Modern Database Systems

Basic Principles

Doc. RNDr. Irena Holubova, Ph.D.
Irena.Holubova@matfyz.cuni.cz

NoSQL Overview

- Main objective: to implement a distributed state
  - Different objects stored on different servers
  - The same object replicated on different servers
- Main idea: give up some of the ACID features
  - To improve performance
- Simple interface:
  - Write (=Put): needs to write all replicas
  - Read (=Get): may get only one
- Strong consistency → eventual consistency

Basic Principles

- Scalability
  - How to handle growing amounts of data without losing performance
- CAP theorem
- Distribution models
  - Sharding, replication, consistency, …
  - How to handle data in a distributed manner

Scalability

Vertical Scaling (scaling up)

- Traditional choice has been in favour of strong consistency
  - System architects have in the past gone in favour of scaling up (vertical scaling)
    - Involves larger and more powerful machines
- Works in many cases but…
- Vendor lock-in
  - Not everyone makes large and powerful machines
    - Who do, often use proprietary formats
  - Makes a customer dependent on a vendor for products and services
    - Unable to use another vendor

Scalability

Vertical Scaling (scaling up)

- Higher costs
  - Powerful machines usually cost a lot more than commodity hardware
- Data growth perimeter
  - Powerful and large machines work well until the data grows to fill it completely
  - Even the largest of machines has a limit
- Proactive provisioning
  - Applications often have no idea of the final large scale when they start out
  - Scaling vertically = you need to budget for large scale upfront

Scalability

Horizontal Scaling (scaling out)

- Systems are distributed across multiple machines/nodes (horizontal scaling)
  - Commodity machines (cost effective)
  - Often surpasses scalability of vertical approach
- But…
- Fallacies of distributed computing:
  - The network is reliable
  - Latency is zero
  - Bandwidth is infinite
  - The network is secure
  - Topology does not change
  - There is one administrator
  - Transport cost is zero
  - The network is homogeneous

https://blogs.oracle.com/jag/resource/Fallacies.html

ACID

- Typical features of transactions we expect, e.g., in traditional relational DBMSs
- Database transaction = a unit of work (a sequence of related operations) in a DBMS
  - Typical example: transferring $100 from account A to account B
  - In fact two operations that are expected to be performed together:
    - Debit $100 to account A
    - Credit $100 to account B

ACID

- **Atomicity** – “all or nothing” = if one part of the transaction fails, then the entire transaction fails
- **Consistency** – brings the database from one consistent (valid, correct) state to another
  - ICs, triggers, …
- **Isolation** – effects of an incomplete transaction might not be visible to another transaction
- **Durability** – once a transaction has been committed, it will remain so
  - Power loss, errors, …
- Distributed systems:
  - Too expensive rules
  - Giving up some ACID feature = improvement of performance

CAP Theorem

Consistency

- After an update, all readers in a distributed system see the same data
- All nodes are supposed to contain the same data at all times
- Example:
  - A single database instance is always consistent
  - If multiple instances exist, all writes must be duplicated before write operation is completed

CAP Theorem

Availability

- All requests (reads, writes) are always answered, regardless crashes
- Example:
  - A single instance has an availability of 100% or 0%
  - Two servers may be available 100%, 50%, or 0%

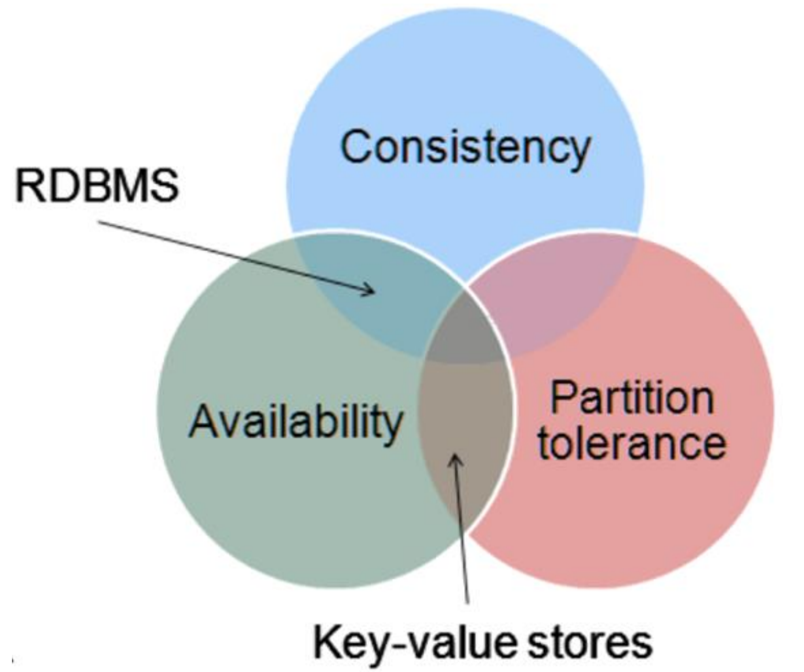
Partition Tolerance

- System continues to operate, even if two subsets of servers get isolated
- Example:
  - Failed connection will not cause troubles if the system is tolerant

CAP Theorem

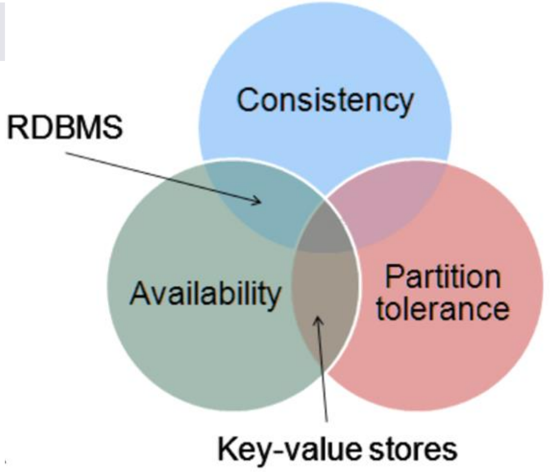
ACID vs. BASE

- **Theorem**: *Only 2 of the 3 guarantees can be given in a "shared-data" system.*
  - Proven in 2000, the idea is older
- (Positive) consequence: we can concentrate on two challenges
- ACID properties guarantee Consistency and Availability
  - **pessimistic**
  - e.g., database on a single machine
- BASE properties guarantee Availability and Partition tolerance
  - **optimistic**
  - e.g., distributed databases (key/value stores)



CAP Theorem

Criticism



- Not really a “theorem”, since definitions are imprecise
  - The real proven theorem has more limiting assumptions
- CP makes no “sense”, because it suggest never available
- No A vs. no C is asymmetric
  - No C = all the time
  - No A = only when the network is partitioned

CAP Theorem
Consistency

- A single-server system is a CA system
- Clusters naturally have to be tolerant of network partitions
  - CAP theorem: you can only get two out of three
  - Reality: you can trade off a little Consistency to get some Availability
    - It is not a binary decision

BASE

- In contrast to ACID
- Leads to levels of scalability that cannot be obtained with ACID
  - At the cost of (strong) consistency

Basically Available

- The system works basically all the time
- Partial failures can occur, but without total system failure

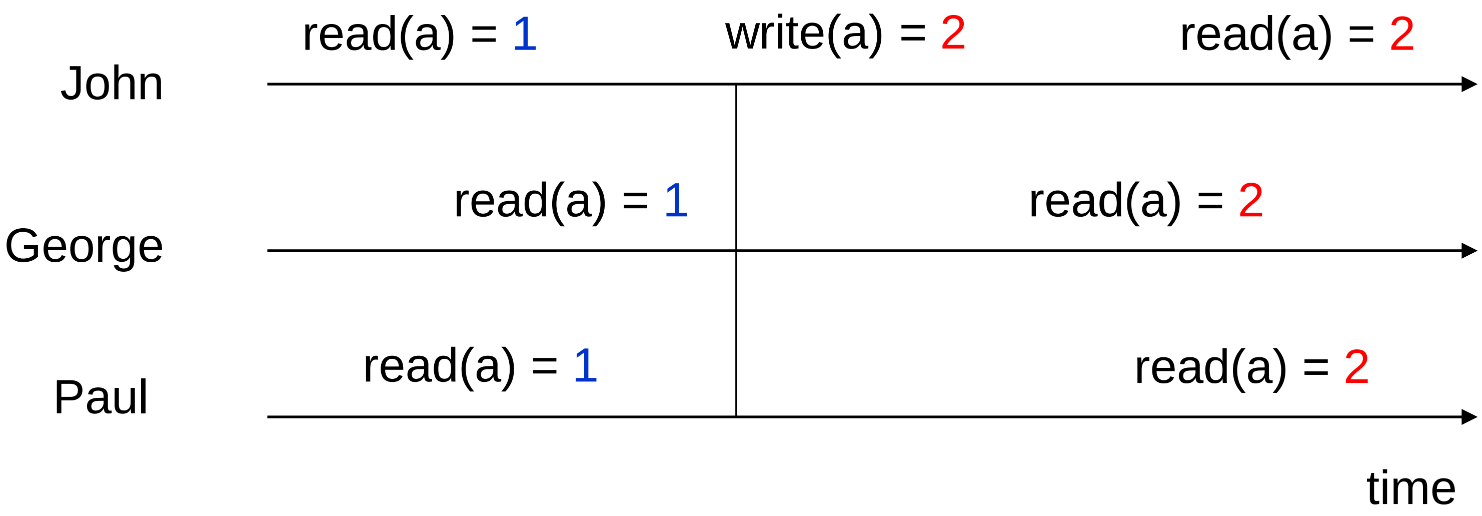
Soft State

- The system is in flux and non-deterministic
- Changes occur all the time

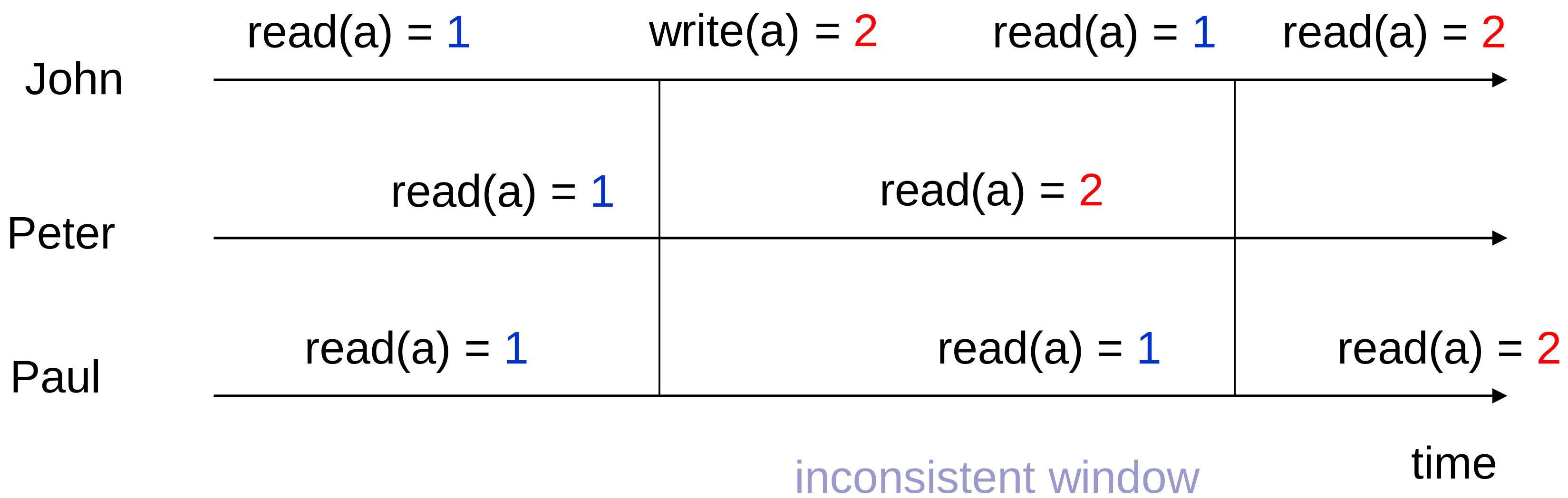
Eventual Consistency

- The system will be in some consistent state at some time in future

Strong Consistency



Eventual Consistency



Distribution Models

- Scaling out = running the database on a cluster of servers
- Two orthogonal techniques to data distribution:
  - Replication – takes the same data and copies it over multiple nodes
    - Master-slave or peer-to-peer
  - Sharding – puts different data on different nodes
- We can use either or combine them

Distribution Models

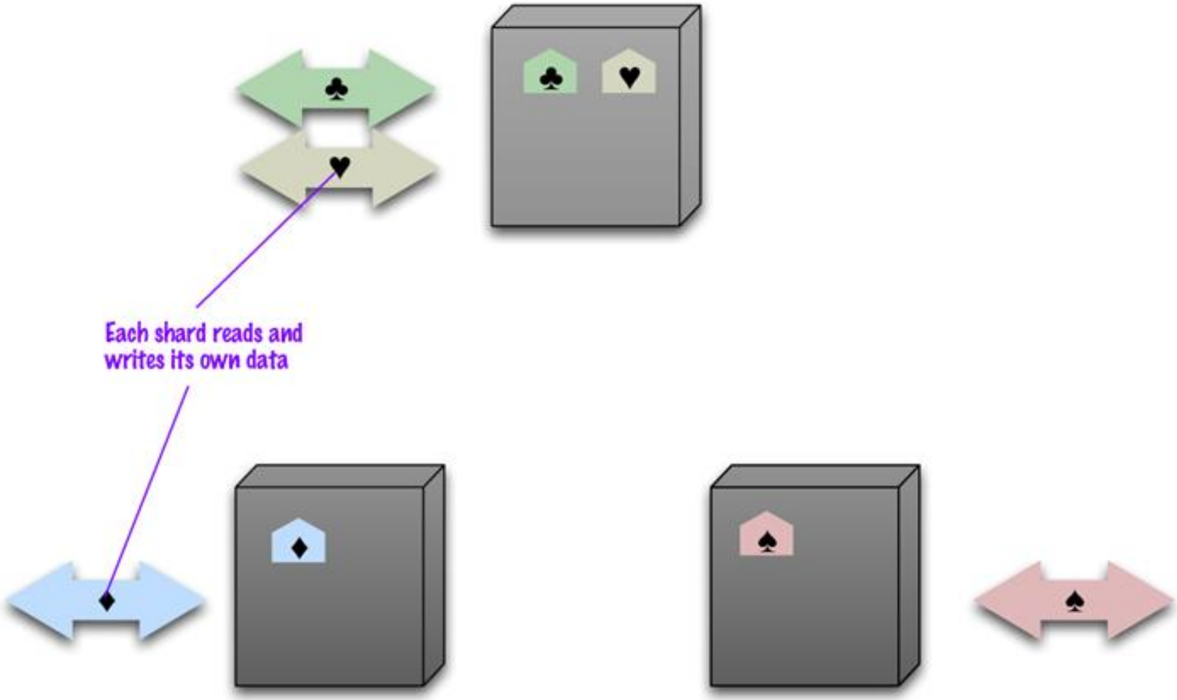
Single Server

- No distribution at all
  - The database runs on a single machine
- It can make sense to use NoSQL with a single-server distribution model
  - Graph databases
    - The graph is “almost” complete → it is difficult to distribute it

Distribution Models

Sharding

- Horizontal scalability → putting different parts of the data onto different servers
- Different people are accessing different parts of the dataset



Distribution Models

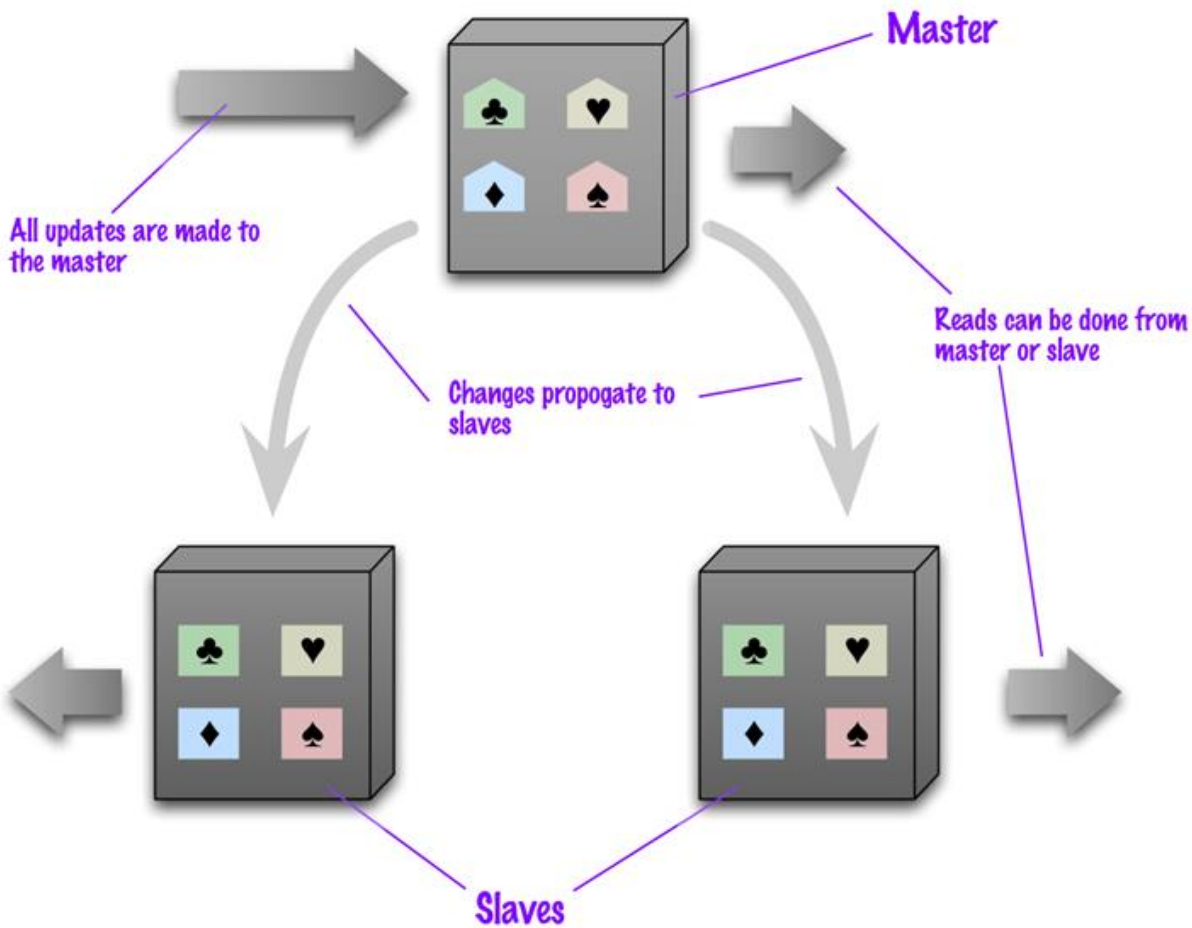
Sharding – how to?

- The ideal case is rare
- To get close to it we have to ensure that data that is accessed together is stored together
- How to arrange the nodes:
  a. One user mostly gets data from a single server
  b. Based on a physical location
  c. Distribute across the nodes with equal amounts of the load
- Many NoSQL databases offer auto-sharding
- A node failure makes shard’s data unavailable
  - Sharding is often combined with replication

Distribution Models

Master-slave Replication

- We replicate data across multiple nodes
- One node is designed as primary (master), others as secondary (slaves)
- Master is responsible for processing any updates to that data



Distribution Models

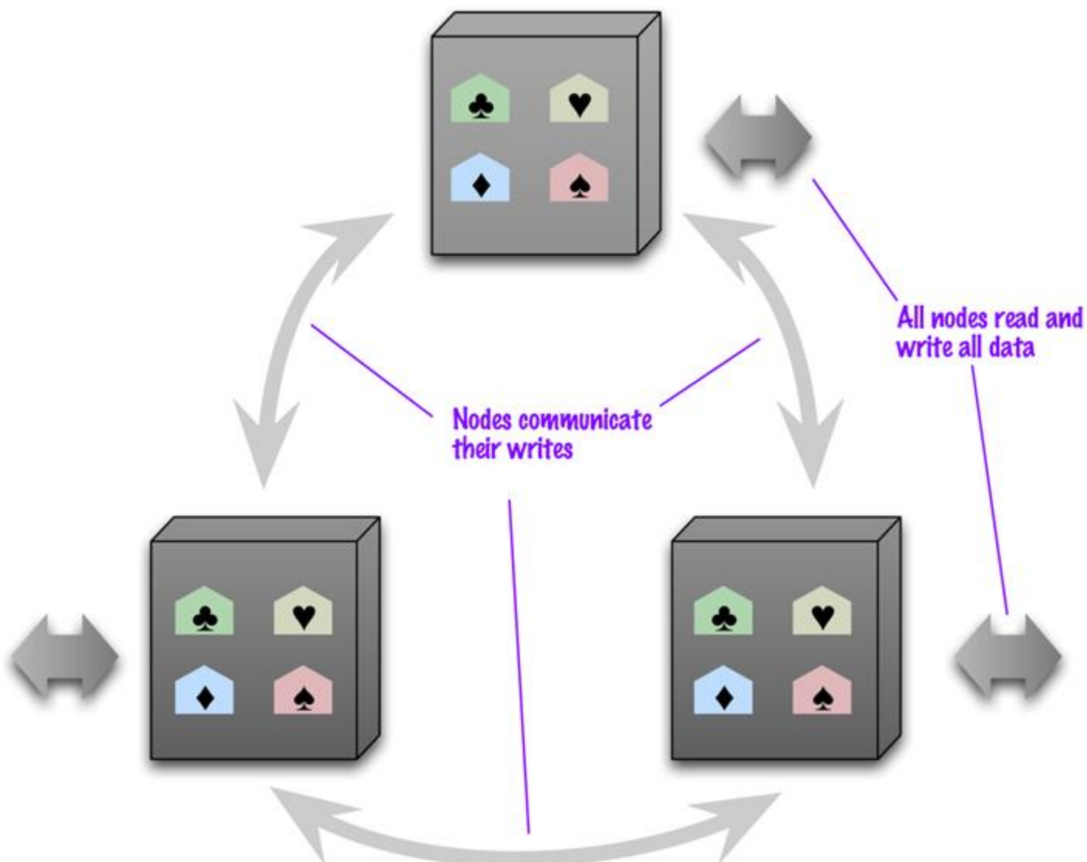
Master-slave Replication

- For scaling a read-intensive dataset
  - More read requests → more slave nodes
  - The master fails → the slaves can still handle read requests
    - A slave can be appointed a new master quickly (it is a replica)
- Limited by the ability of the master to process updates
- Masters are appointed manually or automatically
  - User-defined vs. cluster-elected

Distribution Models

Peer-to-peer Replication

- Problems of master-slave replication:
  - Does not help with scalability of writes
    - The master is still a bottleneck
  - Provides resilience against failure of a slave, but not of a master
- Peer-to-peer replication: no master
  - All the replicas have equal weight



Distribution Models

Peer-to-peer Replication

- Problem: consistency
  - We can write at two different places: a write-write conflict
- Solutions:
  - Whenever we write data, the replicas coordinate to ensure that we avoid a conflict
    - At the cost of network traffic
  - But we do not need all the replicas to agree on the write, just a majority

Distribution Models

Combining Sharding and Replication

- Master-slave replication and sharding:
  - We have multiple masters, but each data item only has a single master
  - A node can be a master for some data and a slave for others
- Peer-to-peer replication and sharding:
  - A common strategy, e.g., for column-family databases
  - A good starting point for peer-to-peer replication is to have a replication factor of 3, so each shard is present on three nodes

Consistency

Write (update) Consistency

- Problem: two users want to update the same record (write-write conflict)
  - Issue: lost update
    - A second transaction writes a second value on top of a first value written by a first concurrent transaction
    - The first value is lost to other transactions running concurrently which need, by their precedence, to read the first value
    - The transactions that have read the wrong value end with incorrect results
- Pessimistic (preventing conflicts from occurring) vs. optimistic solutions (lets conflicts occur, but detects them and takes actions to sort them out)
  - Write locks, conditional update, save both updates and record that they are in conflict, …

Consistency

Read Consistency

- Problem: one user reads, other writes (read-write conflict)
  - Issue: inconsistent read
    - When a transaction reads object x twice and x has different values
    - Between the two reads another transaction has modified the value of x
- Relational databases support ACID transactions
- NoSQL databases usually support atomic updates within a single aggregate
  - But not all data can be put in the same aggregate
- Update that affects multiple aggregates leaves open a time when clients could perform an inconsistent read
  - Inconsistency window
- Another issue: replication consistency
  - A special type of inconsistency in case of replication
  - Ensuring that the same data item has the same value when read from different replicas

Consistency

Quorums

- How many nodes need to be involved to get strong consistency?
- Write quorum: $W > N/2$
  - N = the number of nodes involved in replication (replication factor)
  - W = the number of nodes participating in the write
    - The number of nodes confirming successful write
  - “If you have conflicting writes, only one can get a majority.”
- How many nodes do we need to contact to be sure we have the most up-to-date change?
- Read quorum: $R + W > N$
  - R = the number of nodes we need to contact for a read
  - „Concurrent read and write cannot happen.“

References

- http://nosql-database.org/
- Pramod J. Sadalage – Martin Fowler: **NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence**
- Eric Redmond – Jim R. Wilson: **Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement**
- Sherif Sakr – Eric Pardede: **Graph Data Management: Techniques and Applications**
- Shashank Tiwari: **Professional NoSQL**