2CEIT702

BIG DATA ANALYTICS

Eventual Consistency

Eventual Consistency is a consistency model used in distributed systems. It ensures that, given enough time without new updates, all replicas of a given data item will converge to the same value. This means that while data may be temporarily inconsistent across different nodes, it will eventually become consistent.

Strong Consistency

Strong Consistency is a consistency model used in distributed systems that ensures that any read operation will always return the most recent write for a given piece of data. In other words, once a write operation is confirmed, all subsequent reads will see that write, regardless of which node they access.

Strong Consistency *vs* Eventual Consistency

Feature	**Strong Consistency**	**Eventual Consistency**
Definition	Guarantees that all reads return the latest write immediately.	Guarantees that all replicas will converge to the same value over time, given no new updates.
Read After Write	Always returns the most recent data immediately after a write.	May return stale data for a time after a write.
Data Synchronization	Requires synchronization between nodes, which can cause delays.	Allows temporary inconsistencies, leading to quicker responses.
Latency	Typically has higher latency due to coordination.	Generally has lower latency as it doesn't require immediate synchronization.
Use Cases	Critical applications like banking, online transactions, and booking systems.	Applications where high availability is prioritized, such as social media feeds or caching.
Failure Handling	May lead to unavailability during network partitions.	Remains available even during network issues, with eventual consistency achieved later.
Complexity	Simpler for the end-user since data is always up-to-date.	More complex for users as they may encounter stale data temporarily.

ACID Properties *Vs* BASE Properties

Aspect	ACID Properties	BASE Properties
Focus	Accuracy and reliability.	Availability and flexibility.
Data Availability	Requires complete data accuracy at all times.	Always available, even if not all data is current.
Data State	Must be consistent and correct at all times.	Can change over time (soft state).
Consistency	Immediate consistency is guaranteed.	Eventually becomes consistent.
Use Case	Best for transactions requiring strict accuracy, like banking.	Good for large-scale analytics and real-time data.
Example	Bank transactions that must be accurate immediately.	Live sales dashboards showing real-time updates.

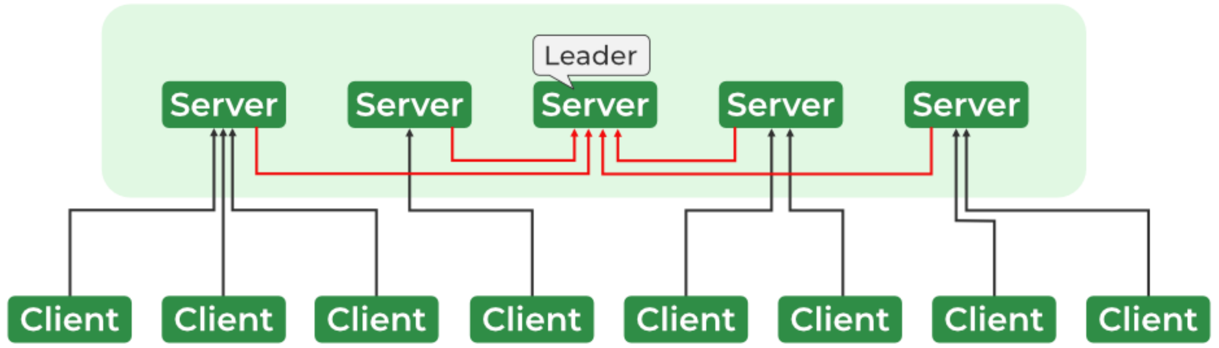
ZooKeeper:-

Apache Zookeeper is a distributed, open-source coordination service for distributed systems. It provides a central place for distributed applications to store data, communicate with one another, and coordinate activities. Zookeeper is used in distributed systems to coordinate distributed processes and services.

The ZooKeeper architecture consists of a hierarchy of nodes called znodes, organized in a tree-like structure.

Each znode can store data and has a set of permissions that control access to the znode.

At the root of the hierarchy is the root znode, and all other znodes are children of the root znode.



Zookeeper Services

Znode

A **Znode** is a data node in the ZooKeeper data tree, which is a hierarchical structure similar to a filesystem. Each Znode can store data and has an associated version number, allowing for efficient coordination and management of distributed systems.

Znode Type	Description
Sequential znode	A **sequential znode** in ZooKeeper is a special type of znode that automatically appends a monotonically increasing sequence number to its name when it is created. This sequence number ensures that each sequential znode has a unique name, even if multiple clients create znodes with the same base name concurrently.
Persistent znode	A **persistent znode** in ZooKeeper is a type of znode that remains in the ZooKeeper ensemble until it is explicitly deleted by a client.
Ephemeral znode	An **ephemeral znode** in ZooKeeper is a type of znode that exists only for the duration of the client session that created it.