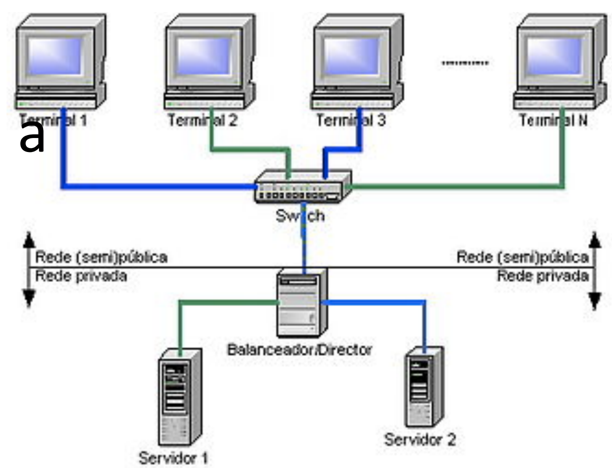
CSCU9YQ - NoSQL Databases
Lecture 5: Consistency and Transactions

Gabriela Ochoa

Terminology

- Node: a computer that offers processing, local storage, and it is connected to other nodes.
- Cluster: a set of connected nodes that work together, in some ways they can be seen as a single system. The nodes perform similar tasks
  - Located on a particular rack on a data centre
  - Geographically distributed



Terminology

- Sharding: A database architecture that partitions data by key ranges and distributes the data among two or more database instances. Sharding enables horizontal scaling.
- Replication: A feature allowing multiple database servers to share the same data, thereby ensuring redundancy and facilitating load balancing

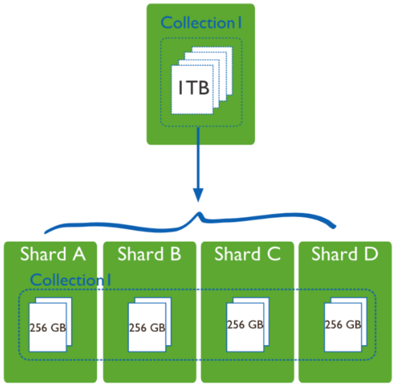
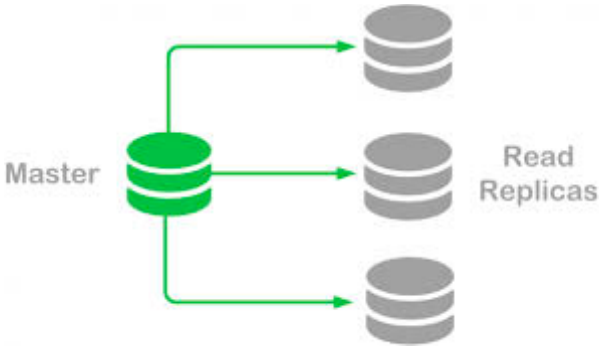
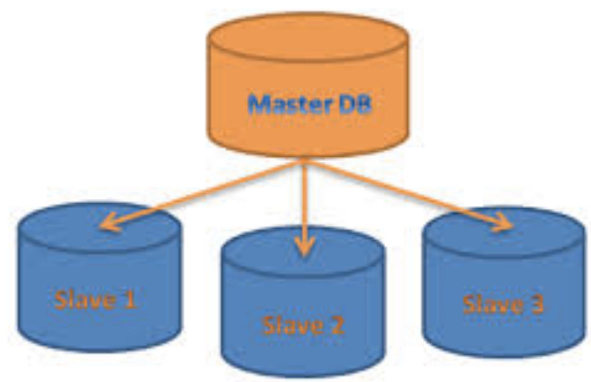


Diagram of a large collection with data distributed across 4 shards.



Primary-secondary replication

- All modifications are made on the primary, and these changes are pushed out to the secondary nodes.
- A problem occurs if the primary node fails before the secondary nodes are modified
  - Secondaries elect a new master, and updates continue.
  - When the old primary is brought back, conflicting changes may need to be reconciled



Note: Keeping all the replicas in the same rack, or even the same data centre increases the risk of data loss

Redundancy and Data Availability

- Replication provides redundancy and increases data availability.
- With multiple copies of data on different database servers, replication provides a level of fault tolerance against the loss of a single database server.
- High Availability DB Systems:
  - Designed for durability, redundancy and failure tolerance.
  - Applications supported by the system can operate continuously and without downtime for a long period of time

Consistency

- NoSQL DBs typically maintain multiple copies of the data for availability and scalability purposes.
- Challenge: How to guarantee consistency of the data across multiple copies? The replicas can become inconsistent.
- Consistency, for a distributed database means:
  - Read consistency
  - Update consistency

Read Consistency

- Read consistency means that two readers see the same data after an update.
- A Read inconsistency can occur if an update to a document hasn't propagated from one replica to another
  - One person sees one value returned, and another person sees a different value.
  - Example: a user updates her address, but the the update has not propagated from the primary to the secondaries

Update Consistency

- When two modifications are issued simultaneously, one update succeeds, and the other is notified that the record it's trying to update has been modified since the last read.
- Write-write conflict: two people updating the same data item at the same time

Example: Bank process is adding interest, while person is removing cash from machine Removed cash is overwritten by new interest calculation!

Bank: Adding Interest	Person: Removing cash
Read balance Calculate interest Add to balance figure Write new balance	Read balance Subtract amount withdrawn Write new balance

What is a Transaction?

- The notion of a transaction was designed to maintain read and update consistency.
- You learned about transactions when studying Relational DBs.
  - A transaction is a series of database operations
  - It involves locking of fields to prevent other processes writing until a transaction is complete
  - ACID transactions are core to relational databases

ACID Transactions

- **Atomic** – Cannot be broken into smaller components – All or Nothing
- **Consistent** – Always leave the database in a consistent state
- **Independent** – Do not interfere with other transactions
- **Durable** – Once complete, cannot be undone (as in the bank example)

Transactions in MongoDB

- MongoDB write operations are Atomic at the level of individual records (documents)
- Embedded documents and arrays capture relationship between data in a single document (instead of normalised multiple tables as in relational DBs)
- Therefore, Atomic single record-operators provide transactions that meet the integrity of the majority of applications
- However, some applications need multi-record transactions. MongoDB have added support for ACID transactions across multiple records.

Consistency and Transactions in NoSQL DBs

- Most NoSQL databases relax ACID constraints.
- NoSQL DBs can impose different guarantees on the consistency of the data across copies
  - Strongly consistent: writes by the application are immediately visible in subsequent queries
  - Eventually consistent: writes are not immediately visible. Depending on which data replica is serving the query. There can be a delay before all replicas are updated.

Applications can have different requirements for Consistency

Examples:

- Facebook – not a problem if a friend in the UK can see a new photo of your cat while a friend in America has to wait a few more seconds before it appears
- Paypal – needs to be sure the balance it reads is correct, and that another node hasn’t spent the remaining money

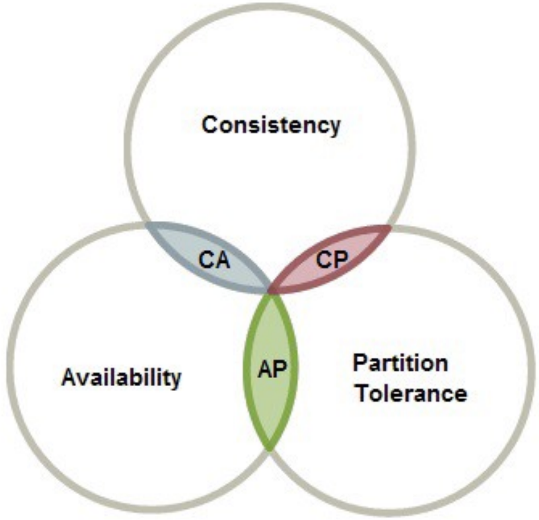
Consistency on NoSQL DB

- Document DBs and Graph DBs can be consistent or eventually consistent
- MongoDB provides tuneable consistency
  - By default, data is consistent: all writes and reads access the primary copy of the data
  - As an option, read queries can be issued against a secondary copies where data maybe eventually consistent (not synchronised yet)
  - The consistency choice is made at the query level

Eventually Consistent Systems

- There is a period of time in which all copies of the data are not synchronised
- This may be acceptable for read-only applications and data stores that do not change often (e.g. Historical Archives)
- It can also be acceptable for write-intensive applications in which the DB capture logs, which will only be read a later point in time

The CAP Theorem



A distributed DB can only have 2/3:

Consistency:

- All nodes see the same data at the same time
- Performing a read operation will return the value of the most recent write operation causing all nodes to return the same data

Availability

- Every client gets a response, regardless of the state of any individual node in the system (success/failure)
- Requires that the system remains operational 100% of the time.

Partition Tolerance

- The system continues to run, despite the number of messages being delayed by the network between nodes.
- The system can sustain any amount of network failure

Trade off Consistency vs. Availability

- Generally, we need a DB that is Partition Tolerant
- We can only do that by losing either consistency or availability
  - It can keep consistent by making some nodes unavailable (CP)
  - Or stay available but accept that it will become inconsistent (AP)

Maintaining Consistency

- One way to maintain consistency is to make sure updates are fully propagated or writes are forced through a primary node
- That means that a secondary node might be reachable on the network, but still ‘unavailable’ because it either hasn’t been updated or can’t contact the primary node
- So available really means **able to respond**

Maintaining Consistency

- One possible solution
- In the case where writes need to go through a master node, but reads don't, availability depends on the request
  - Read available
  - Write unavailable

Example

- Hotel booking system
  - Read from a secondary (might be out of date)
  - Write through the primary
    - If no rooms available, report room was lost
    - If primary is not available, either report error or write to secondary and deal with conflict later
  - Keeps reads (most frequent query) fast using secondary
  - Keeps writes consistent using primary

Really a Continuum

- In reality, the CAP qualities are not all or nothing options, but a continuum. You need to think about:
  - How much do I need consistency?
  - How long are users prepared to wait for it?
  - Can I get away with write consistency only?
  - How can conflicts be solved later, and at what cost?

NoSQL and ACID Transactions

- The Relational model wasn't ACID when it was originally created. It added ACID support over time.
- NoSQL databases are also following this trend.

Summary

- Different consistency models pose different trade-offs in the areas of consistency and availability
- MongoDB provides tuneable consistency, defined at the query level
- Eventually consistent system provide some advantages for inserts, at the costs of making reads, updates and deletes more complex.
- Most NoSQL databases provide single record atomicity. This is sufficient for many applications, but not all