Notes on CAP Theorem, ACID, and BASE

1 CAP Theorem

The CAP theorem was proposed by Eric Brewer in 2000 and formally proved by Seth Gilbert and Nancy Lynch in 2002. It describes the trade-offs made by a distributed system during a network partition.

CAP stands for:

- **Consistency:** Every client sees the same and most recently committed data.
- **Availability:** Every request receives a response, even though the response may not always contain the latest data.
- **Partition Tolerance:** The distributed system continues operating even when communication between some servers or nodes is interrupted.

A **network partition** occurs when some servers in a distributed system cannot communicate with other servers, even though the servers themselves may still be running.

During a network partition, a distributed system must choose between:

- **CP — Consistency and Partition Tolerance**
  - The system prioritizes correct and up-to-date data.
  - Some requests may be rejected, blocked, or delayed.
  - It sacrifices availability during the partition.
  - A simple interpretation is: "If I cannot guarantee correctness, I will not answer."
- **AP — Availability and Partition Tolerance**
  - The system continues responding to requests.
  - Some responses may contain stale or temporarily inconsistent data.
  - It sacrifices immediate consistency during the partition.
  - A simple interpretation is: "I will answer, but my information may be slightly outdated."
- **CA — Consistency and Availability**
  - The system provides both consistency and availability.
  - This combination is possible only when there is no network partition.
  - A truly distributed system cannot guarantee CA when a partition occurs.

1.1 CAP and MongoDB

MongoDB commonly uses replica sets containing one primary node and multiple secondary nodes.

- The primary node normally accepts write operations.
- Secondary nodes maintain copies of the data.
- If the primary becomes isolated from the majority of the replica set, it normally stops accepting writes.
- The remaining majority can elect a new primary.
- This behavior makes MongoDB generally behave like a **CP system for writes**.
- MongoDB also allows developers to configure read concern, write concern, and read preference.
- Reading from secondary nodes may improve read availability, but the returned data may be stale.

2 ACID Properties

ACID describes the properties of reliable database transactions.
ACID stands for:

- **Atomicity:** A transaction is completed entirely or not completed at all.
- **Consistency:** A transaction moves the database from one valid state to another valid state while following database constraints and rules.
- **Isolation:** Concurrent transactions behave as though they were executed independently.
- **Durability:** Once a transaction is committed, its changes remain stored even after a crash or system failure.

For example, consider transferring money from Account A to Account B:

1. The amount is deducted from Account A.
2. The same amount is added to Account B.

Atomicity ensures that either both operations happen or neither operation happens.

3 BASE Properties

BASE is a model commonly associated with large-scale distributed and NoSQL database systems.

BASE stands for:

- **Basically Available:** The system continues responding even if some nodes fail or become unreachable.
- **Soft State:** The state of the system may temporarily change because updates are still being propagated between servers.
- **Eventually Consistent:** Different copies of the data may temporarily disagree, but they will become consistent after updates have propagated.

For example, after changing a profile picture on a social-media platform, some users may briefly see the old picture while others see the new one. Eventually, all servers display the updated picture.

4 Comparison of CAP, ACID, and BASE

5 Key Differences

- **CAP** explains what a distributed system must choose during a network partition.
- **ACID** defines the correctness and reliability properties of database transactions.
- **BASE** describes an availability-oriented approach in which data may become consistent after a short delay.
- CAP consistency and ACID consistency do not mean exactly the same thing.
  - In CAP, consistency means that clients observe the same latest data.
  - In ACID, consistency means that a transaction preserves database constraints and valid states.
- A database can support ACID transactions while also making CAP trade-offs when deployed as a distributed system.
- ACID and BASE are not always absolute opposites. Modern databases may provide ACID transactions for some operations and eventual consistency for others.

6 Simple Summary

- **CP:** Correct data is more important than always responding.
- **AP:** Always responding is more important than immediately showing the latest data.
- **ACID:** Transactions must be reliable and correct.
- **BASE:** The system remains available, and data becomes consistent eventually.

Aspect	CAP Theorem	ACID	BASE
Full form	Consistency, Availability, Partition Tolerance	Atomicity, Consistency, Isolation, Durability	Basically Available, Soft State, Eventually Consistent
Main purpose	Explains the trade-off made by distributed systems during a network partition	Ensures reliable and correct database transactions	Supports highly available and scalable distributed systems
Primary focus	Consistency versus availability during network failure	Transaction correctness and reliability	Availability, scalability, and eventual consistency
Consistency model	May prioritize strong consistency or availability, depending on whether the system behaves as CP or AP	Usually provides strong transactional consistency	Usually provides eventual consistency
Availability	Depends on whether the system selects CP or AP during a partition	A transaction may fail or wait to preserve correctness	The system generally remains available even during partial failures
Network partitions	Directly concerned with network partitions	Does not by itself describe network-partition behavior	Designed to operate effectively in distributed environments where failures may occur
Transaction behavior	CAP is not a transaction model	Transactions follow strict all-or-nothing rules	Transactions may be more flexible and may not immediately update every replica
Typical use cases	Distributed databases, cloud systems, replica sets, and distributed services	Banking, payments, inventory, orders, and other correctness-sensitive systems	Social networks, recommendation systems, analytics, logging, and large-scale web systems
Typical database association	MongoDB replica sets, Cassandra, distributed key-value stores, and other distributed databases	PostgreSQL, MySQL, Oracle Database, and SQL Server	Cassandra, Dynamo-style databases, CouchDB, and other distributed NoSQL systems
Main compromise	A partition forces a choice between consistency and availability	May reduce performance or availability to maintain transaction correctness	Allows temporary inconsistency to maintain availability and scalability

Table 1: Comparison of CAP, ACID, and BASE