Introduction to NoSql Databases
&
Amazon Dynamo

Slide set 9

Distributed Systems

Graduate Level

K. N. Toosi Institute of Technology
Dr. H. Khanmirza
h.khanmirza@kntu.ac.ir

Database

- 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

Database History

- 1960s
  - Navigational data model (hierarchical model)
  - Proposed by Bachman
  - To speedup operations on disk
  - IDS (Integrated Data Store) and CODASYL (data model & language)

```
get department with name='Sales’
get first employee in set department-employees
until end-of-set do {
        get next employee in set department-employees process employee
}
```

Database History

- 1970s
  - Relational Database Management System (RDBMS) by Codd
  - Logical data is disconnected from physical information storage
- 1980s
  - Object Database
  - Information represented by objects
- 2000s
  - NoSQL: BASE principles instead of ACID
  - NewSQL: scalable performance with BASE + ACID

RDBMS

- Relational Database Management System
  - The dominant technology for storing structured data in web and business applications

- SQL
  - A language for data retrieval from RDMBS
  - Rich language
  - Easy to use and integrate
  - Rich toolset
  - Many vendors

RDBMS ACID Properties

- RDMBS promises 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
  - Consistent State defined by consistency model

RDBMS ACID Properties

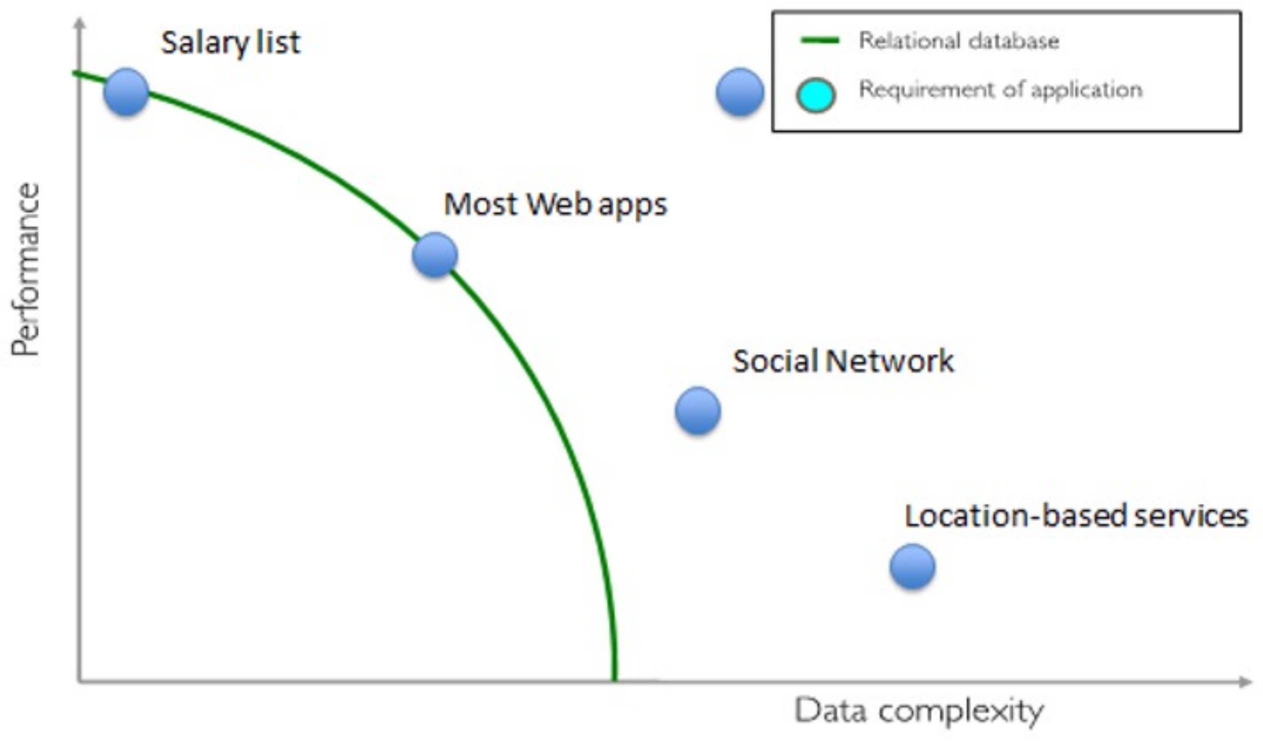
- Isolation
  - Transactions can not see uncommitted changes in the database
- Durability
  - Changes are written to a disk before a database commits a transaction so that committed data cannot be lost through a power failure

RDBMS vs. New Requirements

- Internet has new requirements
  - Internet scale data size
  - High read and write rates
  - Frequent schema changes
  - Joins are expensive

- RDBMS was not designed to be distributed

RDBMS vs. New Requirements



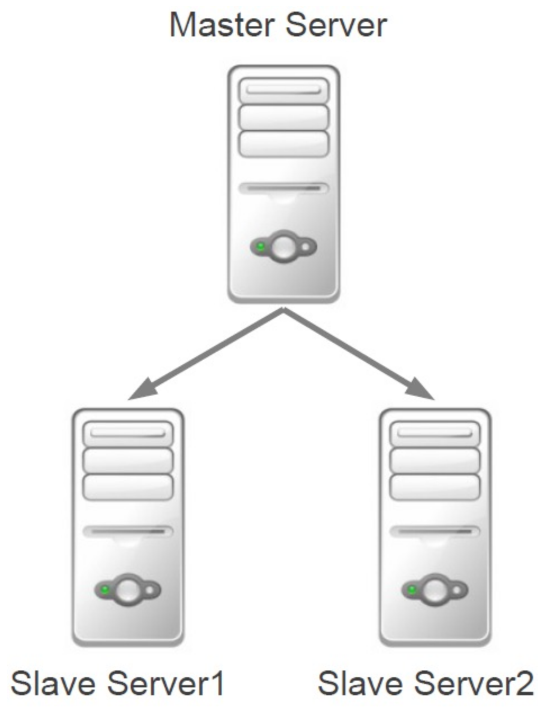
RDBMS vs. New Requirements

- Solutions
  - Replication
  - Sharding

Solutions

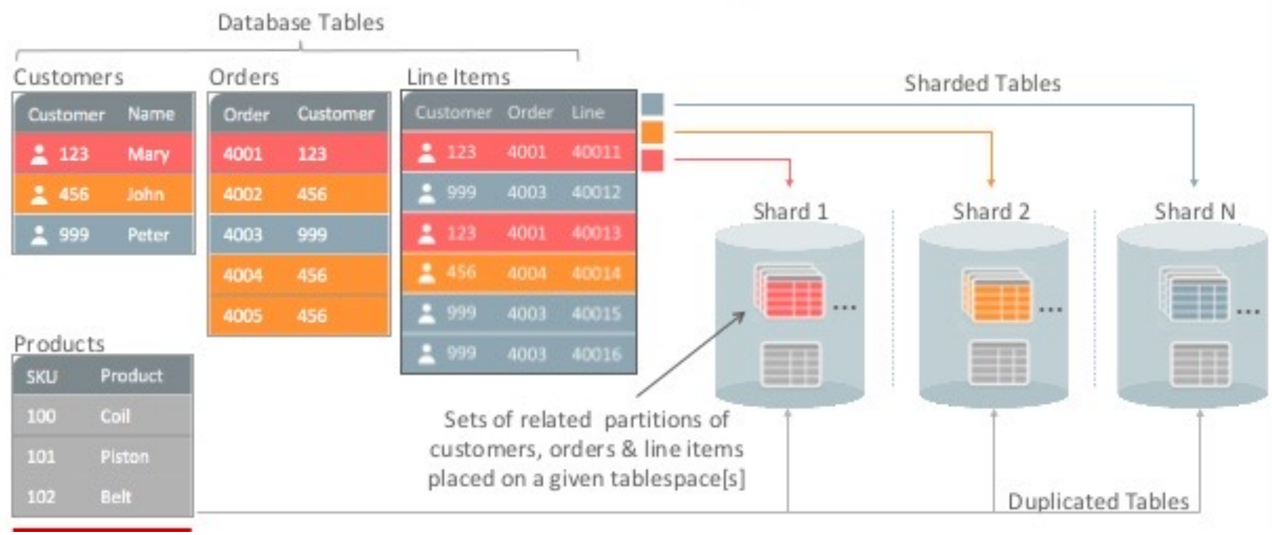
- Replication
  - Master-slave architecture
  - Scales read operations

  - Expensive
    - Hardware
    - Product cost
    - Maintenance



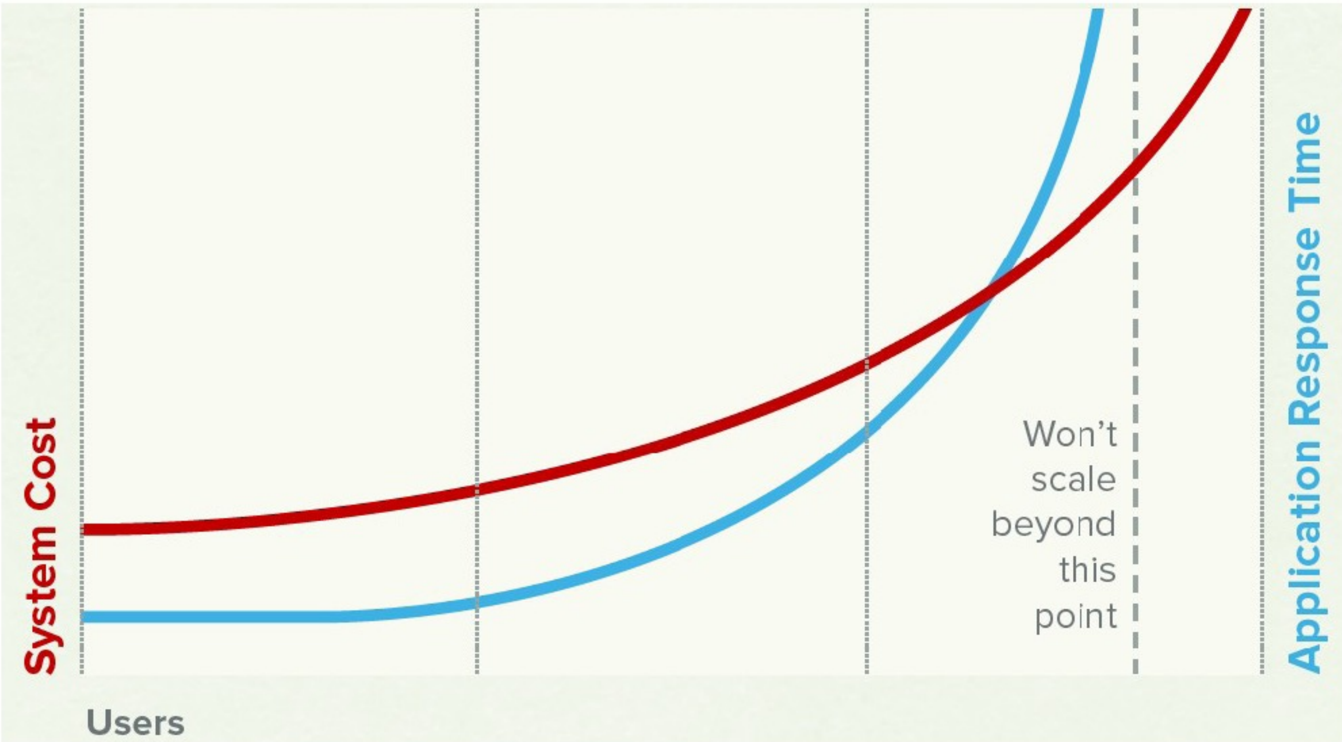
Solutions

- Sharding
  - Divide data base across several machines
  - Scales read and write operations
  - RDBMS Cannot execute transactions across shards (why? CAP theorem?)



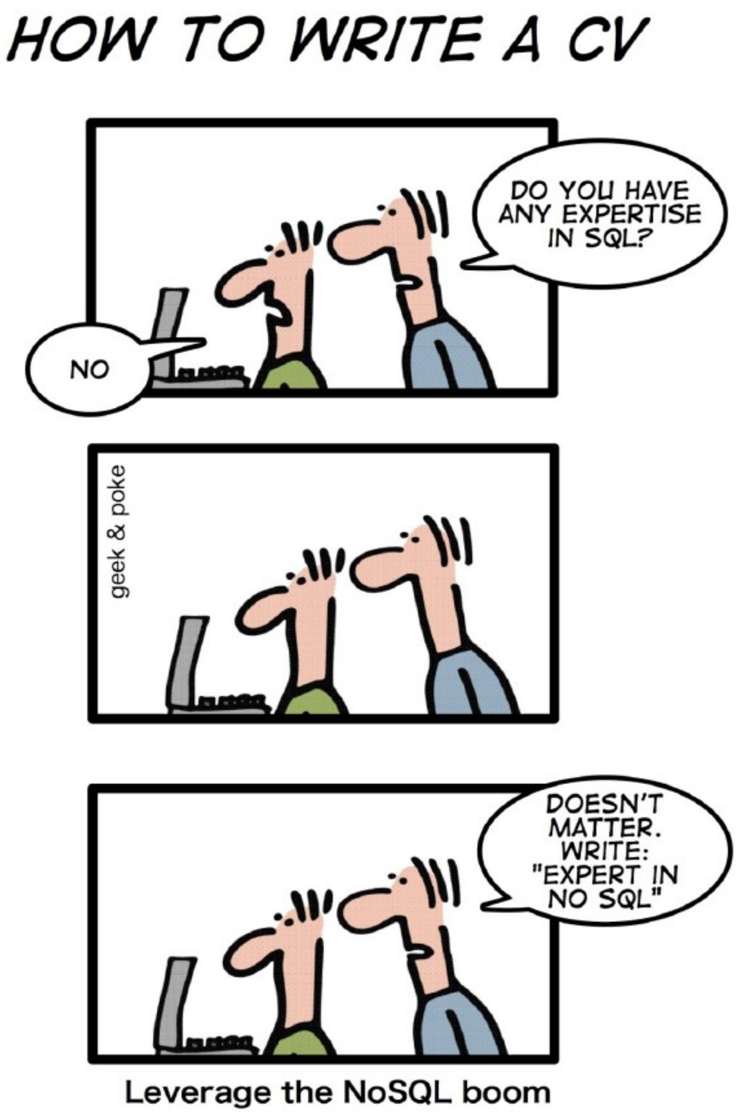
https://image.slidesharecdn.com/oraclesharding18csangam18-181212124352/95/oracle-sharding-18c-technical-overview-15-638.jpg?cb=1544619088

Solutions



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

Not Only SQL


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

NoSQL

- Avoidance of unneeded complexity
- High throughput
- Horizontal scalability and running on commodity hardware
- Compromising reliability for better performance

NoSQL

- Emphasizes on
  - Easy and frequent changes to DB
  - Fast development
  - Large data volumes (e.g. Google)
  - Most of consideration in application layer
    - Transactions
    - No explicit data types
  - Schema-less

NoSQL

- This word was first used in 1998 by Carlo Strozzi to name his relational database that did not expose the standard SQL interface

- The term was picked up again in 2009 when a Last.fm developer, Johan Oskarsson, wanted to organize an event to discuss open source distributed databases

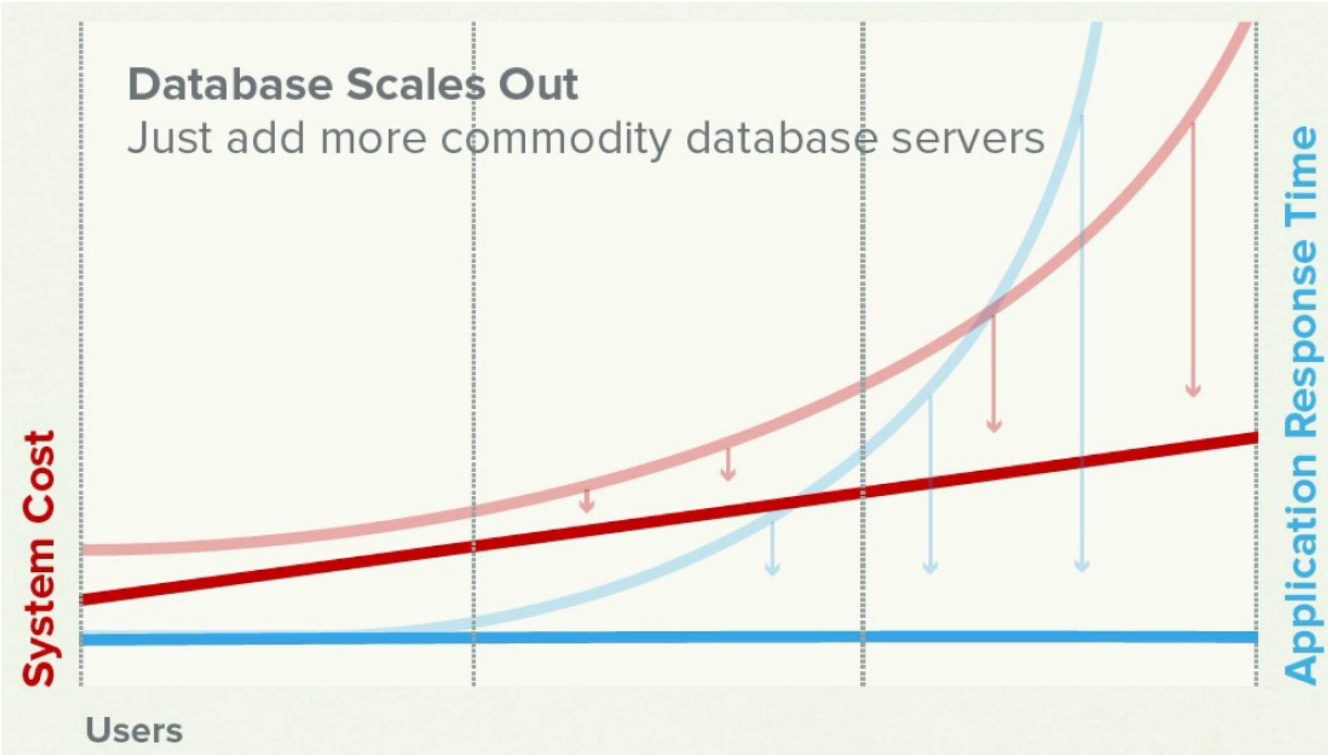
NoSQL

- The name attempted to label the emergence of a growing number of non-relational, distributed data stores that often did not attempt to provide ACID

- In practice, they actually use <u>SQL-like</u> languages

http://www.newsinsurances.co.uk

NoSQL



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

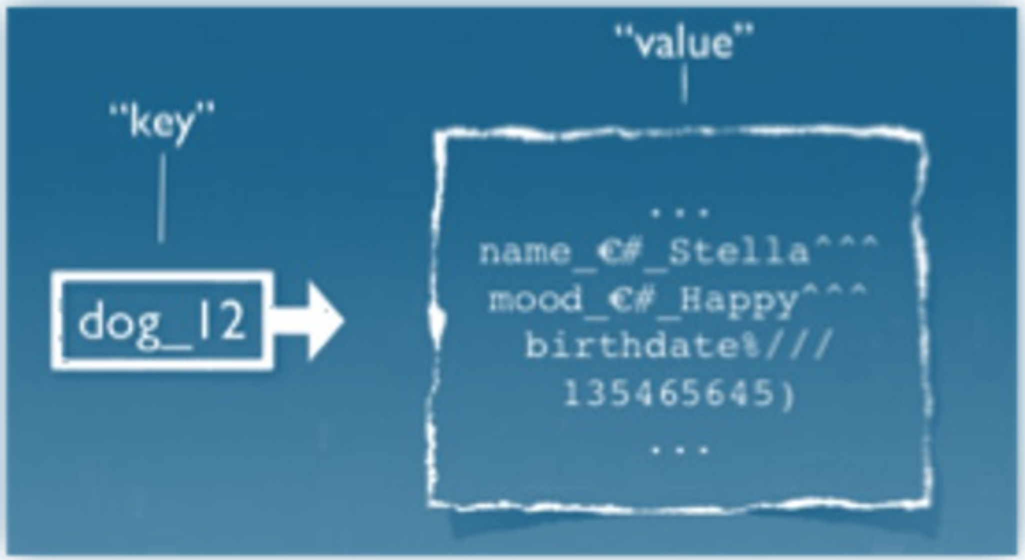
NoSQL

- Data Models
  - Key-Value
  - Document
  - Column-Oriented
  - Graph

Key-Value Data Model

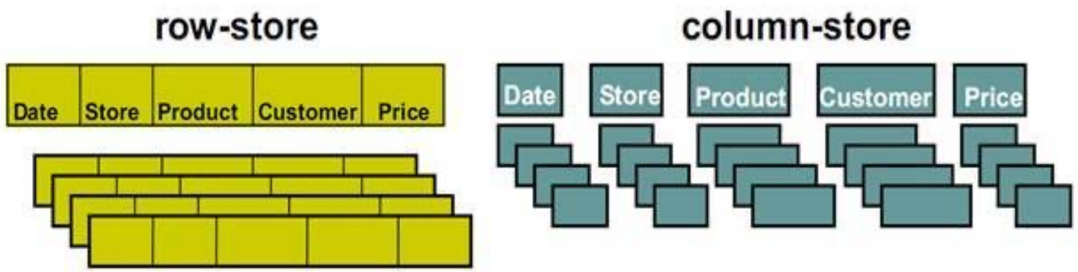
- Collection of key/value pairs
- Ordered Key-Value: processing over key ranges

- Like a big distributed hash table
- Popular products: Redis, 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
- Popular products: BigTable, Hbase, Cassandra



Column-oriented Data Model

RowId	EmpId	Lastname	Firstname	Salary
001	10	Smith	Joe	60000
002	12	Jones	Mary	80000
003	11	Johnson	Cathy	94000
004	22	Jones	Bob	55000

Stored as

```
001:10,Smith,Joe,60000;
002:12,Jones,Mary,80000;
003:11,Johnson,Cathy,94000;
004:22,Jones,Bob,55000;
```

- Row-based model is optimized for efficiently reading of rows from disk
  - Read a contact information
  - Read a product info
- How about finding products with prices between 1000$ and 10000$?

Column-oriented Data Model

- This type is optimized for operating on a set of data
  - Finding products with prices between 1000$ and 10000$?

- In row-based data model, all rows must be read
  - We may improve by index-es on columns
  - Indices add complexity

NoSQL

RowId	EmpId	Lastname	Firstname	Salary
001	10	Smith	Joe	60000
002	12	Jones	Mary	80000
003	11	Johnson	Cathy	94000
004	22	Jones	Bob	55000

Stored as

```
10:001,12:002,11:003,22:004;
Smith:001,Jones:002,Johnson:003,Jones:004;
Joe:001,Mary:002,Cathy:003,Bob:004;
60000:001,80000:002,94000:003,55000:004;
```

- Column-oriented model is optimized for efficiently reading of columns

Column-oriented Data Model

- Facebook statistics
  - MySQL > 50 GB Data
  - Writes Average : ~300 ms
  - Reads Average : ~350 ms

  - Rewritten with Cassandra > 50 GB Data
  - Writes Average : 0.12 ms
  - Reads Average : 15 ms

Column-oriented Data Model

- It is mostly suited for OLAP applications

- OLAP: Online Analytical Processing
  - Complex queries for business intelligence or reporting

- OLTP: Online Transaction Processing
  - Emphasize on availability, speed, concurrency and recoverability
  - Short-lived transactions
    - Touching small amounts of data per transaction
    - Use indexed lookups (No table scans)

Document-based Data Model

- Similar to a column-oriented store, but values can have complex documents, instead of fixed format
- Flexible schema using XML, YAML, JSON, and BSON
- Popular products: 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},
      ]
}
```

Document-based Data Model

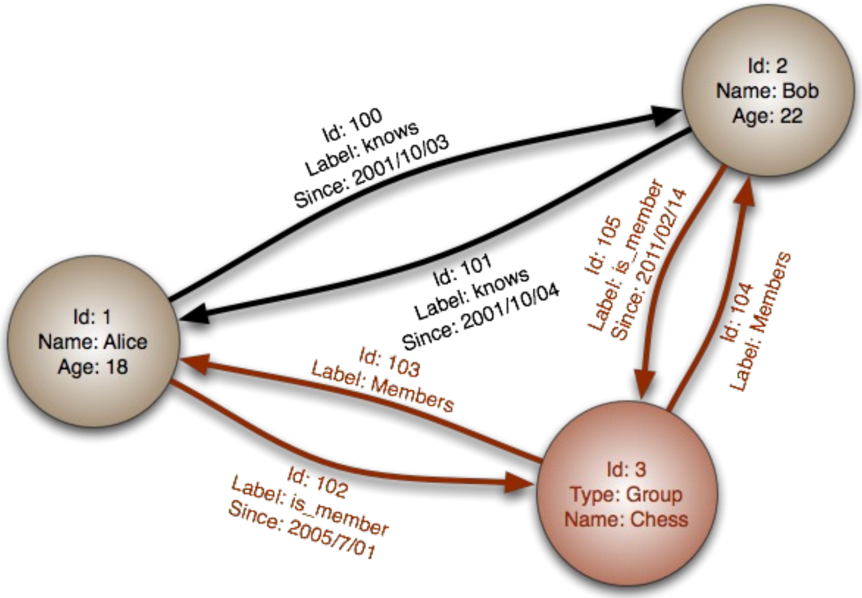
- A great choice for content management applications such as blogs and video platforms

- An intuitive for a developer to update an application as the requirements evolve
  - Only the affected documents need to be updated
  - No schema update is required
  - No database downtime is necessary to make the changes

Graph Data Model

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

- Popular products: Neo4j, InfoGrid
- Some support transaction and ACID properties



[http://en.wikipedia.org/wiki/Graph database]

NoSQL

- The large-scale applications have to be reliable: availability + redundancy
  - These properties are difficult (and theoretically impossible) to achieve with ACID properties as proved with CAP theorem

- The BASE approach forfeits the ACID properties of consistency and isolation in favor of availability and performance

- Relational vs. NoSQL ➔ Right data vs. Fast Data

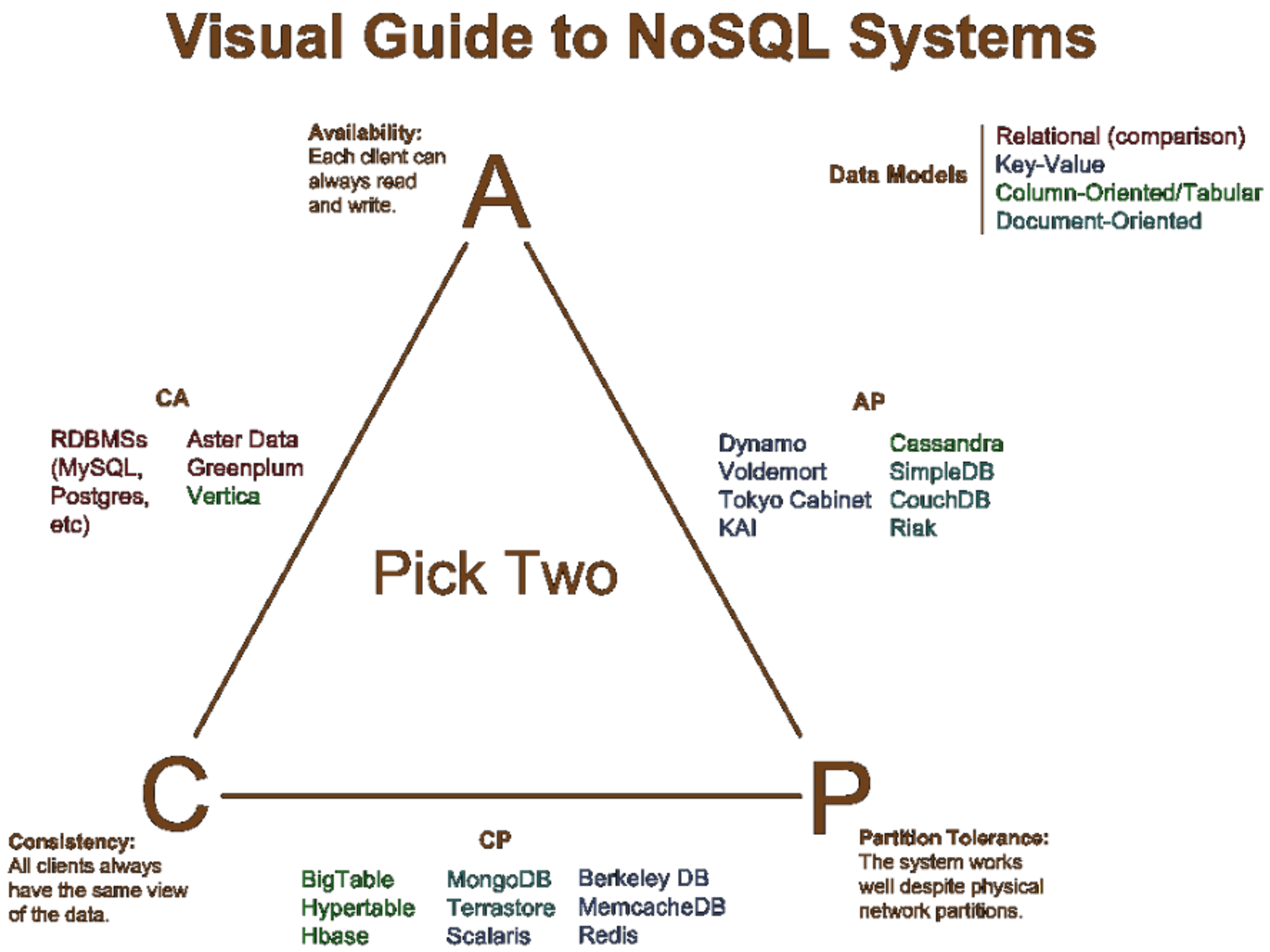
BASE Properties

- 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

BASE & CAP


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.

Amazon Dynamo

Amazon’s Highly Available Key-value Store

Dynamo

- A Distributed, scalable, highly available key/value storage system

- There are many services on Amazon’s platform that only need primary-key access to a data store.
  - Best seller lists
  - Shopping carts
  - Customer preferences
  - Session management
  - Product catalog

- Using a relational database is non-efficient and limits scale and availability

Dynamo

- Dynamo provides a simple primary-key only interface to meet the requirements of these applications

- True scalability Examples
  - The service maintains shopping cart served tens of millions requests that resulted in well over 3 million checkouts in a single day
  - The service managing session state handled hundreds of thousands of concurrently active sessions

Dynamo

- Suitable for
  - Always writable: sacrificing strong consistency for availability
  - Nodes are trusted
  - No hierarchical namespace
  - Fast response time (99.9 read/writes are done in a few hundred ms)
    - Achieved by zero-hop DHT mechanism!

Dynamo

- Design considerations
  - Conflict resolution
    - When & how
    - Mostly given over to application as it has knowledge to resolve
  - Incremental scalability
    - Servers may be joined incrementally
  - Symmetry
    - Every node should have the same set of responsibilities (role) as its peers
  - Decentralization
    - No central administration
  - Heterogeneity
    - Work load are assigned based on capacity of nodes

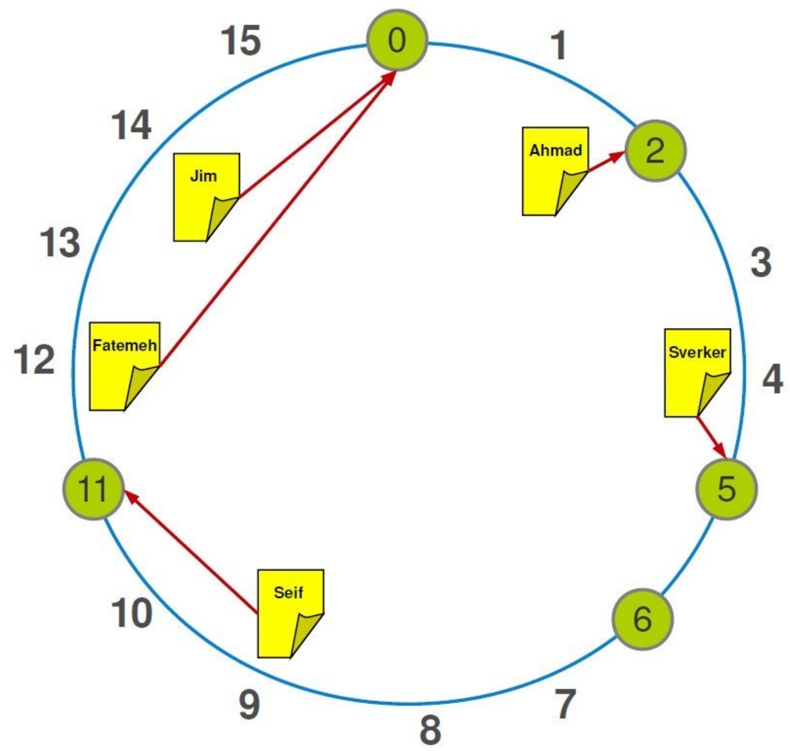
Dynamo

- Architecture
  - Data partitioning
  - Replication
  - Data versioning
  - Dynamo API
  - Failure Handling
  - Membership management

Data Partitioning

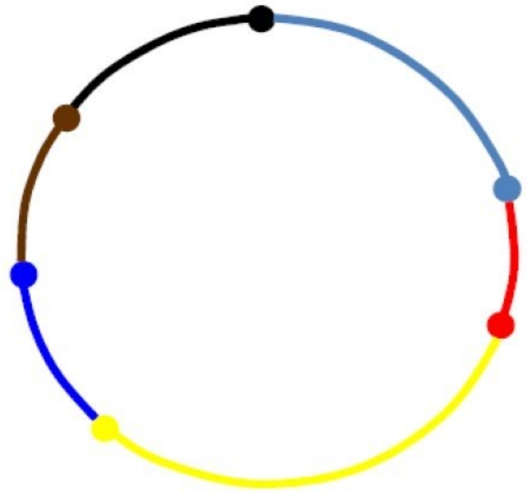
- If size of data exceeds the capacity of a single machine does Sharding (horizontal partitioning)

- Consistent hashing is one form of automatic sharding



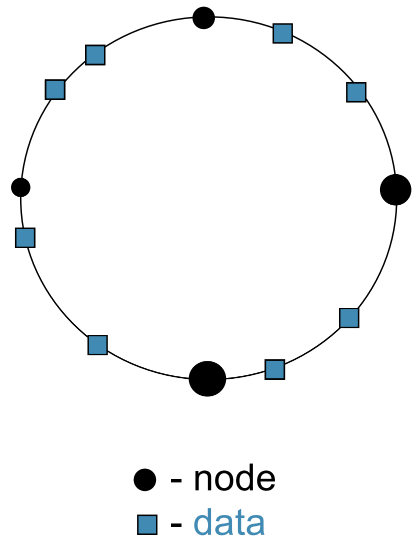
Data Partitioning

- Node identifiers may not be balanced



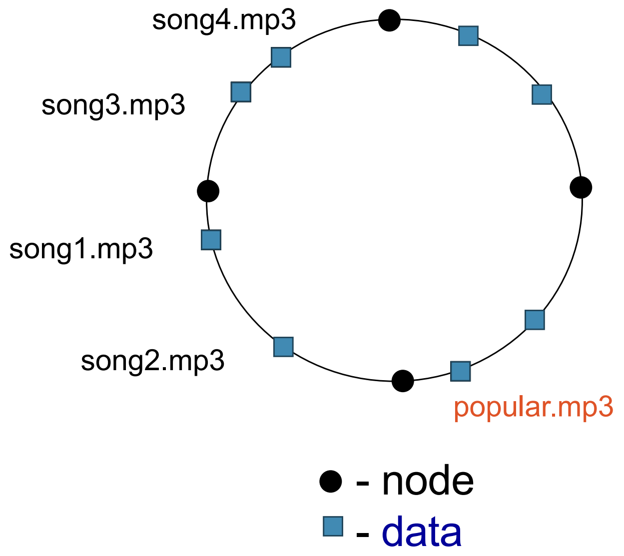
Data Partitioning

- Data identifiers may not be balanced



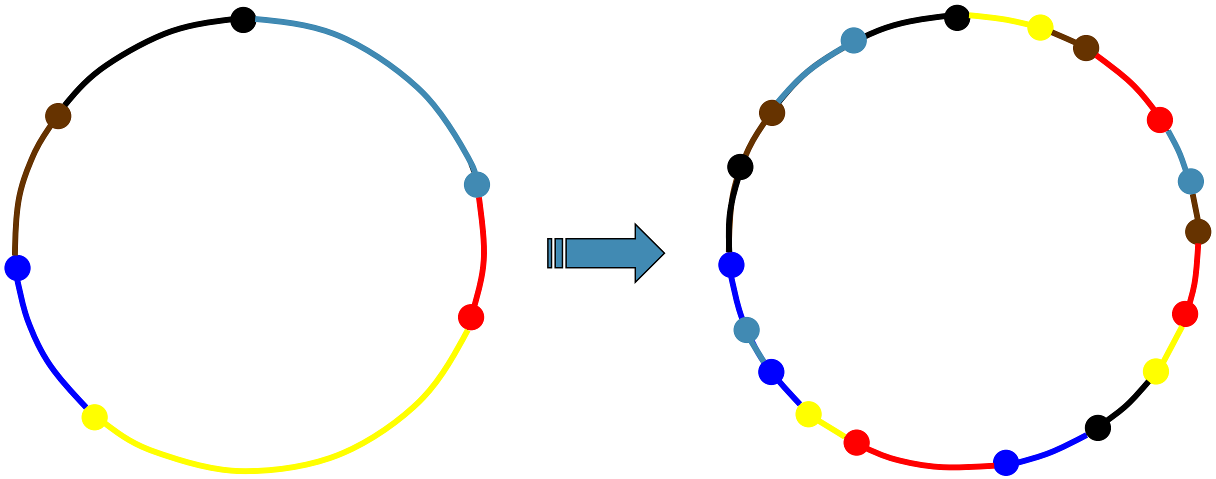
Data Partitioning

- Hot spots



Data Partitioning

- Each physical node picks multiple random identifiers
  - Each identifier represents a virtual node
- For homogeneous, all nodes run log N virtual servers
- For heterogeneous, nodes run clogN virtual servers



Replication

- Data is stored on the coordinator node
  - Coordinator = main responsible node in the ring
- Data is replicated on N-1 clockwise successor physical nodes
  - Skipping duplicate physical nodes
- List of nodes having a piece of data called preference list

Data Versioning

- 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

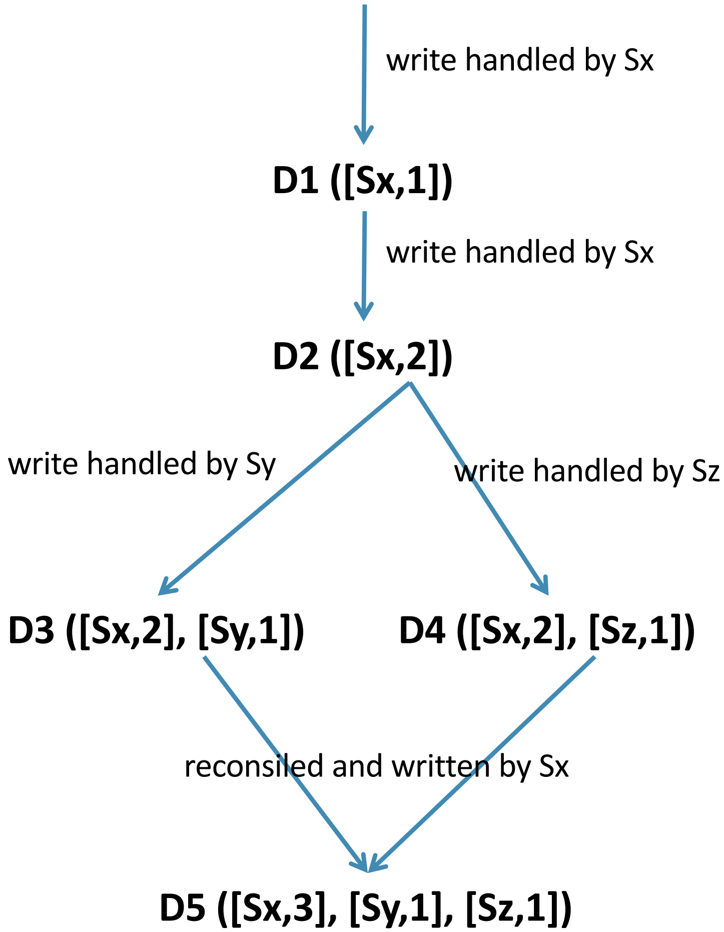
- Version branching can happen due to node/network failures

- Use vector clocks for capturing causality, in the form of (node, counter)

- If causal: older version can be forgotten
- If concurrent: conflict exists, requires reconciliation

Data Versioning

- Client C1 writes new object in Sx Node
- C1 updates the object via Sx.
- C1 updates the object via Sy.
- C2 reads D2 and updates the object via Sz.
  - D1 and D2 are overwritten by the new data and can be garbage collected
- C3 reads D3 and D4 via Sx.
  - D3 and D4 are concurrent and need reconciliation
- The read context is a summary of the clocks of D3 and D4: [(Sx, 2), (Sy, 1), (Sz, 1)].
- Reconciliation by user in Sx



Data Versioning

- Reason
  - Node failures, data center failures, network partitions
  - Large number of concurrent writes to an item

- Occurrence
  - 99.94 % one version
  - 0.00057 % two versions
  - 0.00047 % three versions
  - 0.00009 % four versions

Dynamo Operations

- Easy 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 meta-data, e.g. version number

Dynamo Operations

- Client can send the request:
  - To the node responsible for the data (coordinator): save on latency, code on client
  - To a generic load balancer: extra hops in routing

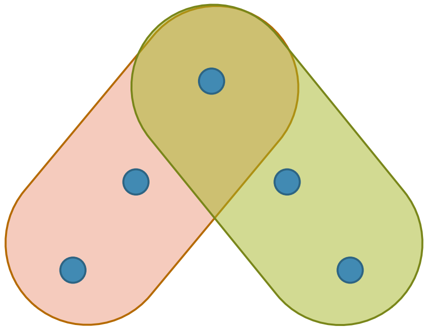
Dynamo Operations

- Dynamo uses quorum-like mechanism to execute operations

- put operation
  - Coordinator generates new vector clock and writes the new version locally
  - Sends to N highest-ranked nodes in preference list
  - Wait for response from W nodes
  - Using W=1
    - High availability for writes
    - Low durability

Dynamo Operations

- get operation
  - Coordinator requests existing versions from N top-ranked in preference list
  - Waits for response from R nodes
  - If multiple versions, return all versions that are causally unrelated
  - Divergent versions are then reconciled
  - Reconciled version written back
- Using R=1
  - High performance read engine
- Recall: in quorum systems R + W > N



R=3, W=3, N=5

Handling Failures

- What if data center failures happen?
  - Power outages, cooling failures, network failures, and natural disasters

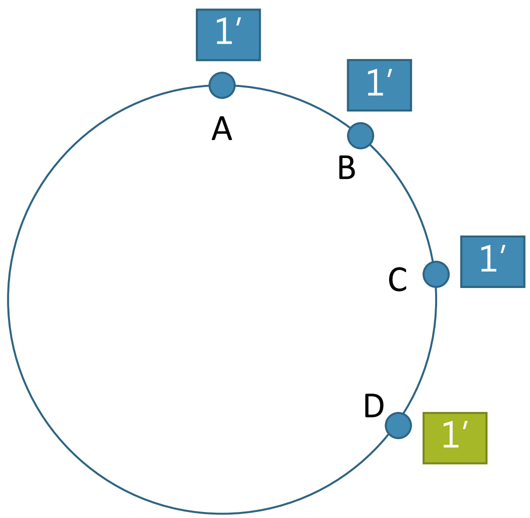
- Preference list of a key is constructed such that the storage nodes are spread across multiple data centers

Handling Failures

- Hinted Handoff
  - Sloppy Quorum
    - All read and write operations are performed on the first N healthy nodes from the preference list
    - They may not always be the first N nodes encountered while walking the consistent hashing ring

Handling Failures

- Hinted Handoff
  - What if a node temporarily down or unreachable?
  - To maintain durability of N, just replicate object in the next node
    - Hinted replicas store objects in separate database
    - Periodically scan to see if the respective node is alive, if so, transfer the object and remove it locally



Handling Permanent Failures

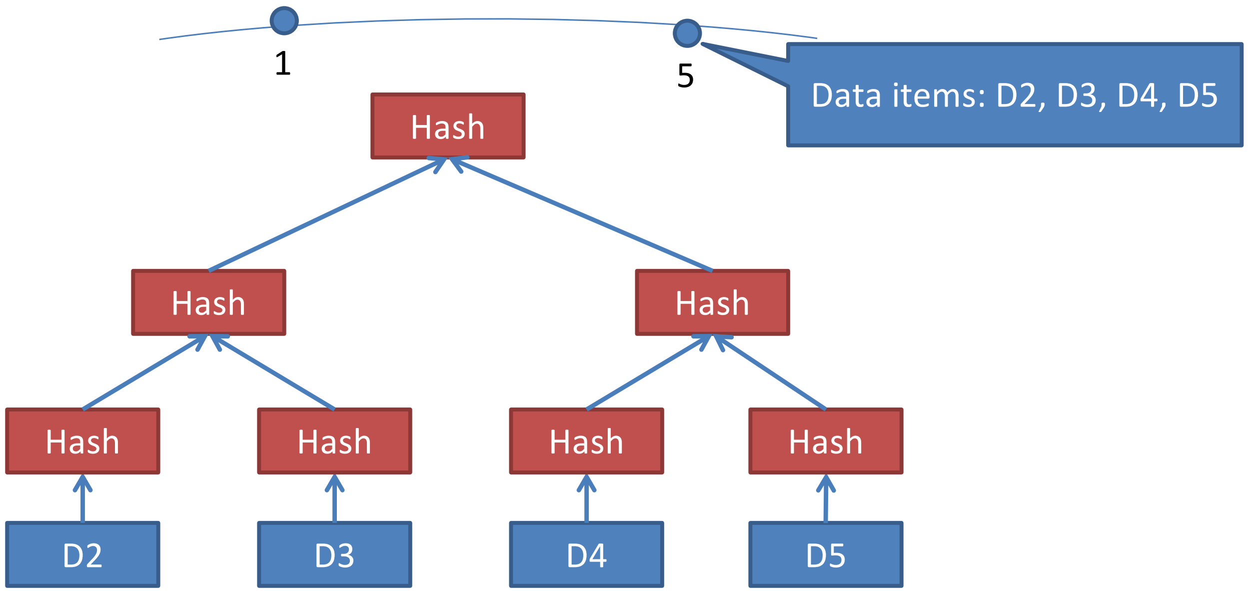
- What if hinted replica become unavailable?
- Anti-entropy protocol for replica synchronization
- Use Merkle trees for fast inconsistency detection and minimum transfer of data

Handling Permanent Failures

- A Merkle tree is a hash tree where leaves are hashes of the values of individual keys
- Parent nodes higher in the tree are hashes of their children
- Advantage:
  - Each branch of the tree can be checked independently without requiring nodes to download the entire tree

Handling Permanent Failures

- Nodes maintain Merkle tree of each key range
- Exchange root of Merkle tree to check if the key ranges are up-to-date



Membership Management

- Administrator explicitly adds and removes nodes
  - Respective keys are transferred from/to previous coordinators
- Gossiping to propagate membership changes
  - push-based model
  - Eventually consistent view

Dynamo: End Notes

- Peer-to-peer techniques have been the key enablers for building Dynamo

"… decentralized techniques can be combined to provide a single highly-available system."

References

- Wikipedia
- Slides of Dr. Payberah: https://www.slideshare.net/payberah/
- Kumar Ashwani, Introdution to NoSQL databases, http://pages.di.unipi.it/turini/Basi%20di%20Dati/Materiale%202017-18/NoSQL-slides.pptx
- Slides of Shafaat: https://www.kth.se/social/upload/519229b5f276547d7882c57f/9-dynamo-id2210-10.ppt

The End!