NOSQL Databases and Big Data Storage Systems

Introduction to Big Data

Big Data:

Big Data is like a huge amount of information. It's so big that regular tools (hardware devices) can't handle it. We need special tools and tricks to make sense of this massive amount of data.

Five Characteristics of Big Data:

- **Volume:** How much data we have collected.
- **Velocity:** How fast data is coming at high speed.
- **Variety:** Different types of data (like numbers, words, pictures, voice-records).
- **Veracity:** How much we can trust the data (accuracy, precision, integrity, reliability).
- **Value:** How useful the data is to make useful decisions.

Introduction to NOSQL

- **NOSQL**: Stands for "Not Only SQL."
- Designed for **distributed databases/storage systems**.
- Focuses on:
  - **Semi-structured data** storage.
  - **High performance**, **availability**, **data replication**, and **scalability**.

Applications of NOSQL:

NOSQL systems are widely used in:

- **Social media** (posts, tweets).
- **Web links** and **user profiles**.
- **Marketing and sales** data.
- **Road maps**, **spatial data**, and **emails**.

Categories of NOSQL Systems:

1. **Document-based systems**: Store data as JSON-like documents. **Examples**: MongoDB, CouchDB.
2. **Key-value stores**: Simple data structures with key-value pairs. **Examples**: DynamoDB.
3. **Column-based systems**: Optimized for queries over large datasets. **Examples**: BigTable, Cassandra.

4. **Graph-based systems**: Represent data as nodes and edges. **Examples**: Neo4J, GraphBase.
5. **Hybrid systems**: Combine features of multiple models. **Examples**: OrientDB.
6. **Object databases**: Store data as objects.
7. **XML databases**: Specialized for XML documents.

Key NOSQL Characteristics:

1. Distributed Databases:

- **Scalability**: Easily handles growing data.
- **Availability and replication**:
  - Models include **master-slave** and **master-master** replication.
- **Sharding**: Data is split into smaller, manageable parts.
- **Eventual consistency**: Updates are propagated asynchronously.

2. Data Models and Query Languages:

- **Schema-less**: No predefined structure is needed.
- Supports **less powerful query languages** than traditional databases.
- **Versioning**: Tracks changes to data over time.

The CAP Theorem:

- **Consistency**: Ensures that all nodes in a system have the same data at any given time.
- **Availability**: Every request to the system gets a response, even if some nodes are down.
- **Partition Tolerance**: The system still works even if there is a failure or delay in communication between nodes.

The **CAP Theorem** states that it's not possible to guarantee all three properties simultaneously in a distributed system with data replication.

- A system can choose any **two** out of the three: Consistency, Availability, and Partition Tolerance.
- Often, **weaker consistency** is acceptable, especially in **NoSQL** systems, where guaranteeing **availability** and **partition tolerance** is more important.
- **Eventual consistency** is commonly used, meaning the system will eventually become consistent but may not be immediately consistent at all times.

Master-Slave Replication:

- **Master-Slave Replication** involves one primary database (the "master") and one or more secondary databases (the "slaves").
- The **master** handles all the write operations (INSERT, UPDATE, DELETE), while the **slaves** only handle read operations.

- The **master** replicates its data to the **slaves**, ensuring that the slaves have the same data.
- **Advantages**:
  - Good for scaling read operations, as multiple slaves can handle read queries.
  - Easy to set up and manage.
- **Disadvantages**:
  - If the master fails, no new writes can happen unless a new master is promoted.
  - The master becomes a bottleneck for write-heavy systems.

Master-Master Replication:

- **Master-Master Replication** involves two or more databases that all act as **masters**.
- Each database can handle both **read** and **write** operations, and changes are replicated across all other masters.
- **Advantages**:
  - No single point of failure, as all nodes can handle reads and writes.
  - Better availability and fault tolerance.
- **Disadvantages**:
  - Conflicts can occur when different masters try to update the same data simultaneously (e.g., data inconsistency).
  - More complex to set up and manage, especially when resolving conflicts.