ISQA 8450 NoSQL and Big Data Technologies

Instructor: Dr. Martina Greiner

CAP Theorem and other NoSQL concepts

Important Concepts

- CAP Theorem
- High Availability
  - Metrics and SLA
  - Clusters
    - Distribution Model - Master-Slave, masterless (P2P)
  - Partitioning / Sharding
  - Replication
  - Load Balancing
- Consistency

Brewer's CAP Theorem

Presented by **Eric Brewer** in 2000
https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

- Highlights trade-offs in distributed data systems
- Distributed database systems can have at most two of the following three desirable properties
  - Consistency *
    - All clients (users) read always the same data
  - Availability
    - Each client can always read and write; internal communication failures shouldn't prevent updates; system remains operational
  - Partition tolerance
    - System responses to clients even if there is a communication failure between database partitions in the network
  - However, in essence, it is a trade-off between C and A

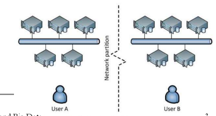


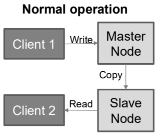
Figure 3-3. Network partition in a distributed database application
Source: "Next Generation Databases" by Harrison

* (Be careful: this is different from the C in ACID.)

Brewer's CAP Theorem

- Partition tolerance is no choice in distributed system – networks will fail (http://robertgreiner.com/2014/08/cap-theorem-revisited/, http://blog.cloudera.com/blog/2010/04/cap-confusion-problems-with-partition-tolerance/)
- In case communication channels between partitions are broken, systems designer need to choose whether to place higher importance on availability or consistency

Normal operation

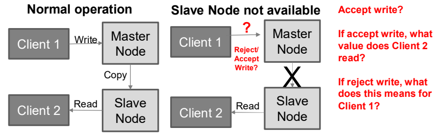


Source: Making Sense of NoSQL by McCreary and Kelly

Brewer's CAP Theorem

- Partition tolerance is no choice in distributed system – networks will fail
- In case communication channels between partitions are broken, systems designer need to choose whether to place higher importance on availability or consistency

Normal operation **Slave Node not available**

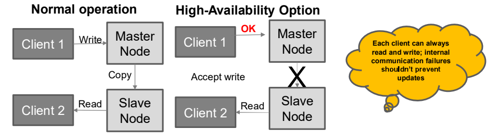


Source: Making Sense of NoSQL by McCreary and Kelly

Brewer's CAP Theorem

- Partition tolerance is no choice in distributed system – networks will fail
- In case communication channels between partitions are broken, systems designer need to choose whether to place higher importance on availability or consistency

Normal operation **High-Availability Option**

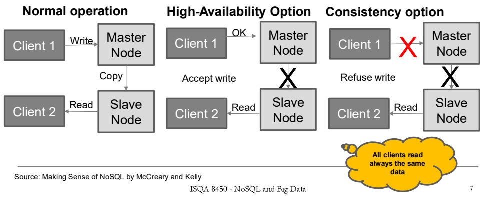


Source: Making Sense of NoSQL by McCreary and Kelly

Brewer's CAP Theorem

- Partition tolerance is no choice in distributed system – networks will fail
- In case communication channels between partitions are broken, systems designer need to choose whether to place higher importance on availability or consistency

Normal operation **High-Availability Option** **Consistency option**



Source: Making Sense of NoSQL by McCreary and Kelly

CAP Theorem

- NoSQL systems usually focus on either C or A
  - Consistency
    - MongoDB, MemchacheDB, Redis, HBase
  - Availability
    - Cassandra, CouchDB, DynamoDB, Riak
- This is only a rough categorization
- System designer may also set parameters to choose level of availability and level of consistency
- Other characteristics and trade offs are also important*:
  - Latency, fault-tolerance, simplicity of programming model, operability, transaction support, use of storage space

https://martin.kleppmann.com/2015/05/11/please-stop-calling-databases-cp-or-ap.html

BASE

- Maintaining consistency in a distributed database system with multiple copies of data distributed across the system is difficult and increases latency
  - Some applications need strong consistency, some are fine with relaxing consistency – so called eventual consistency!
- BASE
  - Basically available – availability in terms of the CAP theorem, temporarily inconsistency is okay
  - Soft-state – data may be change over time, even without input
  - Eventual consistency – without input, over time, the data will be consistent

Building High Availability Data Stores

- Measuring availability of NoSQL databases
  - Several measures available, e.g., Availability = Uptime / (Uptime + Downtime) expressed in counts of 9s

Availability %	Annual downtime
95%	18.25 days (438 hours)
99% (two nines)	3.65 days (88 hours)
99.9% (three nines)	8.76 hours
99.99% (four nines)	52.56 minutes (.88 hours)
99.999% (five nines)	5.26 minutes (.088 hours)

Today's golden standard

  - The overall availability is determined by the predicted availability of each dependent (single point-of-failure) component
    - network (99.9%) x server (99%) x power (99.9%) = overall (98.8%)
  - E.g., Amazon S3 (Simple Storage Service – key-value store)
    - Durability of 99.999999999% of objects (probability that all 3 replicas of your data fails)*)
    - Availability of 99.99%

*)"For example, if you store 10,000 objects with Amazon S3, you can on average expect to incur a loss of a single object once every 10,000,000 years." https://aws.amazon.com/s3/faqs/

Source: http://www.edgeblog.net/2007/in-search-of-five-9s/, https://aws.amazon.com/s3/details/
Making Sense of NoSQL by McCreary and Kelly

SLA

- Detailed availability targets are determined in a Service Level Agreement
  - Defines availability and response time goals for service
  - You need to know your system's points-of-failure and overall availability as well as other metrics, e.g.,
    - Client yield (probability of a request returning in a specified time)
    - Harvest (data available divided by the total data sources)
  - May include consequences of not meeting the goals
    - E.g., Amazon S3 specifies penalties (10% service credit) if your system is not up 99.9% in any given month https://aws.amazon.com/s3/sla/

Source: Making Sense of NoSQL by McCreary and Kelly

SQL Strategies for creating high-availability Services

- Clusters
- Replication
- Load balancer
- High-availability distributed filesystems

Clusters

- "Clusters are sets of connected computers that coordinate their operations" (NoSQL for mere mortals)

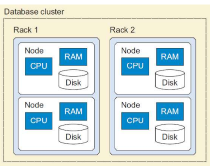


Figure 2.4 Some of the terminology used in distributed database clusters. A cluster is composed of a set of processors, called nodes, grouped together in racks. Nodes are commodity processors, each of which has its own local CPU, RAM, and disk.

Making Sense of NoSQL pg 20, 137

Shared Nothing and Shared Disk

- Shared everything (shared RAM)
  - Every database process shares RAM, CPU, disk. This is usually a single-node architecture.
- Shared disk
  - Every database process may be on different node with own RAM and CPU, but access the same disk
- Shared nothing
  - Each node has its own CPU, RAM, and Disk

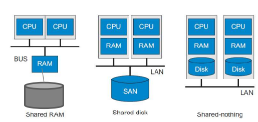


Image adapted from: "Making sense of NoSQL" McCreary and Kelly
Text: Next generation databases by Harrison

Shared Nothing Architecture

- NoSQL systems most commonly use the Shared-Nothing Architecture

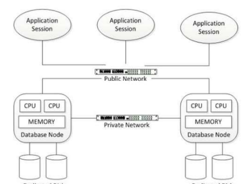
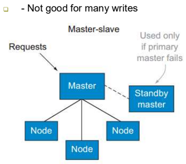


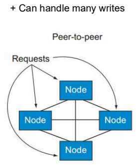
Figure 8-3. Shared-nothing database architecture

Source: Next generation databases by Harrison (in chapter 8)

Distribution models
"Who is in charge of write requests?"

- Master-slave (example of multiple node type architecture)
  - One node is in charge of writing and replicating (the master node)
  - - Single point-of-failure
  - + Simpler model (each node only communicates with master node)
  - - Not good for many writes
- Peer-to-peer (masterless architecture)
  - All nodes process requests
  - + No single point-of-failure (better for high-availability systems)
  - - Complexity/ Communication overhead / Consistency
  - + Can handle many writes

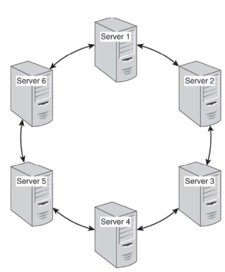




Making Sense of NoSQL pg 138

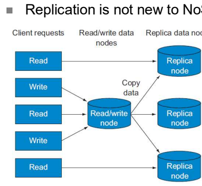
Ring Structure for Masterless Clusters

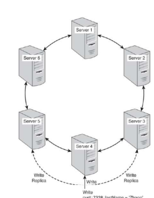
- A logical structure for organizing partitions
- Each server is linked to two adjacent servers
- Each server is responsible for managing a partition
- A partition key determines on which server the data goes



Replication

- Multiple copies of the same data are stored on different nodes in case a node becomes unavailable
  - Improves availability and read performance
- Replication is not new to NoSQL databases





Making Sense of NoSQL pg 145

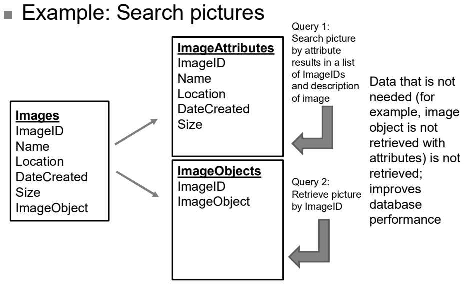
NoSQL for mere Mortals p 135

Partitioning

- Splitting a database into parts
- Usually involves distributing partitions to different servers
- Note: Not the same as a partition in the CAP Theorem
- RDBMS often uses vertical partitioning
  - Separating commonly retrieved columns into different tables to improve retrieval speed
- NoSQL stores often use horizontal partitioning
  - Separating data (e.g., rows or documents) into different parts

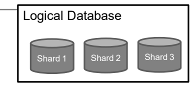
Vertical Partitioning

- Example: Search pictures



Source: NoSQL for mere Mortals by Sullivan

Horizontal Partitioning (Sharding)

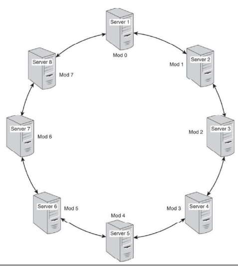


- Separating data (e.g., rows or documents) into different parts (shards)
- Shards are stored on separate servers
- This allows scaling out by adding additional servers
- Who determines where the Shard goes?
  - Traditional sharding architecture
    - Distribution based on a Shard key (One or more fields in a document that determined how documents are distributed into shards) Desired: Distribute load evenly
      - Range (e.g., all orders created in one month go into a shard)
      - List (e.g., product type is shard key and determines the shard)
      - Hash (hash function distributed documents evenly into shards)
    - An "omniscient master" determines where data should be located in the cluster, based on load and other factors
    - The Amazon Dynamo consistent hashing model, in which data is distributed across nodes of the cluster based on predictable mathematical hashing of a key value

Source: Next generation databases by Harrison (in chapter 8), NoSQL for mere Mortals by Sullivan

Using Hash Rings to evenly distribute Data on a Cluster

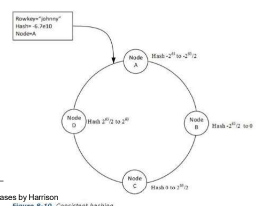
- Hash function = complex algorithms that takes a value as an input key and produces an address to store data
  - Goal: Evenly distribute the data
- Easy example: modulo
  - 8 nodes -> mod 8
  - Input key: 20
  - Node: 20 mod 8 = 4
  - Input key: 21
  - Node: 21 mod 8 = 5
  - Input key: 40
  - Node: 40 mod 8 = 0



NoSQL for Mere Mortals pg 109

Consistent Hashing

- Hash functions are usually more complicated than simply using a Mod function
- Consistent hashing: Each node is allocated a range of hash values logically arranged in a consistent hash ring.

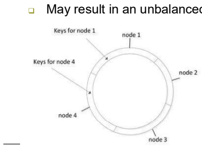


Source: Next generation databases by Harrison

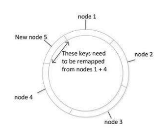
Figure 8-10. Consistent hashing

Growing / Shrinking a Cluster

- Solution 1: Remap values
  - Problem: If nodes are added or removed, all addresses have to be re-calculated and data moved to the correct address (node)
  - Using a method that reduces the number of data points that need to be moved
- Solution 2: Map new mode within an existing range
  - May result in an unbalanced cluster



Split of keys in 4-node cluster

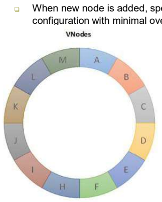


Adding a new node to the cluster

Source: Next generation databases by Harrison

Virtual Nodes

- Practice
  - Each server is added to the ring a multiple times.
- Advantage
  - Distributes each server evenly on the ring
  - When new node is added, specific virtual nodes can be reallocated for a more balanced configuration with minimal overhead



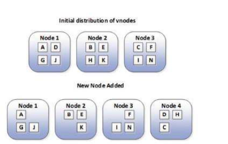
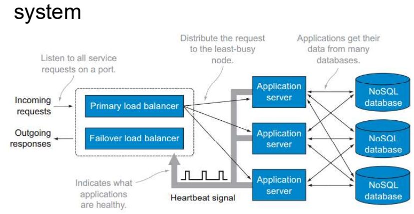


Figure 8-12. Using virtual nodes to partition data among physical nodes

Source: Next generation databases by Harrison

Load Balancing

- Distributing client requests to the nodes in the system



Making Sense of NoSQL pg. 179

Using high-availability Distributed Filesystems

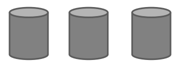
- Such as Hadoop Distributed File Systems
- Advantages
  - No need to use own file system
  - Allowing data replication
  - Rack and site awareness (e.g., to avoid replicating to same rack)
- Disadvantages
  - May not work on all operating systems
  - Learning curve and setup time required

Tunable Consistency

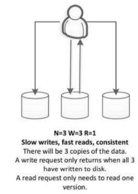
- Replica or copy = 1 data set
  - True: 1 Replica means we store 1 data set
  - False: 1 Replica means we store 2 data sets
- Consistency Level
  - Specify how many replicas need to be written before write request is complete
  - Specify how many replica need to be read for a read request
- Consistency versus Latency

Tunable Consistency

- N = number of replica the system should store (= replication factor)
- W = number of replica the system should write before the write request can be completed (= write consistency)
- R = number of replica the system will read when a read requests is made (= read consistency)
- Example: N = 3
  - 3 replica on 3 different nodes

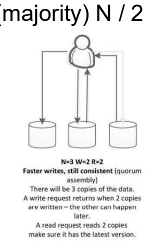


Highest Consistency, highest Latency (writes)

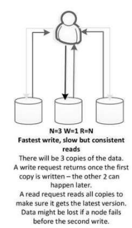


Balanced Consistency and Latency

- QUORUM (majority) N / 2 + 1



Highest Consistency, highest Latency (reads)



No Consistency, fast (writes and reads)