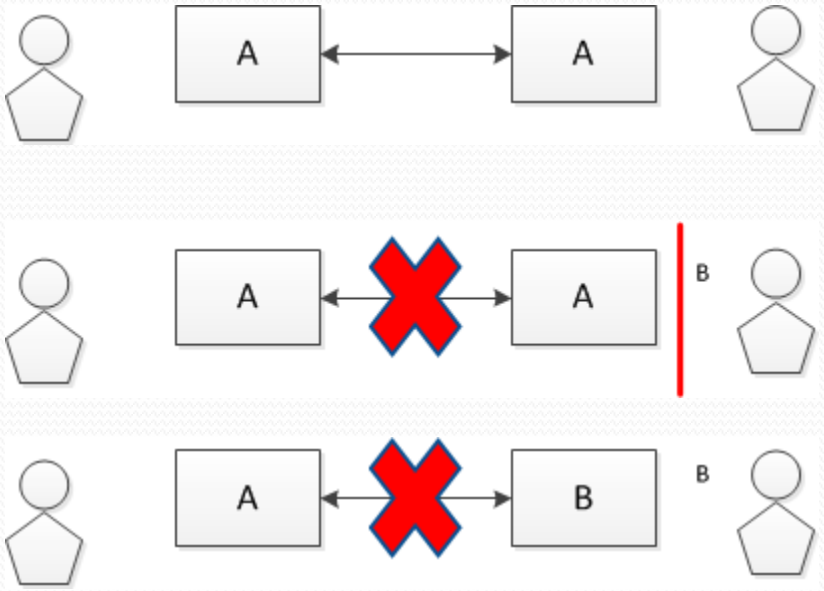
Consistency Issue

Concurrent users and data replicas distributed across different nodes

Could cause consistency issue.

CAP Theorem





NoSQL Databases

Consistency Issue

- For partition-tolerance, choice can be made based on domain business
  - Strict consistency
  - Eventual consistency

- Tradeoff consistency versus response time

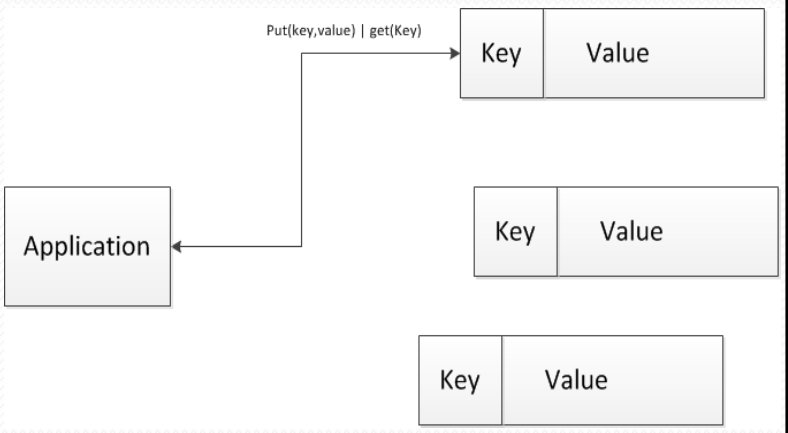
NoSQL databases

Based on the data models the NoSQL databases support, they can be classified into four groups:

- **Key-Value data model**
  - Ex: Dynamo, DynamoDB, Redis
- **Column family data model**
  - Ex: Bigtable, Cassandra
- **Graph data model**
  - Ex: neo4j
- **Document data model**
  - Ex: MonogoDB, CouchDB

NoSQL databases

Key-Value data model

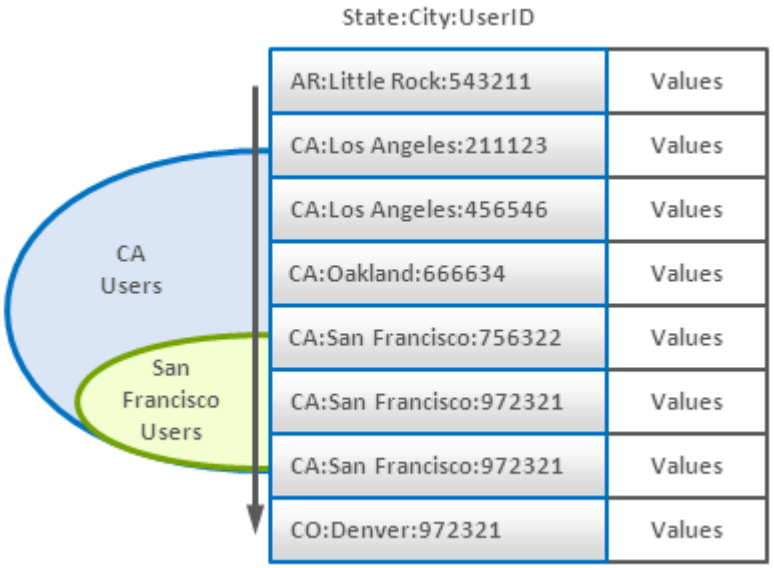


- Simple and very powerful model
- The other models relying on it
- Data values identified by keys
- System dose not care about the internal structure of the data
- Key is used to store and retrieve an object
- Poor aggregation capabilities
- Value modeling done at application level

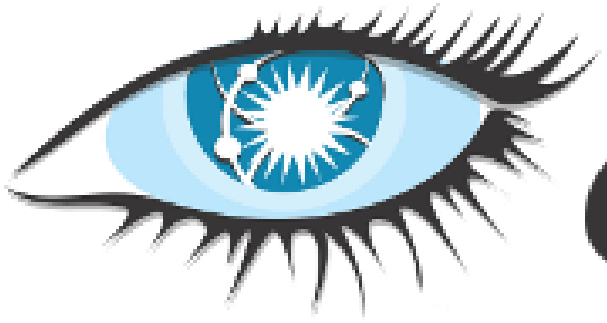
NoSQL databases

Column family data model

- Improvement of key-value model
- Organizing columns into families
  - Locality groups
- Providing different level data acces
  - Ex: Bittable uses
    - <column families, columns, timestamp>
- Support aggregation operations
- Value can be anything



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



Cassandra

Cassandra

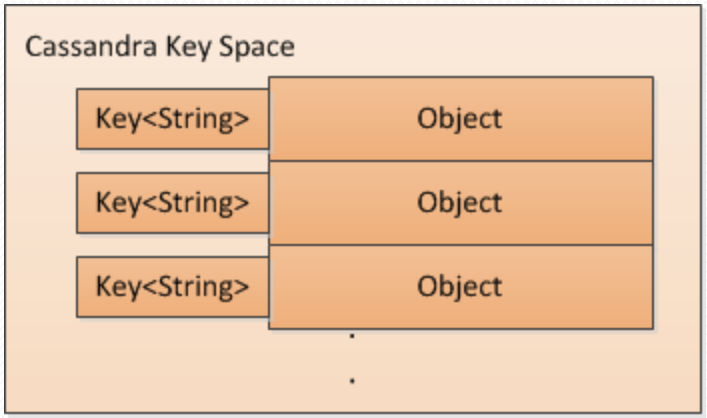
- Implemented by Facebook to solve the Inbox Search problem
- Open source
- Cassandra = Dynamo(design) + Bigtable (data model)
- P2P
- O(1) routing
- Supporting flexible structured data model
- Used by more 100 companies
  - Including Facebook, Netflix, eBay, and Twitter
- The largest known cluster has over 300 TB data in over 400 nodes

Data Model

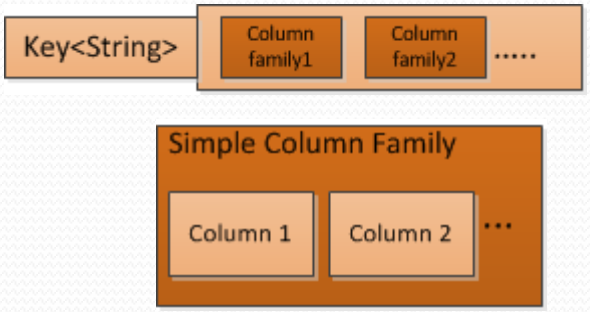
Cassandra

Data Model

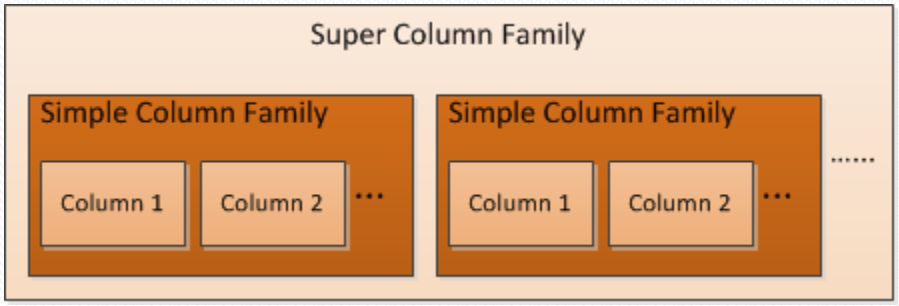
- Column-Oriented Data Model
  - Same model of Bigtable
- Row <Value>



- Column Family
  - Simple
  - Super

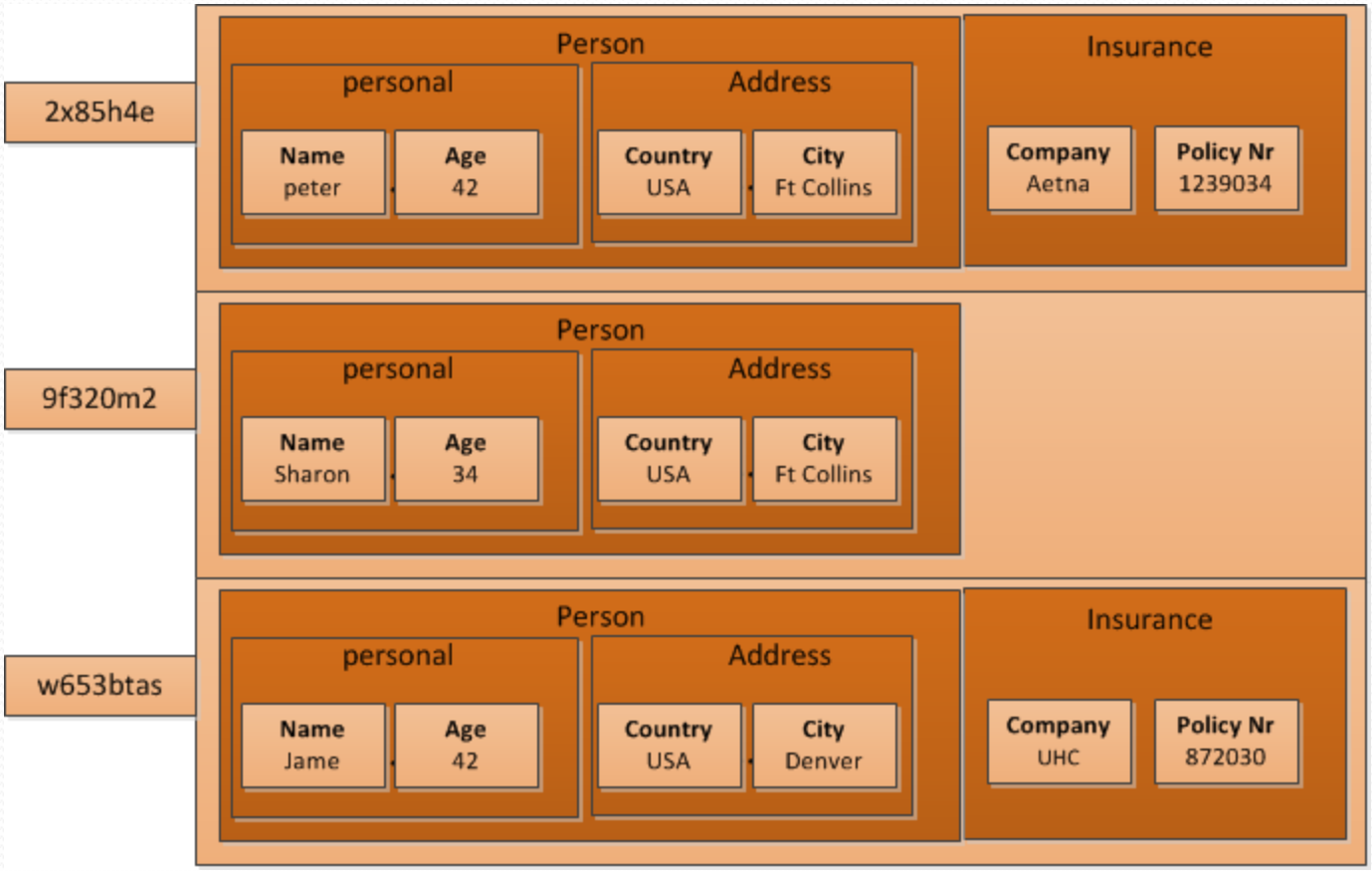


- Columns
  - Name, Value, Timestamp
  - Sorted by name or time



Cassandra

Data Model



Data Partition

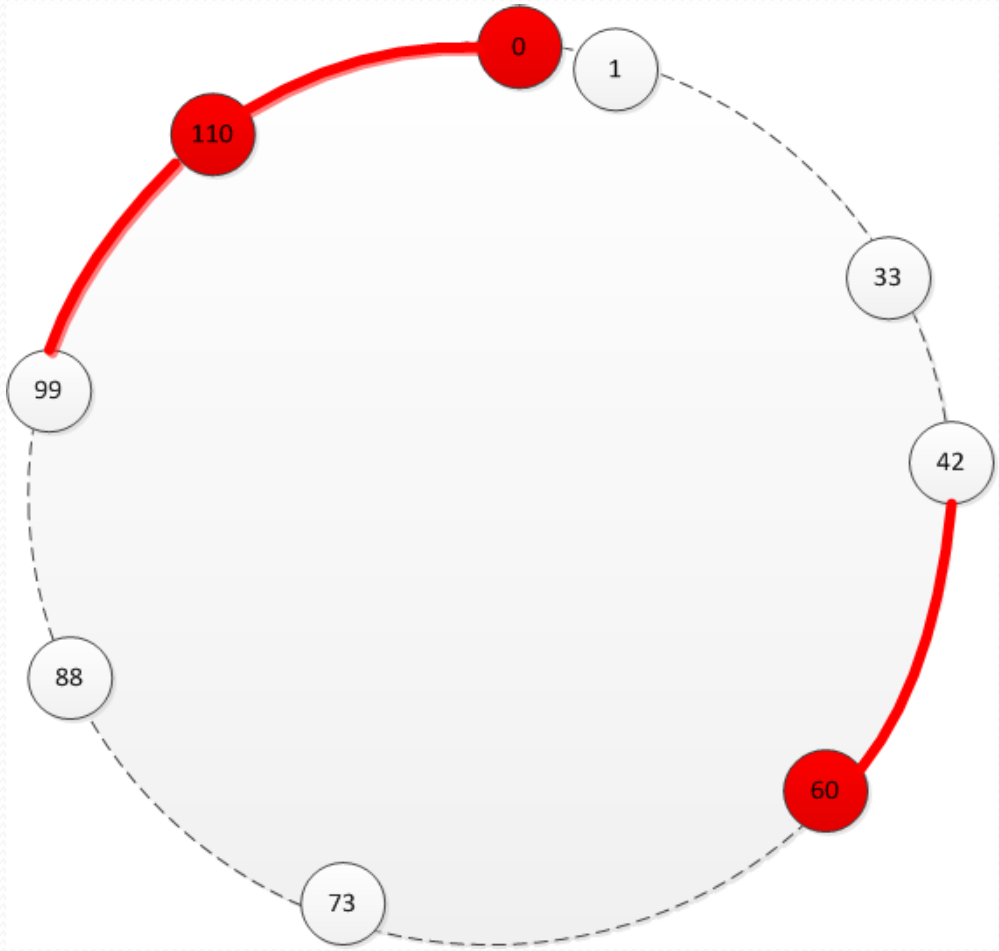
Data Partition(1)

The same concept of Dynamo to distribute data across nodes

- Organizing nodes in logical ring
- Using consistent hashing to determine data locations on the ring
  - MD5 produces hashed identifier in 2^{128} key space
- Each node is assigned a random key (token) between 0 and 2^{128} - 1
  - Placing it in random location on the ring

Data Partition (2)

- Each node is responsible for the keys fall
  - Between itself and its predecessor
- Data rows stored in nodes
  - Based on their hashed keys
  - The responsible node called coordinator
- Unbalance?
  - Move nodes



Data Partition (3)

Comparing to other systems

- Decentralized
  - Cassandra and Dynamo
    - Using consisting hashing
- Centralized
  - Bigtable
    - One master assigns tablets to tablet servers
  - GFS
    - One master assigns chunks to chunk servers

Replication

Replication (1)

- High availability, durability, and performance is achieved by
  - Replicating the data across data centers
- Each data item replicated at N nodes
  - Configurable replication factor N
- When a node joins the system, it receives data items and replicas
- The Coordinator is the responsible for replicating its data
- Replication can be configured to performed across on many data centers

Replication (2)

Replication Strategies

The first replica is always stored at coordinator, but the remainder will be placed according to the chosen strategy.

- **Rack Unaware**
  - Replicating at N-1 successive nodes
- **Rack Aware**
  - Placing one replica in another data center
  - Placing the remaining N-2 across different racks in the same data center of the coordinator
- **Datacenter Aware**
  - Placing M replicas in another data center
  - Placing the remaining N-(M+1) across different racks in the same data center of the coordinator

Replication (3)

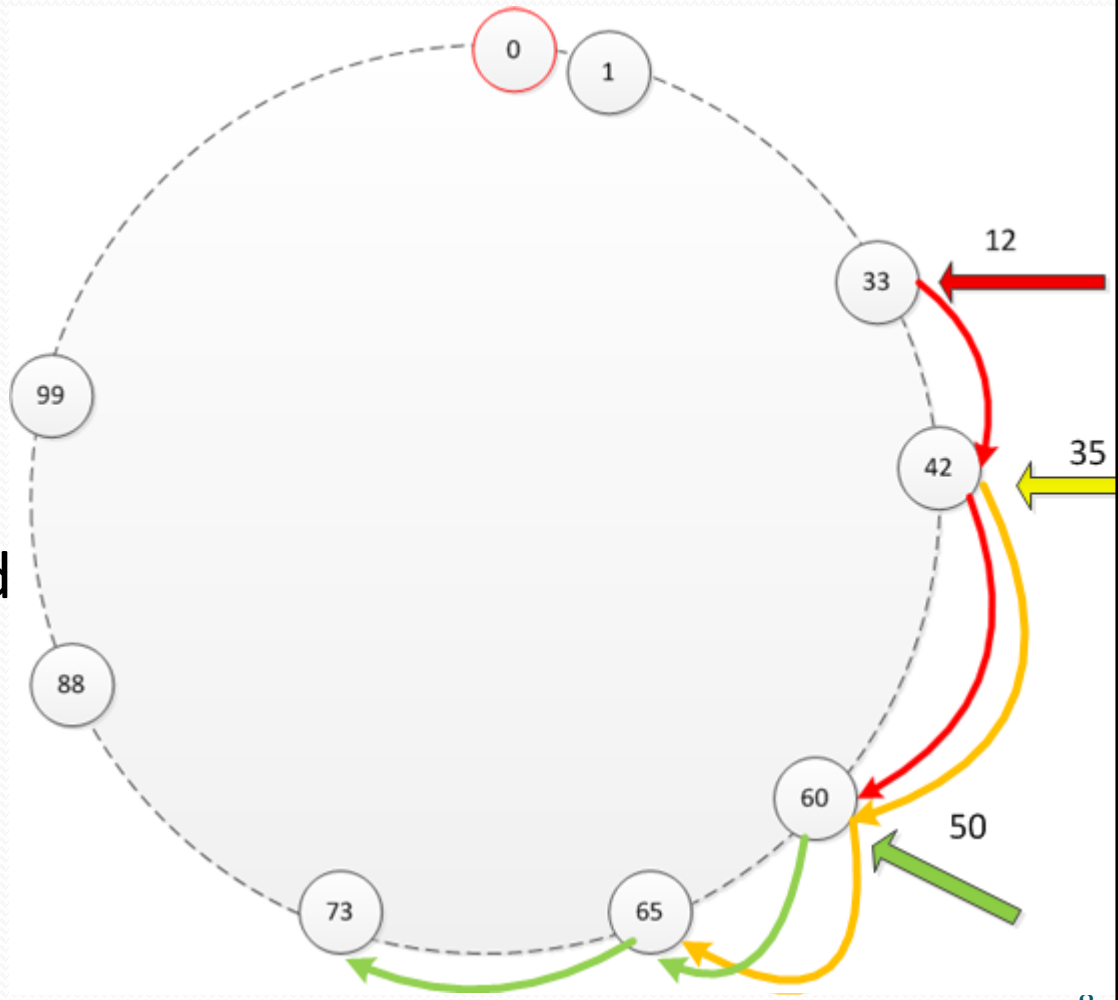
Rack Unaware

- Default
- For single data center
- Example:
  - N=3

- If node fails, data of

N nodes must be replicated

- failing node?
  - Data transfer
  - Locally effect

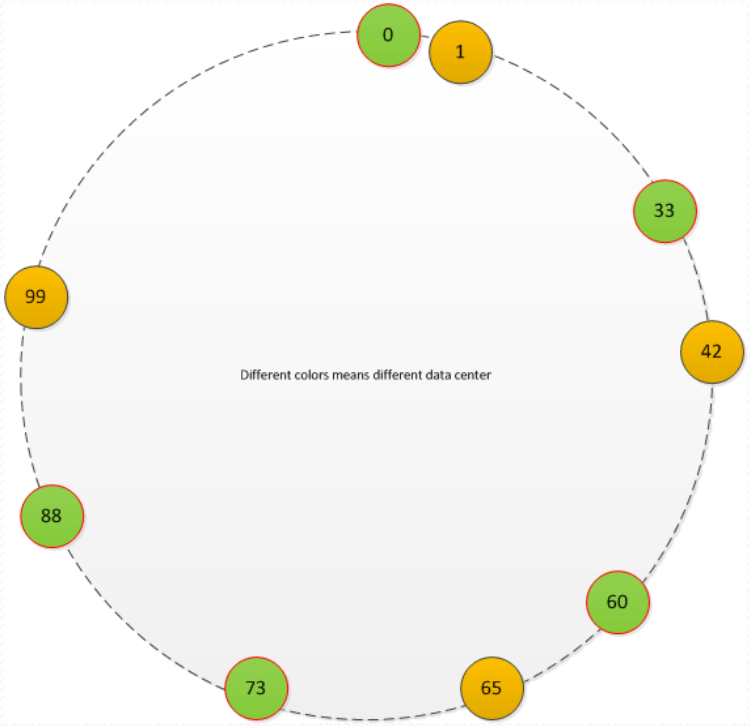


Replication (4)

Rack Aware & Datacenter Aware

Provides higher availability and adds extra latency

- To keep the effect locally
  - Keep neighbors from different DCs
  - Use 1'st node along the ring
    - That belongs in another data center



Replication (5)

Comparing to other systems

- Cassandra and Dynamo use the same replication technique
  - Both replicate rows data
- Pnuts is geographic replication
  - Tablets containing rows are replicated
  - One replica per record is chosen as master
  - Mainly used to provide quick response
- Bigtable relies on GFS to replicate data
  - Tablets containing column families are replicated
  - The main replica is managed by one tablet server

Membership
&
Failure Detection

Membership & Failure Detection

Membership

- Joined nodes can take
  - InitialToken specified in the configuration file or
  - Token so that highly loaded node will be alleviated?
- Gossip protocol is used so that each node sends
  - State information about itself and about other nodes they know
  - To three other nodes every second
- Every node knows about other nodes in the system

Membership & Failure Detection

Failure Detection

Cassandra uses Φ Accrual Failure Detector (AFD) algorithm to detect whether a node is dead or not

- Traditional failure detectors implements heartbeats
- AFD finds a suspicion level for each node
  - Between the extremes of dead and alive
- AFD computes per-node threshold considering network performance, workload, or other parameters
- Lightly loaded nodes can move to alleviate heavily loaded nodes

Failure detection

Comparing to other systems

- Bigtable
  - Joined servers register themselves with chubby service
  - Master can detect failing servers by checking the servers folder on chubby service.
  - Master reassign tablets to other servers to balance load

- GFS
  - Uses heartbeats to detect failing nodes
  - Joined servers must contact the master

Consistency

Consistency(1)

Client specifies consistency level for read & write operations

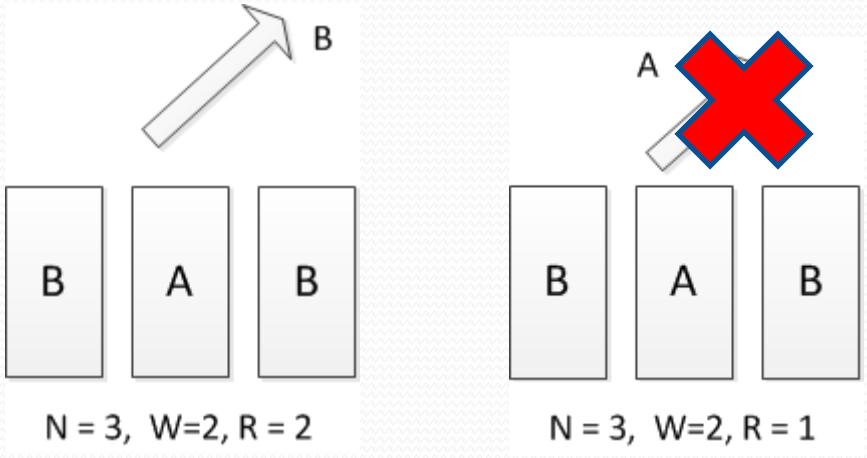
Write	
Level	Description
Zero	Immediately return
Any	Minimum one Replica or hint
One	One Replica
Quorum	$\frac{N}{2} + 1$ Replicas
All	All Replicas

Read	
Level	Description
One	One Replica
Quorum	$\frac{N}{2} + 1$ Replicas
All	All Replicas

Consistency(2)

Consistency

- **Quorum Level**
  - **Choose R and W so that**
    - **R + W > N** (overlap prevents read-write conflict)
    - **W > N/2** (prevents write-write conflicts)



Consistency (3)

Comparing to other systems

- PNUTS
  - One replica exists in each region
  - One replica is always chosen as a master
  - Master applies updates
    - To all replicas per record
    - In the same order but might be not at the same time
  - Eventual consistency
    - Immediately applying to local replica
    - Lazy applying to other replicas
      - Cause race condition

Consistency (4)

Comparing to other systems

- GFS
  - One replica server is chosen by the master as a primary
  - Primary replica server applies updates to other replicas
  - Applied updates increase the version number for each replica
    - Version number is used to detect stale replicas

Evaluation (1)

High Performance

Results of operations done on 50 GB Data

- **Inbox Search**

Latency Stat	Search Interactions	Term Search
Min	7.69 ms	7.78 ms
Median	15.69 ms	18.27 ms
Max	26.13 ms	44.41 ms

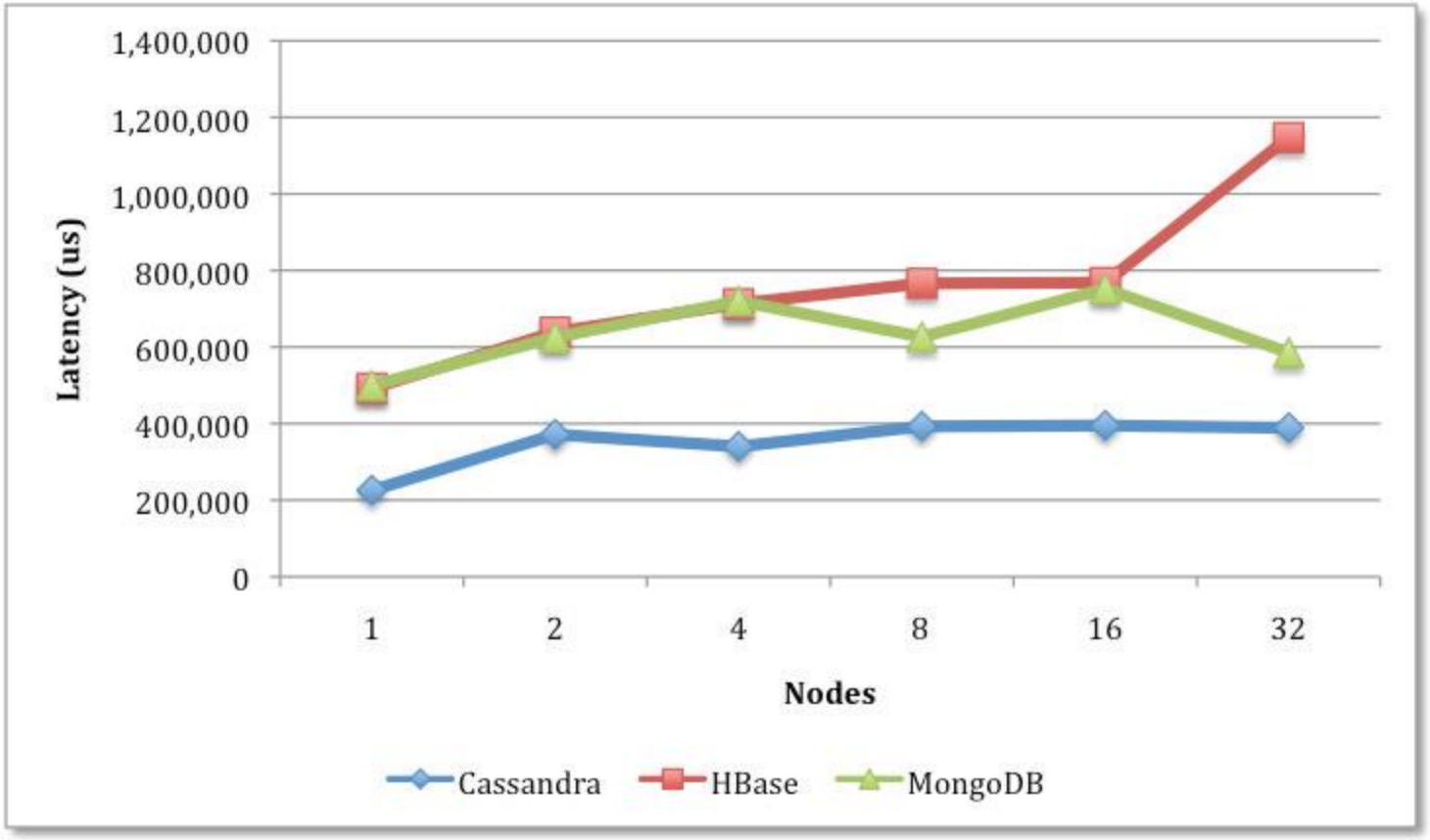
- **MySQL VS Cassandra**

	Writes Average	Reads Average
MySQL	~300 ms	~350 ms
Cassandra	0.12 ms	15 ms

Source: Slides of the authors of the paper

Evaluation (2)

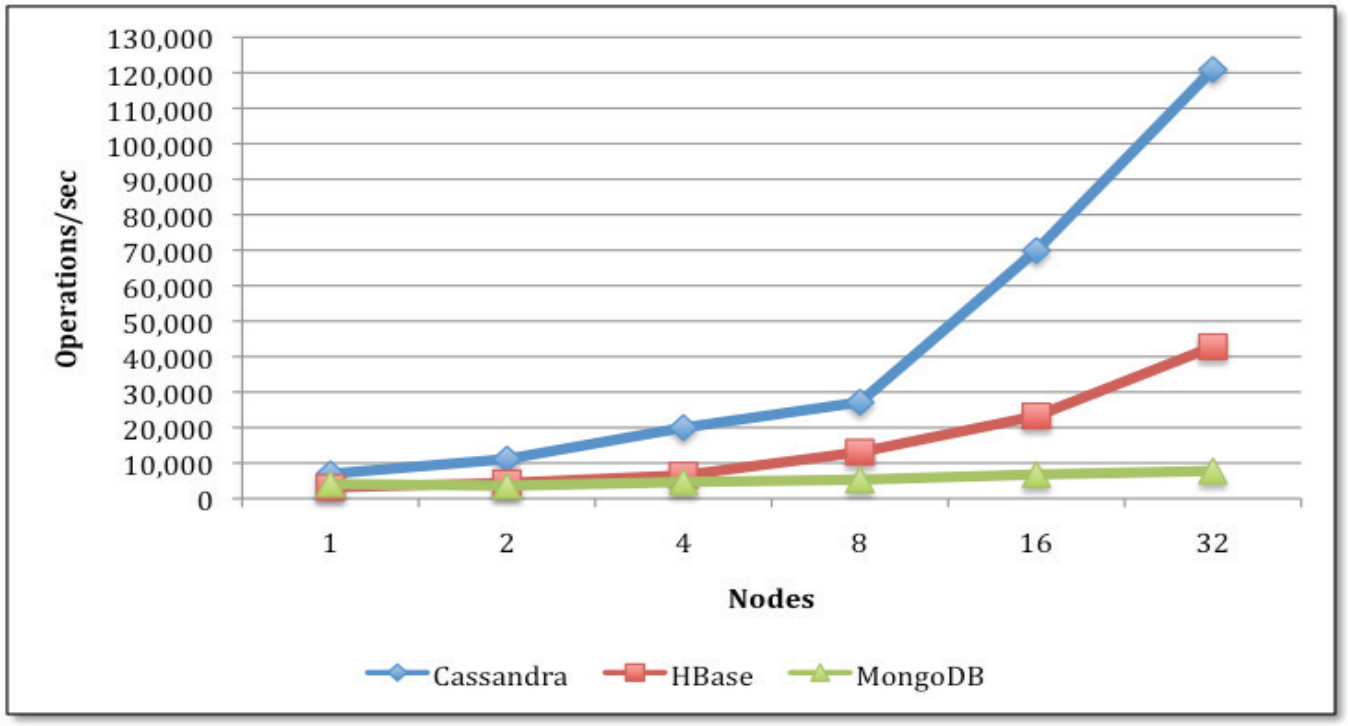
Latency



Source: Benchmarking Top NoSQL Databases White Paper BY DATASTAX CORPORATION FEBRUARY 2013

Evaluation (3)

Throughput



Source: Benchmarking Top NoSQL Databases White Paper BY DATASTAX CORPORATION FEBRUARY 2013

Dynamo

Dynamo

- High available
  - Data partitioned and replicated
  - Write operation is always available
- Scalable
- Decentralized
  - Data partitioning used consistent hashing
- Consistency alleviated by versioning
  - Quorum technique for replicas consistency
  - Decentralized replica synchronization
  - Eventual consistency

Dynamo

- Membership and failure handling
  - Gossip protocol
- No authentication required
  - Trusted environment
- Service Level Agreement (SLA)
- Dynamo targets application that
  - Requires primary key to access data
  - Deals with small values(< 1 MB)

Data Model

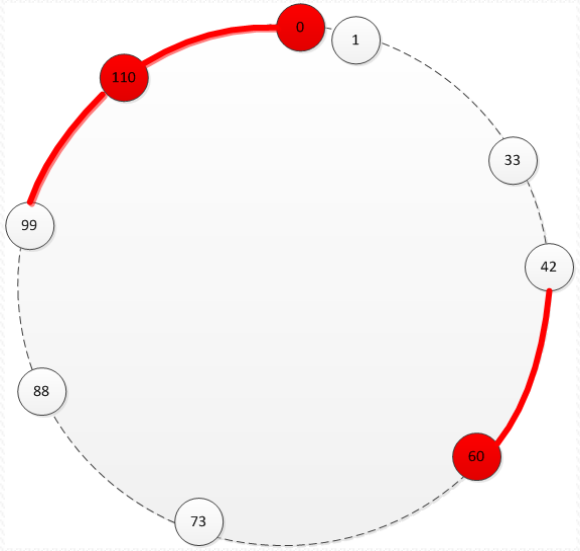
Data model

- Using unstructured key-value data model
- The value can be anything
- Allowing putting and getting value using the key
- Does not support aggregations

Data Partition

Partitioning(1)

- Evenly partitioning the data across the nodes
  - Using MD5 produces hashed identifier in 2^{128} key space
- Each node is assigned a random key (token)
- Data items stored in nodes according to their hashed keys
- Each node will be responsible for the keys
  - Fall in the range between it and its predecessor
  - The responsible node is called the coordinator
- Each node will have full information about others
  - Using gossip protocol



Partitioning(2)

- Assigning nodes to random position leads
  - Uneven data distribution
  - Uneven load balance
- Splitting powerful physical node to virtual nodes
  - To avoid unevenly data distribution
  - To exploit heterogeneity in the system
- Number of virtual nodes depends on node capacity
- If node maintaining virtual nodes fails
  - Its data will be dispersed over available nodes
- To avoid unbalance in the ring
  - Virtual nodes are used

Replication

Replication (1)

- Dynamo achieves high availability and durability
  - By replicating each piece of data N -1 times
- Coordinator replicates data on N – 1 successor nodes
- Coordinator for each key stores preference list
  - To know where to replicate data

Replication (2)

- Preference list for particular key containing replicas nodes for that key
- The preference list maintains information of more than N nodes
  - To cope with failing nodes
  - To use some of them for hinted handoff
- Some positions in the ring are not included in the list
  - To avoid having of virtual nodes of the same physical node in preference list

Versions

Versions (1)

- Dynamo aims for availability
  - Achieved by performing asynchronies updates to the replicas
- Old versions could be used to provide high availability
  - In cases of failing of the coordinator and
  - Existing not up to date replicas
- To avoid having multiple versions of the same object
  - Dynamo make these versions as immutable

Versions (2)

- Concurrent updates with failure
  - Conflicting versions can be resulted
- Vector clock is used to capture causality between Conflicting versions
  - New version always wins
- If the causality is considered as concurrent
  - Automatic reconciliation is not possible
  - The clients have to reconcile it
  - Merging different versions of shipping cart

Consistency

Consistency

- Dynamo uses Quorum protocol to provide different levels of consistency on
  - Read and write operations
  - By choosing values of R and W
- Small R and large W means
  - Reduce the response time for read operation
  - Increasing the latency to do the write operation

Failure Handling

Failure Handling

- If coordinator node fails
  - Replica of data coordinated by the failed node will be sent to the successor (N+1)'th node
  - Once the coordinator comes back
    - The (N+1)'th node will send the data into it
    - The data will be deleted from the (N+1)'th node
- In the case of failing of data center,
  - Replicas of each object are stored on different data centers