Moving Beyond the Relational Model

Created	@January 27, 2025 9:20 AM
Class	DS 4300
Reviewed	☐

Benefits of the Relational Model

- Mostly Standard Data Model and Query Language
- ACID Compliance
    - Atomicity, Consistency, Isolation, Duration
- Works well with highly structured data
- Can handle large amounts of data
- Well understood, lots of tooling, lots of experience

Relational DB Performance

- Many ways that a RDBMS increases efficiency:
    - indexing
    - directly controlled storage
    - column oriented storage vs row oriented storage
    - query optimization
    - caching/preferencing
    - materialized views
    - precompiled stored procedures
    - data replication and partitioning

Transaction Processing

- Transaction - a sequence of one or more of the CRUD operation performed as a single, logical unit of work
    - Either the entire sequence succeeds (COMMIT)
    - OR the entire sequence fails (ROLLBACK or ABORT)
- Help ensure
    - Data integrity
    - Error recovery
    - Concurrency Control
    - Reliable Data Storage
    - Simplified Error Handling

ACID Properties

- Atomicity
    - transaction is treated as an atomic unit - it is fully executed or no parts of it are executed
- Consistency
    - a transaction takes a database from one consistent state to another consistent state
    - consistent state - all data meets integrity constraints
- Isolation
    - Two transactions T1 and T2 are being executed at the same time but cannot affect each other
    - If both T1 and T2 are reading the data - no problem
    - If T1 is reading the same data that T2 may be writing, can result in:
        - Dirty Read
            - A transaction T1 is able to read a row that has been modified by another transaction T2 that hasn't yet executed a COMMIT

        - Non-repeatable Read
            - two queries in a single transaction T1 execute a SELECT but get different values because another transaction T2 has changed data and committed
        - Phantom reads
            - when a transaction T1 is running and another transaction T2 adds or deletes rows from the set T1 is using
- Durability
    - Once a transaction is completed and committed successfully, its changes are permanent
    - Even in the event of a system failure, committed transactions are preserved

Cons of RDB

- sometimes, schemas evolve over time
- not all apps may need the full strength of ACID compliance
- joins can be expensive
- a lot of data is semi-structured or unstructured (JSON, XML, etc.)
- Horizontal scaling presents challenges
- some apps need something more performant

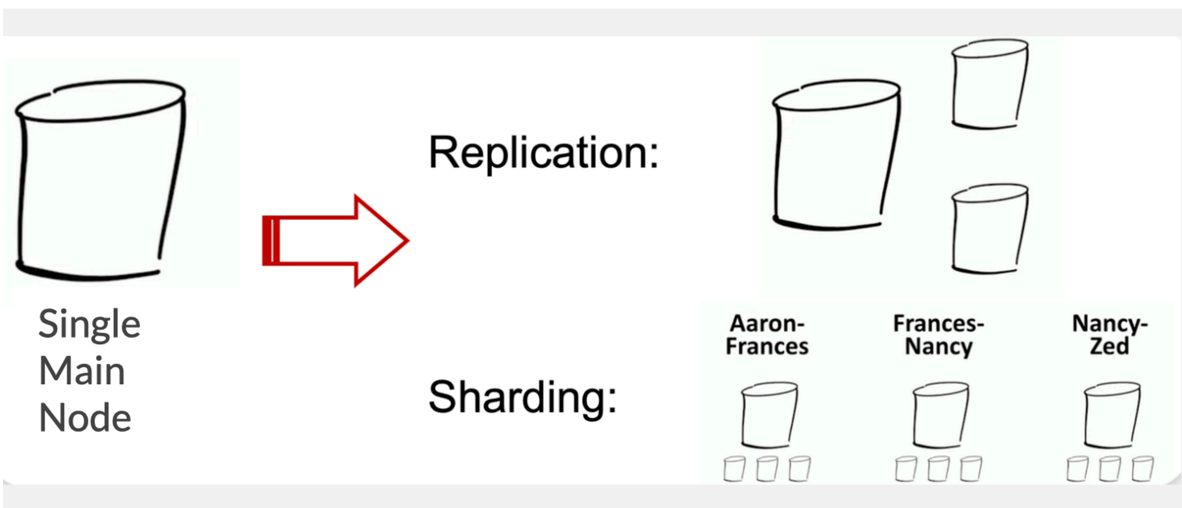
Scalability

- Conventional Wisdom: Scale vertically (up with bigger more powerful systems) until the demands of high-availability make it necessary to scale out with some type of distributed computing model
    - Vertical Scaling: more power / computing capability on same system
    - Horizontal Scaling: more computers / compute nodes

- Why? Scaling up is easier - no need to really modify your architecture. But there are practical and financial limits
- However: there are modern systems that make horizontal scaling less problematic

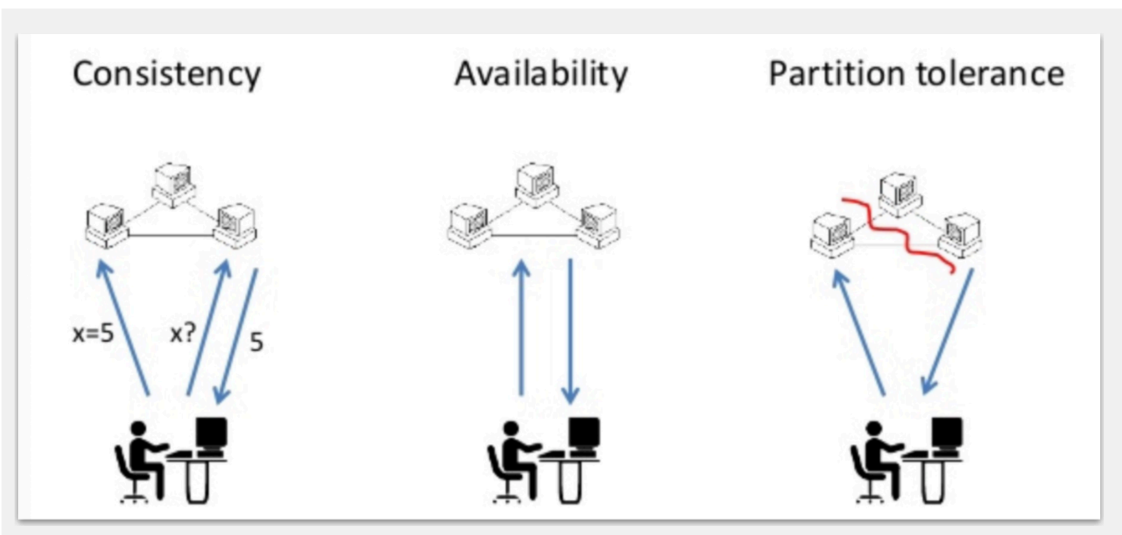
Distributed System

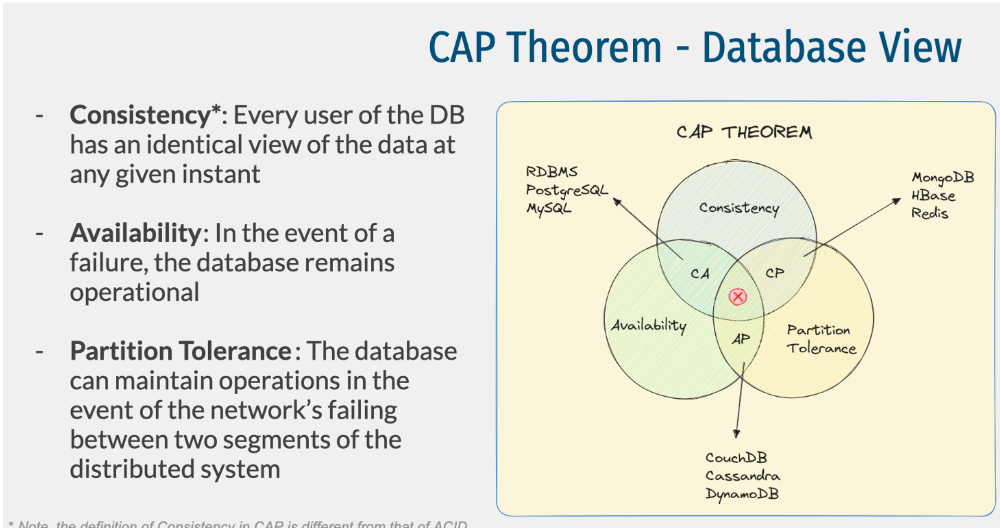
- a distributed system is a "collection of independent computers that appear to its users as one computer."
- Characteristics of Distributed Systems:
    - computers operate concurrently
    - computers fail independently
    - no shared global clock
- ▼ Distributed Storage - 2 Directions

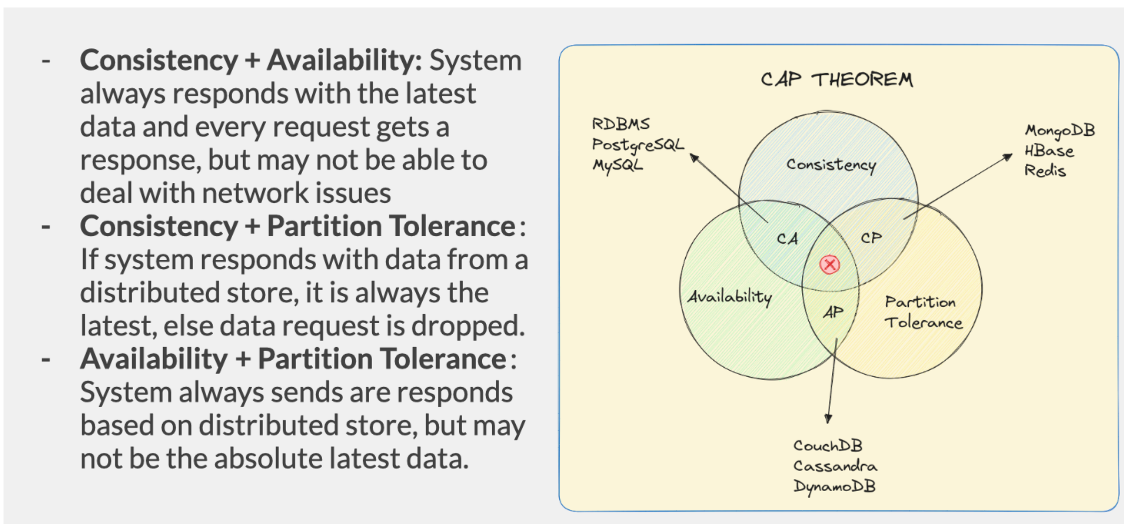


- Distributed Data Stores
    - Data is stored on > 1 node, typically replicated
        - i.e each block of data is available on N nodes
    - Distributed databases can be relational or non-relational
        - MySQL and PostgreSQL support replication and sharding

        - CockroachDB - new player on the scene
        - Many NoSQL systems support one or both models
    - But remember: Network partitioning is inevitable!
        - network failures, system failures
        - Overall system needs to be Partition Tolerant
            - System can keep running even w/ network partition
- The CAP Theorem
    - The CAP Theorem states that it is <u>impossible</u> for a distributed data store to simultaneously provide more than two out of the following three guarantees:
        - Consistency - every read receives the most recent write or error thrown
        - Availability - every request receives a (non-error) response - but no guarantee that the response contains the most recent write
        - Partition Tolerance - the system can continue to operate despite arbitrary network issues
    - ▼ Diagrams







- What this really means?
  - If you cannot limit the number of faults, requests can be directed to any server, and you insist on serving every request, then you cannot possibly be consistent
- But it is interpreted as:
  - You must always give up something: consistency, availability, or tolerance to failure