MODULE - 2

1. Write a note on Distribution model.

Distribution model: The distribution model in NoSQL refers to how data is distributed and stored across multiple servers or nodes in a cluster.

Replication: Data replication is a distribution model where copies of the data are stored on multiple servers. Each replica holds a complete copy of the dataset, ensuring high availability and fault tolerance.

Consistent Hashing: Consistent hashing is a technique used to determine which server or node will store a specific piece of data. It ensures a balanced distribution of data across the nodes and minimizes data movement when nodes are added or removed from the cluster.

Scalability: The distribution model in NoSQL databases allows for horizontal scalability. By distributing data across multiple servers, the database can handle large datasets and accommodate increasing data volumes.

Fault tolerance: With data replication, if one server fails, the data can still be accessed from other replicas, ensuring fault tolerance and data availability.

Performance: By distributing data and workload across multiple servers, the distribution model improves performance by enabling parallel processing and reducing data access latency.

Data retrieval: When retrieving data, the distribution model affects how queries are executed. Depending on the distribution model, the database may need to coordinate and merge results from multiple nodes to provide a complete response.

2. Explain the Single Server in NoSql.

Definition: The Single Server model in NoSQL means that the entire database system, including data storage, processing, and management, operates on a single server or machine.

Database Operations: In this model, all database operations, such as storing, retrieving, updating, and querying data, are performed on the same server. The server is responsible for handling all the tasks associated with managing the database.

Limitations: The Single Server model has certain limitations. Firstly, it can handle only a limited amount of data and traffic, as the resources of a single server are finite. As the data volume and workload increase, the server may become overloaded, leading to performance issues.

Scalability: Scaling a Single Server NoSQL database model can be challenging. Since all the data and operations are concentrated on a single server, adding more resources to that server may provide some performance improvement, but it has limitations in terms of accommodating substantial growth or sudden spikes in traffic.

Use Cases: The Single Server model is often suitable for small-scale applications, prototypes, or development environments where the data volume and workload are minimal. It can be an efficient and cost-effective choice when the requirements are modest, and the focus is more on simplicity rather than scalability and high availability.

3. Explain the Master-slave replication in NoSql.

Definition: Master-Slave replication is a data replication technique used in NoSQL databases. It involves a master node that handles write operations and one or more slave nodes that replicate the data from the master.

Write Operations: The master node receives write requests from clients and performs modifications to the data. These changes are recorded in a replication log.

Data Replication: Slave nodes periodically retrieve the replication log from the master node and apply the recorded write operations to their local copies of the data. This ensures that the slave nodes have an up-to-date replica of the data.

Read Operations: Clients can perform read operations on both the master and slave nodes. However, it is common to direct read requests to the slave nodes to distribute the read workload and reduce the burden on the master node.

Benefits: Master-Slave replication provides data redundancy, fault tolerance, and read scalability. In the event of a master node failure, one of the slave nodes can be promoted as the new master, ensuring continuous availability of the database.

4. Write a note on combining sharding and replications.

Sharding: Sharding involves dividing the data into smaller subsets called shards and distributing them across different servers. It enables horizontal scalability by allowing data to be spread across multiple servers, reducing the load on individual servers.

Replication: Replication involves creating copies of data and storing them on multiple servers. It ensures high availability and fault tolerance, as data can still be accessed from other replicas if one server fails.

Combining Sharding and Replication: Combining sharding and replication in NoSQL databases allows for both horizontal scalability and fault tolerance.

Sharding with Replication: Each shard in the sharded environment can have its own set of replicas. This means that each shard has multiple copies of its data stored on different servers.

Benefits: Combining sharding and replication provides several benefits. It enables distributing the data across multiple servers for scalability while ensuring redundancy and fault tolerance through replication. This architecture improves both read and write performance by allowing parallel processing across shards and replicas.

5. Explain the peer-to-peer replication in NoSql.

Peer-to-Peer Replication: Peer-to-peer replication is a data replication technique used in NoSQL databases. It involves multiple nodes that act as both data sources and data replicas.

Equal Roles: In peer-to-peer replication, all participating nodes have equal roles. Each node can act as a source of data and replicate data from other nodes.

Data Exchange: Nodes in a peer-to-peer replication setup exchange data directly with each other. They share their data with other nodes in the network, and at the same time, receive data from other nodes to update their own replica.

Distributed Updates: When a node receives an update or modification to the data, it propagates that change to other nodes in the network. This ensures that all nodes have an up-to-date copy of the data.

Benefits: Peer-to-peer replication offers several advantages. It provides high availability and fault tolerance since data is replicated across multiple nodes. In the event of a node failure, other nodes can still provide access to the data. Additionally, peer-to-peer replication can improve scalability and load balancing as each node can handle read and write requests.

6. Explain the Read Consistency in NoSql.

Read consistency in NoSQL refers to ensuring that a read operation returns the most up-to-date and accurate data, regardless of which node in a distributed database system the read request is served by.

This is important because different nodes may have different versions of the data, and it's important to ensure that a read operation returns the latest version. There are different approaches to achieving read consistency in NoSQL databases, including strong consistency, eventual consistency, and tunable consistency.

-Strong consistency ensures that all nodes in the system have the same data at all times, but this can impact performance.
-Eventual consistency allows for better performance but may result in slightly out-of-date data.

-Tunable consistency provides a trade-off between consistency and performance by allowing the application to specify the level of consistency required for each read operation.

7. Explain the CAP Theorem.

CAP Theorem: The CAP (Consistency, Availability, Partition tolerance) Theorem is a fundamental concept in distributed systems that states it is impossible for a distributed data store to simultaneously guarantee all three aspects of consistency, availability, and partition tolerance.

Consistency: Consistency refers to ensuring that all nodes in a distributed system have the same view of the data at all times. In other words, if a write operation is successful, all subsequent read operations will return the updated data.

Availability: Availability means that the system continues to operate and respond to read and write requests even in the face of failures or network partitions. It ensures that users can always access the system and perform operations.

Partition Tolerance: Partition tolerance is the system's ability to function and tolerate network partitions or communication failures that may occur in a distributed environment. It ensures that the system can continue to operate even if some nodes are unable to communicate with each other.

The Trade-off: The CAP Theorem states that in the presence of a network partition (i.e., when nodes are unable to communicate), a distributed system must choose between consistency and availability. It means that when faced with a network partition, the system must decide whether to prioritize consistency (ensuring all nodes have the same view of the data) or availability (allowing continued read and write operations even if they may return stale or inconsistent data).

8. Write a note on Quorums.

Quorums: In distributed systems, quorums are a concept used to determine the minimum number of nodes or replicas that need to agree on an operation before it is considered successful or valid.

Consensus: Quorums are commonly used in scenarios where achieving consensus among distributed nodes is important, such as in distributed databases or distributed consensus protocols like Paxos or Raft.

Quorum Reads: In read operations, a quorum read requires that a certain number of nodes or replicas respond to the read request. The system collects the responses and returns the most recent version of the data that was returned by the required number of nodes.

Quorum Writes: In write operations, a quorum write requires that a certain number of nodes or replicas successfully acknowledge the write request. Once the required number of acknowledgments is received, the write is considered successful.

Quorum Size: The size of a quorum is determined by a combination of factors, such as the desired level of consistency, fault tolerance, and performance requirements.

9. Explain the Version Stamps.

Version Stamps: In distributed databases or systems, version stamps are used to assign unique identifiers or timestamps to data entries, enabling tracking and synchronization of updates across multiple nodes.

Unique Identifiers: Version stamps provide each data entry with a unique identifier or timestamp, allowing for precise tracking of changes and updates over time.

Synchronization: With version stamps, distributed nodes can determine the order of updates and resolve conflicts that may arise when multiple nodes attempt to modify the same data concurrently.

Concurrency Control: Version stamps are often used in concurrency control mechanisms, such as optimistic concurrency control, to ensure that concurrent updates to the same data are handled correctly and consistently.

Ordering and Consistency: By assigning version stamps to data entries, the system can establish a temporal ordering of updates, enabling consistency across distributed nodes and ensuring that all nodes eventually converge to a consistent state.

10. Write a note on Relaxing Durability.

Durability: In databases, durability refers to the property that ensures that once a transaction is committed, its changes are permanently stored and will survive any subsequent system failures or crashes.

Relaxing Durability: Relaxing durability is a concept in databases where the strict requirement of immediate and synchronous data persistence is relaxed to improve performance and throughput.

Delayed Durability: One approach to relaxing durability is to introduce delayed durability. Instead of immediately persisting changes to permanent storage, the database defers the write operations to disk, relying on faster in-memory caches or buffers to handle data modifications.

Asynchronous Commit: Another technique is asynchronous commit, where the commit acknowledgement is returned to the user before the changes are durably stored on disk. This allows the system to continue processing transactions without waiting for the costly disk write operations to complete.

Trade-offs: Relaxing durability provides performance benefits by reducing the I/O latency associated with synchronous disk writes. However, it introduces the risk of potential data loss if a failure occurs before the changes are durably stored. It's important to carefully consider the trade-offs and assess the level of durability required based on the specific application's needs and criticality of the data.