Vidyalankar Institute of Technology Accredited A+ by NAAC	**DEPARTMENT OF COMPUTER ENGINEERING**

Activity No. 02

Semester	B.E. Semester VII – Computer Engineering
Subject	Big Data Analysis
Subject Professor In-charge	Prof. Pankaj Vanvari
Academic Year	2024-25

Student Name	Deep Salunkhe
Roll Number	21102A0014

CAP Theorem and Its Takeaway

CAP Theorem (Consistency, Availability, Partition Tolerance)

The CAP theorem, introduced by Eric Brewer, states that in a distributed data store, only two out of the following three guarantees can be achieved simultaneously:

1. **Consistency**: Every read receives the most recent write or an error.
2. **Availability**: Every request (read or write) receives a response, without guarantee that it contains the most recent write.
3. **Partition Tolerance**: The system continues to operate despite arbitrary partitioning due to network failures.

Takeaway

- **Trade-offs**: When designing distributed systems, trade-offs between consistency, availability, and partition tolerance are inevitable. You cannot achieve all three at the same time.
- **System Design Choices**: The choice between consistency and availability depends on the specific requirements of your application.
  - Systems like HBase, MongoDB (NoSQL) often choose availability and partition tolerance (AP systems).
  - Systems like traditional RDBMS (SQL) often choose consistency and partition tolerance (CP systems).

RDBMS vs. HADOOP

RDBMS (Relational Database Management System)

When Preferred:

- **Structured Data**: Ideal for structured data with predefined schema.
- **ACID Transactions**: Requires strong consistency and transactional support (Atomicity, Consistency, Isolation, Durability).
- **Complex Queries**: Supports complex queries and joins.
- **Data Integrity**: Ensures high data integrity through constraints and relationships.

Use Cases:

- Banking systems.
- E-commerce applications.
- Enterprise resource planning (ERP) systems.

HADOOP (and similar distributed storage systems)

When Preferred:

- **Unstructured and Semi-structured Data**: Efficiently handles unstructured or semi-structured data like logs, videos, and images.
- **Big Data**: Designed for processing and storing large volumes of data across distributed systems.
- **Scalability**: Highly scalable and can handle petabytes of data across clusters of machines.
- **Batch Processing**: Suitable for batch processing and large-scale data analysis.

Use Cases:

- Data warehousing.
- Large-scale data analytics.
- Processing log files and sensor data.

Strong Consistency vs. Eventual Consistency

Strong Consistency

- **Definition**: After a write is completed, any subsequent read will return the most recent write.
- **Use Case**: Critical applications where the latest data is required, such as banking transactions.

Eventual Consistency

- **Definition**: After a write, reads may return stale data for a period, but eventually, all nodes will converge to the latest data.
- **Use Case**: Applications where high availability is more critical than having the most recent data, such as social media feeds.

SQL vs. NoSQL: Applications and Case Studies

SQL (Relational Databases)

Importance:

- **Structured Data**: Best for structured data with predefined schemas.
- **Data Integrity**: Enforces data integrity through constraints.
- **Transactional Support**: Ensures ACID properties, making it suitable for financial transactions and critical applications.

Applications/Case Studies:

- **Banking Systems**: SQL databases ensure transactional integrity and consistency.
- **E-commerce Platforms**: Use SQL databases for managing inventory, user accounts, and transactions.
- **Healthcare Systems**: Manage patient records and hospital administration with SQL databases.

NoSQL (Non-Relational Databases)

Importance:

- **Scalability**: Horizontally scalable, handling large volumes of unstructured data.
- **Flexibility**: Schema-less design allows for flexible and rapid changes in data structure.

- **High Availability**: Designed for high availability and partition tolerance.

<u>Importance of SQL and NoSQL: Applications and Case Studies</u>

SQL (Relational Databases)

Importance

1. **Structured Data Management**: SQL databases are ideal for managing structured data with predefined schemas. They enforce data integrity through constraints and relationships, ensuring that data adheres to specific formats and rules.
2. **ACID Transactions**: SQL databases support Atomicity, Consistency, Isolation, and Durability (ACID) transactions. This ensures reliable transaction processing and data integrity, which is crucial for applications where precise data accuracy is required.
3. **Complex Queries**: SQL databases support complex queries and joins, allowing for sophisticated data analysis and reporting. This makes them suitable for applications that require detailed data insights.
4. **Data Integrity**: Through the use of primary keys, foreign keys, and other constraints, SQL databases maintain high levels of data integrity. This is essential for applications that depend on accurate and reliable data.

Applications/Case Studies

1. **Banking Systems**
   - **Use Case**: SQL databases are used to manage customer accounts, transactions, loans, and other financial operations.
   - **Example**: A major bank uses an SQL database to handle millions of transactions per day, ensuring each transaction is accurately recorded and retrievable. The strong consistency guarantees of SQL databases ensure that account balances are always correct after transactions.
2. **E-commerce Platforms**
   - **Use Case**: SQL databases manage product inventories, customer data, orders, and payment processing.
   - **Example**: An online retailer like Amazon uses SQL databases to keep track of inventory levels, manage customer orders, and process payments. The ability to

perform complex joins and transactions ensures that customers receive accurate information about product availability and order statuses.

3. **Healthcare Systems**
    - **Use Case**: SQL databases are used to manage patient records, appointment schedules, billing, and clinical data.
    - **Example**: A hospital uses an SQL database to store patient records, including medical history, lab results, and treatment plans. The ACID properties of SQL databases ensure that patient information is always accurate and up-to-date, which is critical for providing quality healthcare.

NoSQL (Non-Relational Databases)

Importance

1. **Scalability**: NoSQL databases are designed to scale horizontally, allowing them to handle large volumes of data across distributed systems. This makes them suitable for applications with rapidly growing data and high traffic.
2. **Flexibility**: NoSQL databases often have a schema-less design, which allows for flexible and rapid changes in data structure. This is useful for applications where the data model evolves over time.
3. **High Availability and Partition Tolerance**: NoSQL databases are designed to be highly available and partition-tolerant, making them suitable for applications that require continuous availability even in the presence of network partitions.
4. **Handling Unstructured Data**: NoSQL databases are well-suited for storing and managing unstructured or semi-structured data, such as logs, videos, and social media posts.

Applications/Case Studies

1. **Social Media Platforms**
    - **Use Case**: NoSQL databases store user profiles, posts, comments, likes, and other social interactions.
    - **Example**: Facebook uses Cassandra, a NoSQL database, to handle billions of read and write requests per second across its distributed system. The flexibility and scalability of NoSQL allow Facebook to efficiently manage vast amounts of unstructured data generated by users.
2. **Real-time Analytics**

- **Use Case**: NoSQL databases are used for real-time data processing and analytics.
- **Example**: Twitter uses a combination of Hadoop and NoSQL databases like HBase to process and analyze real-time data streams. This allows Twitter to provide real-time search and trend analysis, delivering timely insights to users and advertisers.

3. **Content Management Systems (CMS)**
   - **Use Case**: NoSQL databases manage diverse content types, including text, images, and videos.
   - **Example**: The New York Times uses MongoDB, a NoSQL database, to manage its content management system. The flexibility of MongoDB allows the newspaper to store various content types and easily scale as the volume of digital content grows.