Module -1

Introduction to Big Data

Big Data refers to datasets so large and complex that they cannot be processed using traditional data processing techniques. As data grows exponentially, especially with the rise of digital technology, the volume, velocity, and variety of data have exceeded the capabilities of conventional systems.

Understanding the Scale:

- **Small-scale data:** Typically measured in megabytes (MB) or gigabytes (GB), such as Word documents or movies.
- **Big Data scale:** Data measured in petabytes (PB, 10^15 bytes) or even exabytes (10^18 bytes), requiring specialized systems to store, process, and analyze.

Sources of Big Data:

1. **Social Networking Sites:** Platforms like Facebook, Twitter, and Instagram generate massive amounts of user interaction data in real-time.
2. **E-Commerce Sites:** Websites like Amazon and Flipkart track user behavior, transactions, and product trends.
3. **Weather Stations:** Data from weather sensors, satellites, and meteorological stations.
4. **Telecom Companies:** Large volumes of data from call records, internet usage, and customer transactions.

Challenges of Big Data

1. Volume:

- Big Data refers to massive datasets that grow continuously. Traditional storage and database management systems (DBMS) struggle with the sheer amount of data, leading to problems in storage and retrieval.

2. Velocity:

- The speed at which data is generated is high, especially from sources like social media, sensors, and transaction logs. Traditional data processing systems often cannot handle real-time data streams effectively.

3. Variety:

- Big Data consists of different formats, including:
    - **Structured data:** Databases (rows/columns)
    - **Unstructured data:** Text files, images, videos, social media posts

- **Semi-structured data:** JSON, XML
- Managing and integrating these diverse types of data presents a major challenge.

4. Veracity:

- Big Data often includes noise or inaccuracies, which makes it important to verify the quality and reliability of the data.

5. Value:

- The ultimate goal is to extract meaningful insights from large datasets. The value lies in the insights, trends, and patterns discovered, which can drive business decision-making.

Solutions to Big Data Challenges

1. Storage:

- **HDFS (Hadoop Distributed File System)** is used for distributed storage. It breaks large files into smaller blocks and distributes them across a cluster of commodity hardware, ensuring fault tolerance and scalability.
  - **Commodity hardware:** Inexpensive and widely available hardware used to build clusters.
  - **Write once, read many times:** HDFS stores data by writing it once, and then reading it many times, which is efficient for batch processing.

2. Processing:

- **MapReduce** is a programming model for processing large datasets by dividing them into smaller tasks that can be processed in parallel across multiple nodes.

3. Analysis:

- **Pig and Hive** are tools that simplify querying and analyzing large datasets. Pig uses a high-level scripting language, while Hive uses SQL-like queries to analyze data.

4. Cost:

- **Hadoop** is an open-source platform, which means companies can implement Big Data solutions without the high cost of proprietary software.

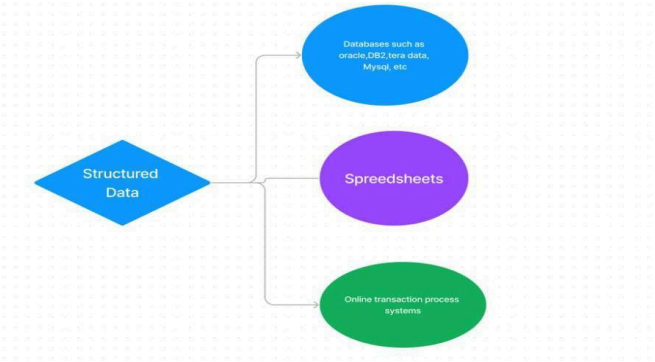
Characteristics of Big Data (The 5 Vs)

1. **Volume:** Refers to the size of the data. Big Data involves petabytes or even exabytes of data.

2. **Velocity:** The speed at which new data is generated. Real-time data streams from IoT devices, social media, and sensors require rapid processing.
3. **Variety:** The different types of data (structured, unstructured, semi-structured). Each format requires different tools for analysis.
4. **Veracity:** Refers to the uncertainty or trustworthiness of the data. With noisy or inconsistent data, verifying the quality becomes crucial.
5. **Value:** The potential insights and business value extracted from analyzing large datasets. Tools like machine learning, predictive analytics, and data mining are used to extract value from Big Data.

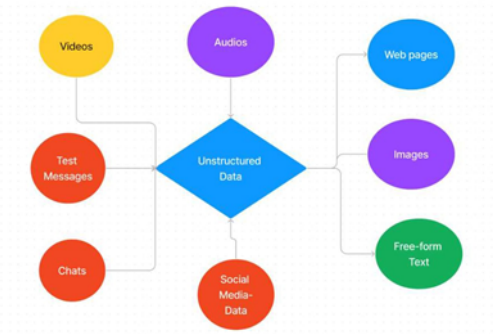
Types of Big Data

1. **Structured Data:**



- **Definition:** Data organized in a fixed format (e.g., databases with rows and columns) with a predefined schema.
- **Characteristics:** High organization and easy retrieval. Stored in relational databases such as MySQL or Oracle.
- **Examples:** Spreadsheets, financial records, transactional data.

2. Unstructured Data:



- **Definition:** Data without a predefined structure, difficult to analyze using traditional tools.

- **Characteristics:** Includes text, images, audio, video, making it harder to search and process.
- **Examples:** Social media feeds, emails, images, customer reviews.

3. Semi-Structured Data:

- **Definition:** Lies between structured and unstructured data. While not organized into relational databases, it contains tags or markers that separate elements (e.g., key-value pairs).
- **Characteristics:** Does not conform to a strict schema but has organizational elements that make data handling easier.
- **Examples:** XML files, JSON documents, log files, NoSQL databases.

Traditional Data vs. Big Data

Traditional Data vs. Big Data

Traditional Data	Big Data
Small in size, manageable with traditional methods.	Large and complex, unmanageable with traditional methods.
Mostly structured and stored in spreadsheets/databases.	Includes structured, semi-structured, and unstructured data.
Data collection is often manual.	Information is collected automatically via systems.
Data is sourced internally.	Data is sourced from various external sources like social media.
Consists of simple, structured data like customer info.	Includes complex data like images, videos.
Analyzed with basic statistical methods.	Analyzed using advanced methods like machine learning.
Processing is slower.	Processing is fast and real-time.
Limited insights and value.	Provides in-depth insights for better decision-making.
Easier to secure and protect.	More challenging to secure due to its size and complexity.
Less time and cost for storage.	Requires more time and resources for storage.
Can be managed on a single system.	Requires distributed storage across many systems.

Applications of Big Data

1. Agriculture

1. **Precision Agriculture**: Big data analytics can help farmers optimize crop yields by analyzing data from sensors, drones, and satellites. It enables precise decisions about planting, irrigation, and fertilization.
2. **Livestock Management**: Monitoring data on animal health and behavior can improve livestock management. Wearable sensors and data analytics can provide insights into animal health and breeding patterns.
3. **Supply Chain Optimization**: Big data can help optimize the agricultural supply chain by tracking and managing the movement of crops and livestock from farm to market, reducing waste and improving efficiency.

2. Healthcare

1. **Disease Surveillance**: Big data can be used for real-time monitoring of disease outbreaks and tracking the spread of infectious diseases, helping healthcare authorities respond more effectively.
2. **Clinical Decision Support**: Electronic health records (EHRs) and patient data analysis can assist healthcare professionals in making more informed diagnoses and treatment decisions.
3. **Drug Discovery**: Big data analytics can accelerate drug discovery by analyzing vast datasets, identifying potential drug candidates, and predicting their effectiveness.

3. Travel & Tourism

1. **Personalized Recommendations**: Big data is used to analyze traveler preferences and behaviors, enabling travel companies to provide personalized recommendations for destinations, accommodations, and activities.
2. **Demand Forecasting**: Travel companies use data to predict peak travel times, allowing them to adjust pricing, availability, and staffing accordingly.
3. **Customer Experience Enhancement**: Airlines and hotels use big data to improve the overall customer experience by analyzing feedback and operational data to make improvements.

4. Finance

1. **Risk Management**: Big data analytics is crucial in financial institutions for assessing and managing risks by analyzing historical and real-time data to detect fraud, assess credit risk, and monitor market conditions.
2. **Algorithmic Trading**: High-frequency trading firms use big data to make split-second trading decisions by analyzing market data and trends.
3. **Customer Insights**: Financial institutions use data analytics to understand customer behavior and preferences, offering customized financial products and services.

5. Retail & E-Commerce

1. **Inventory Management**: Big data helps retailers optimize inventory levels, reducing overstock and stockouts through predictive analytics.
2. **Personalized Marketing**: E-commerce companies use big data to provide personalized product recommendations and targeted marketing campaigns based on customer browsing and purchase history.
3. **Price Optimization**: Dynamic pricing strategies in e-commerce adjust prices in real-time based on demand, competition, and other factors, optimizing revenue.

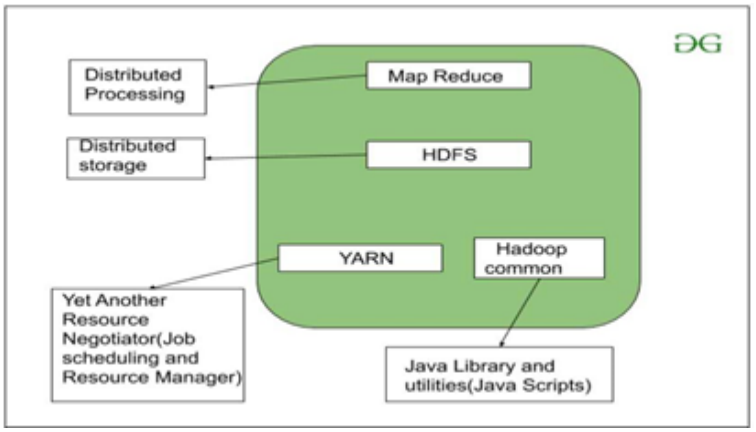
Hadoop Architecture

Hadoop is an open-source framework from Apache that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from a single server to thousands of machines, each offering local computation and storage. Hadoop is written in Java and is not an OLAP (online analytical processing) system.

Key features:

- **Scalable:** Hadoop can handle increasing amounts of data by adding more nodes to the cluster.
- **Reliable:** It handles hardware failures automatically by replicating data across different nodes.
- **Cost-effective:** As it uses commodity hardware and open-source software, it's cost-efficient.

Today, many companies like **Facebook, Yahoo, Google, Twitter, LinkedIn**, and more use Hadoop to handle their big data needs.



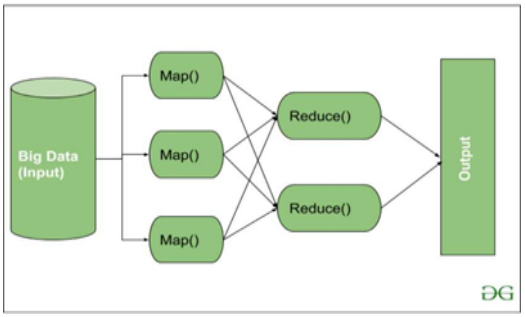
Main Components of Hadoop:

1. **MapReduce**
2. **HDFS (Hadoop Distributed File System)**
3. **YARN (Yet Another Resource Negotiator)**
4. **Hadoop Common**

---

1. MapReduce:

MapReduce is a programming model and processing technique used to handle and process large amounts of data in a distributed manner. It breaks the job into smaller tasks that can be processed in parallel across multiple nodes in the cluster.



- **Parallel Processing:** MapReduce splits the input data into smaller, manageable chunks and processes them in parallel, significantly speeding up the execution time.
- **Phases of MapReduce:**
    - **Map Phase:** The input data is provided to the Map() function, which processes the data and breaks it into key-value pairs.
    - **Shuffle & Sort:** The key-value pairs are shuffled, sorted, and passed to the Reduce() phase.
    - **Reduce Phase:** The Reduce() function combines the results based on the keys and generates the final output.

Example:

- **Input:** List of words.
- **Map Phase:** Breaks the input into tuples (key-value pairs), such as ("apple", 1), ("banana", 1).
- **Reduce Phase:** Combines the tuples based on keys, e.g., sums the counts of each word, resulting in ("apple", 3), ("banana", 2).

---

2. HDFS (Hadoop Distributed File System):

HDFS is the primary storage system in Hadoop, designed to store large files by breaking them into smaller blocks.

- **Fault Tolerance:** HDFS stores multiple copies (replicas) of each block across different nodes to ensure data availability even if a node fails.
- **Master-Slave Architecture:**
    - **NameNode (Master):** Manages the metadata (such as block locations) and regulates file access.
    - **DataNodes (Slaves):** Store the actual data blocks and send periodic heartbeats to the NameNode.

Example:

- A 2GB file is divided into smaller blocks (e.g., 128MB each), and these blocks are distributed across different DataNodes in the cluster. Even if one DataNode fails, the file can be reconstructed from other nodes.

---

3. YARN (Yet Another Resource Negotiator):

YARN is responsible for resource management and job scheduling in Hadoop. It acts as the central point for managing and monitoring workloads.

- **Job Scheduling:** YARN breaks large tasks into smaller jobs, which are then distributed to different nodes in the cluster. It ensures efficient utilization of cluster resources.
- **Resource Management:** It keeps track of the available resources (memory, CPU, etc.) and allocates them based on the task's requirements.

Example:

- If there is a big data processing task, YARN will divide the task into smaller jobs and schedule them across different nodes for parallel execution, ensuring efficient use of resources.

---

4. Hadoop Common:

Hadoop Common provides a set of utilities and libraries that support other Hadoop modules, including HDFS, YARN, and MapReduce.

- **Java Libraries:** These are required by all components to function correctly in a Hadoop environment. They provide essential features such as file system abstractions and I/O utilities.

Example:

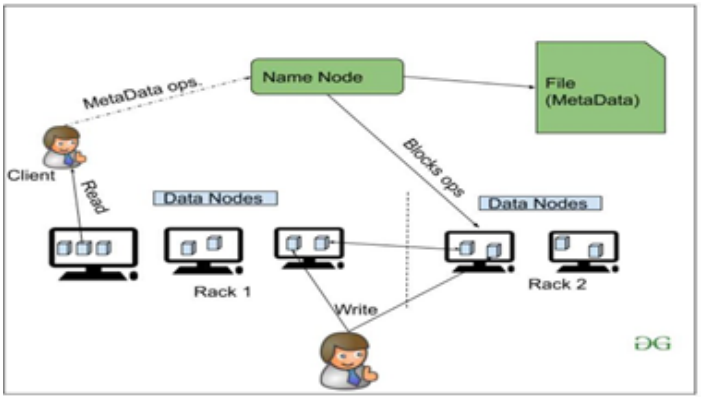
- Hadoop Common is used by HDFS to handle I/O operations and by YARN and MapReduce for inter-process communication.

Module – 2

Hadoop Distributed File System (HDFS)

HDFS is the primary storage system for Hadoop. It is designed to store large datasets across a cluster of computers. HDFS follows a **master/slave architecture** and is fault-tolerant, scalable, and reliable.

Architecture:



- **HDFS** is an open-source framework for distributed data processing.
- **Storage of Large Datasets:** HDFS stores large files by breaking them into smaller blocks (default size 128MB) and distributing these blocks across multiple nodes.
- **Fault Tolerance:** Data is replicated across different nodes, ensuring high availability.
- **Master-Slave Architecture:** The system is built on the concept of a single **NameNode** (master) and multiple **DataNodes** (slaves).

Components of HDFS:

NameNode (Master):

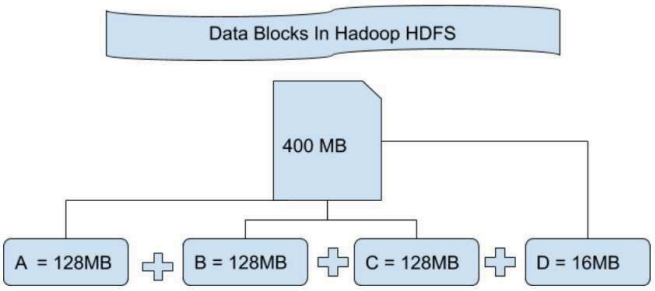
- **Role:** NameNode acts as the master in the HDFS architecture and manages the metadata (data about data).
- **Metadata:** Includes details like file names, file sizes, and the location of file blocks on DataNodes. It keeps a record of users' activities in the Hadoop cluster.
- **Operations:** NameNode instructs DataNodes to perform actions like **create, delete, and replicate** blocks of data.

DataNodes (Slaves):

- **Role:** DataNodes store the actual data and communicate with the NameNode.

- **Storage Capacity:** The more DataNodes in the cluster, the more data can be stored. Typically, large storage capacity is required to store large numbers of file blocks.
- **Number of DataNodes:** Hadoop can have 1 to over 500 DataNodes, depending on the system's capacity.

File Blocks in HDFS:

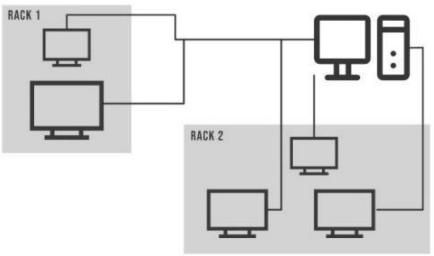


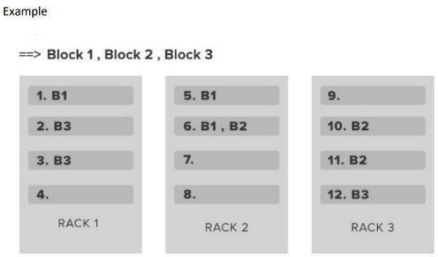
- **Data Storage:** Data in HDFS is stored in the form of blocks. A large file is divided into multiple blocks of default size 128MB (can be modified).
    - Example: A 400MB file would be split into four blocks: three 128MB blocks and one 16MB block.
- **HDFS Characteristics:** HDFS doesn't care about the content of blocks; it only ensures that data is divided and stored across nodes.

Replication in HDFS:

- **Definition:** Replication is the process of making copies of file blocks for fault tolerance.
- **Default Factor:** By default, HDFS creates 3 replicas of each file block for backup purposes, which can be changed.
    - Example: If you have 4 blocks, HDFS will create 3 replicas of each block, totaling 12 blocks (4 original + 8 replicas).

Rack Awareness in HDFS:





- **Rack:** A physical collection of nodes in the Hadoop cluster.
- **Rack Awareness:** NameNode uses the information about racks to select the closest DataNodes for storing replicas, improving performance by reducing network traffic.
- **Replica Placement:**
    - At least one replica is placed on a different rack.
    - No more than two replicas of a block are stored on the same rack.
    - Number of racks should be fewer than the number of replicas to ensure redundancy.

Benefits:

- **Data Safety:** Ensures data is stored across different racks, reducing the risk of data loss.
- **Improved Performance:** Optimizes network bandwidth and speeds up read/write operations.

HDFS commands:

1. **hadoop fs -ls**
    - **Description:** Lists the contents of a directory in HDFS.
    - **Example:**

bash Copy code

```
hadoop fs -ls /user/hadoop
```

2. **hadoop fs -mkdir**
   - **Description:** Creates a new directory in HDFS.
   - **Example:**

```bash
hadoop fs -mkdir /user/hadoop/new_directory
```

3. **hadoop fs -put**
   - **Description:** Copies files or directories from the local file system to HDFS.
   - **Example:**

```bash
hadoop fs -put localfile.txt /user/hadoop/hdfsfile.txt
```

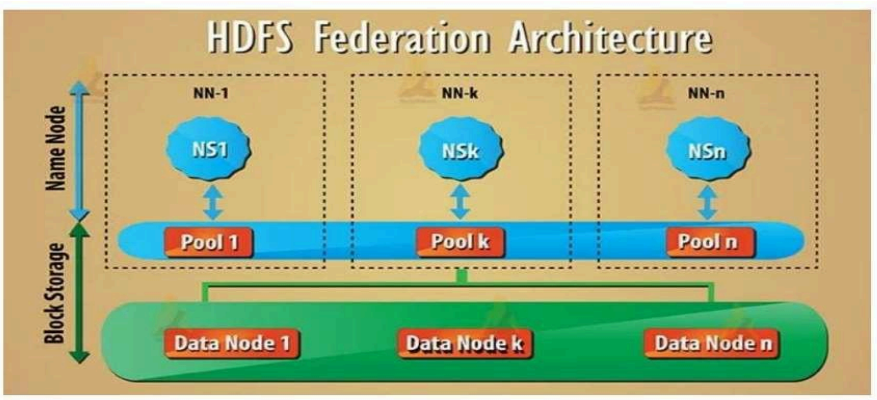
4. **hadoop fs -get**
   - **Description:** Copies files or directories from HDFS to the local file system.
   - **Example:**

```bash
hadoop fs -get /user/hadoop/hdfsfile.txt localfile.txt
```

5. **hadoop fs -rm**
   - **Description:** Deletes files or directories from HDFS.
   - **Example:**

```bash
hadoop fs -rm /user/hadoop/hdfsfile.txt
```

HDFS Federation:



- **Improvement:** HDFS Federation enhances the original HDFS architecture by using multiple independent **NameNodes**, overcoming the limitation of having only one NameNode.
- **Previous Limitation:** In the original HDFS, a single NameNode managed the entire namespace. If the NameNode failed, the whole cluster became unavailable until it was restarted.

- **Advantage:** Federation scales the system horizontally by allowing multiple NameNodes to handle different namespaces. DataNodes are shared across these NameNodes.

Benefits of HDFS Federation:

1. **Isolation:** Each namespace is independent, with its own file system hierarchy.
2. **Improved Scalability:** Manages multiple namespaces, distributing the load across NameNodes, making it easier to handle large files.
3. **Increased Throughput:** Allows for the handling of larger volumes of data, improving performance.

MapReduce

MapReduce is a **data processing tool** used to process data in parallel across multiple nodes in a distributed manner.

Phases of MapReduce:

1. **Map Phase:**
    - **Input:** Takes data as a key-value pair.
    - **Output:** Provides a list of key-value pairs, which are not necessarily unique.
2. **Shuffle and Sort:**
    - The Map output is shuffled and sorted based on the keys, ensuring that unique keys are sent to the Reducer.
3. **Reduce Phase:**
    - **Input:** Takes the shuffled and sorted data as input.
    - **Output:** Produces the final aggregated output.

Process Flow:

Input --> Map --> Shuffle and Sort --> Reduce --> Final Output

---

Map Task Components:

1. **Record Reader:**
    - Breaks the input into key-value pairs and passes them to the Map() function.
    - **Key:** Represents the locational information, and **Value** is the data associated with it.

2. **Map Function:**
    - Processes the input data and generates intermediate key-value pairs.
3. **Combiner:**
    - Acts as a local reducer, combining the intermediate data from the Map() function to reduce the amount of data transferred to the reducer.
4. **Partitioner:**
    - Responsible for dividing the data based on the hash of the key. This ensures that all the data with the same key goes to the same reducer.

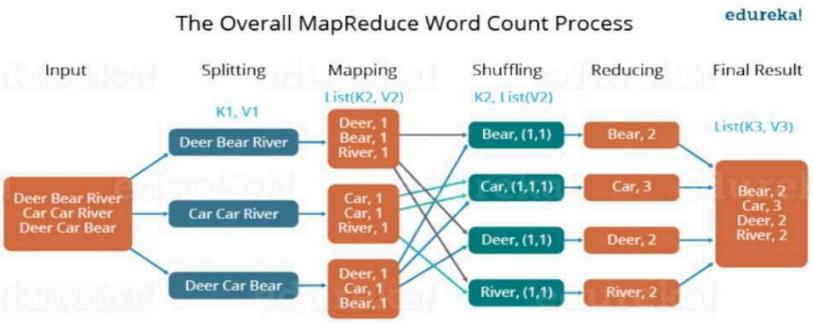
Reduce Task Components:

1. **Shuffle and Sort:**
    - Sorts the mapper output and partitions the data for the reducer. All the values for each unique key are collected and passed to the reducer.
2. **Reduce:**
    - Gathers the tuples generated from the Map() function and performs operations like sorting or aggregation to generate the final result.
3. **Output Format:**
    - Writes the final key-value pairs to a file using the record writer.

Partitioner and Combiner:

- **Combiner:**
    - Acts as a mini-reducer to summarize the map output records before they are sent to the reducer. The **combiner** class implements the same reduce function as the reducer but works locally to reduce the amount of data transferred between the map and reduce phases.
- **Partitioner:**
    - Partitions the data based on a user-defined condition, which works like a hash function. The total number of partitions must equal the number of reducers.

Word Count Example Using MapReduce:



1. **Input:** A large text file containing words.
2. **Map:** Breaks the text into words and creates key-value pairs like ("word", 1).
3. **Shuffle and Sort:** Groups the key-value pairs based on the key (i.e., the word) and prepares them for the reducer.
4. **Reduce:** Counts the number of occurrences for each word and outputs the result.

Module – 3

NoSQL Introduction:

NoSQL stands for "Not Only SQL" and is a type of database designed to handle a wide variety of data formats, offering flexibility in data storage and retrieval, especially for big data. Traditional relational databases (SQL-based) focus on structured data in tables, but NoSQL databases accommodate **unstructured, semi-structured, and structured data**, making them highly adaptable o modern applications.

NoSQL Business Drivers:

1. Volume: Dealing with Big Data
Explanation:
Traditional relational databases often struggle to manage and process large volumes of data efficiently due to their reliance on a single server or a few high-capacity servers. NoSQL databases are designed to scale horizontally by distributing data across multiple servers, enabling them to handle large datasets and high throughput with greater efficiency.

- **Scaling:** NoSQL databases use distributed architectures that allow for the addition of more servers (nodes) as the amount of data grows, rather than relying on a single, powerful server.
- **Data Distribution:** Data is partitioned across multiple nodes, which ensures that no single node becomes a bottleneck, allowing for better performance and reliability.

Example:
A company like Facebook handles massive amounts of user-generated content and interactions. Using a NoSQL database, they can distribute this data across many servers, ensuring fast access and processing times even as their user base grows.

---

2. Velocity: Handling Data in Real-Time
Explanation:
Real-time data processing is critical for many applications that require immediate responses or updates. Traditional databases might face performance issues under high-load conditions, whereas NoSQL databases are built to handle real-time data with low latency.

- **Data Ingestion:** NoSQL databases can handle high-speed data ingestion from multiple sources simultaneously, which is crucial for applications like financial trading platforms or live-streaming services.
- **Speed:** They optimize data storage and retrieval processes to ensure quick access and processing, even when the volume of concurrent users or transactions is very high.

Example:
During major online shopping events (like Black Friday), a retail site using a NoSQL

database can handle thousands of transactions per second, ensuring that all customer actions are processed in real-time without delays.

---

3. Variability: Managing Different Data Types
Explanation:
NoSQL databases support various data models that allow for flexible data storage and retrieval. This flexibility is especially important for modern applications that handle diverse and evolving types of data, which may not fit neatly into a traditional relational database schema.

- **Schema Flexibility:** NoSQL databases do not require a fixed schema, making it easy to store and manage semi-structured or unstructured data without extensive reconfiguration.
- **Adaptability:** Changes to data formats or structures can be accommodated without significant downtime or redesign.

Example:
A content management system (CMS) that manages articles, multimedia content, and user comments can use a document-oriented NoSQL database to store and retrieve diverse types of content efficiently.

---

4. Agility: Quick Adaptation to Changes
Explanation:
Business environments and application requirements can change rapidly. NoSQL databases provide the agility needed to adapt to these changes with minimal disruption.

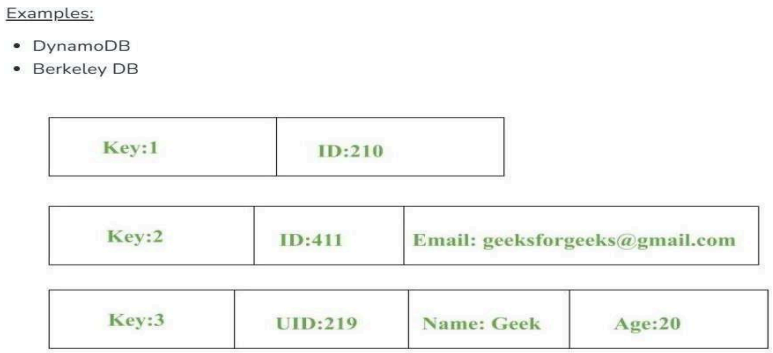
- **Schema Evolution:** You can add new fields or change data structures on-the-fly without major changes to the underlying database schema.
- **Development Speed:** Faster development and deployment cycles are possible because developers can iterate quickly without worrying about rigid schema constraints.

Example:
A startup developing a new feature for their app can deploy updates to the NoSQL database schema without downtime, enabling them to respond quickly to user feedback and market demands.

NoSQL Data Architecture Patterns:

1. Key-Value Store Database
Explanation:
Key-value stores are one of the simplest NoSQL database models, where data is stored as a collection of key-value pairs. Each key is unique, and the corresponding value can be any type of data, such as a string, number, or binary object.

- **Data Model:** The key serves as a unique identifier, and the value can be anything from a simple string to a complex JSON object.
- **Use Cases:** Ideal for applications requiring high-speed data retrieval by key, such as session management or caching.



Example:
Redis is a popular key-value store used for caching web application data to speed up response times.

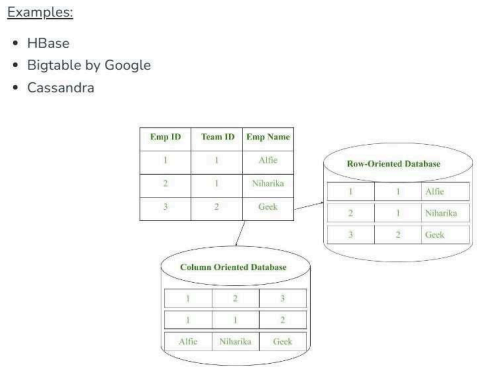
Advantages:

- Simple and fast access to data using keys.
- Scales horizontally by adding more nodes.

---

2. Column Store Database
Explanation:
Column family stores organize data by columns rather than rows. This structure allows for efficient querying and storage, especially for analytics and data warehousing applications where columnar access patterns are prevalent.

- **Data Model:** Data is stored in columns, with each column family containing a set of related columns. Each column can have its own data type and format.
- **Use Cases:** Suitable for applications with heavy read and write operations on specific columns, such as time-series data or large-scale data analytics.



Example:
Apache Cassandra is a widely-used column store database that excels in handling high-throughput data writes and reads.

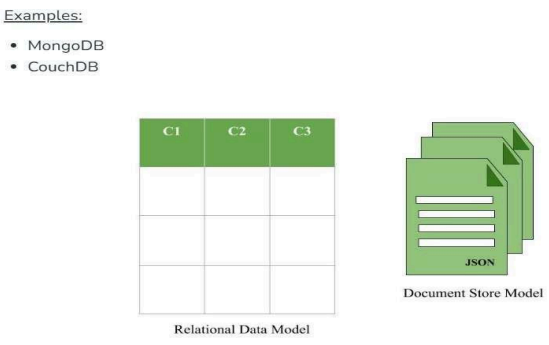
Advantages:

- Efficient for aggregate queries like SUM and COUNT.
- Handles large volumes of data and high-speed reads/writes.

---

3. Document Store Database
Explanation:
Document stores manage data as documents, which are typically stored in JSON, XML, or BSON formats. Documents are semi-structured and can contain nested data, allowing for flexible and complex data models.

- **Data Model:** Each document is self-contained and can include nested structures and various data types.
- **Use Cases:** Well-suited for applications with complex data structures, such as content management systems and e-commerce catalogs.



Example:
MongoDB is a leading document store database known for its flexible schema and ease of use with JSON-like documents.

Advantages:

- Flexible schema for semi-structured data.

- Easy to manage and retrieve complex data structures.

---

4. Graph Database
Explanation:
Graph databases are designed to store and query data represented as graphs. They use nodes to represent entities and edges to represent relationships between those entities.

- **Data Model:** Nodes represent entities (e.g., users, products), and edges represent relationships (e.g., friendships, purchases). This model is highly effective for querying complex relationships.
- **Use Cases:** Ideal for applications involving intricate relationships, such as social networks, recommendation engines, and fraud detection systems.

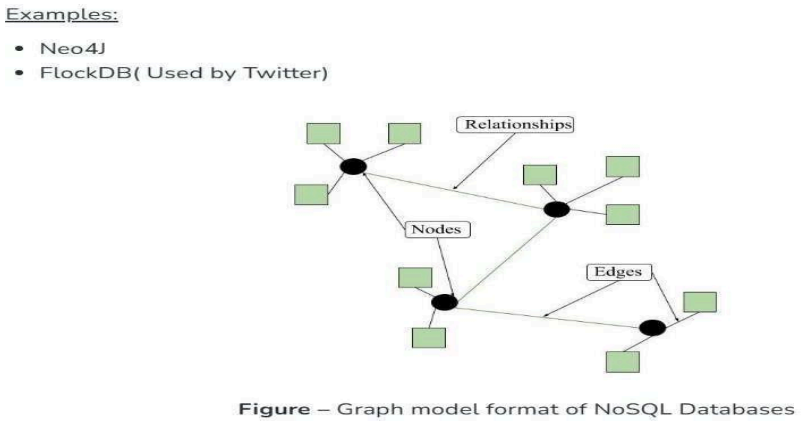


Figure – Graph model format of NoSQL Databases

Example:
Neo4j is a popular graph database used for managing and querying interconnected data.

Advantages:

- Efficiently handles complex queries and traversals.
- Provides insights into relationships and connections between data points.

Managing Big Data with NoSQL:

Shared-Nothing Architecture
Explanation:
In a shared-nothing architecture, each node operates independently with its own resources (CPU, memory, storage), and nodes do not share data or processing tasks. This design helps in scaling systems horizontally by adding more nodes without affecting the existing ones.

- **Scalability:** Adding more nodes increases the system's capacity to handle data and requests.

- **Fault Tolerance:** Failure of one node does not impact the others, as there is no shared resource.

Benefits:

- High scalability and performance.
- Improved fault tolerance and reliability.

Example:
Many cloud-based NoSQL databases, like Amazon DynamoDB, use a shared-nothing architecture to achieve high scalability and resilience.

Master-Slave vs. Peer-to-Peer Distribution Models

Master-Slave Architecture:

Explanation:
In a master-slave architecture, one node (the master) is responsible for coordinating operations, while the other nodes (slaves) perform tasks based on the master's instructions. The master node handles read and write operations, and the slaves replicate data from the master to ensure redundancy.

- **Master Role:** Manages data and controls all write operations.
- **Slave Role:** Replicates data from the master and handles read operations.

Advantages:

- Simple to implement and manage.
- Clear division of responsibilities, making it easier to understand and troubleshoot.

Disadvantages:

- The master can become a bottleneck for write operations.
- Single point of failure; if the master fails, the system may become unavailable until a new master is elected.

Example:
MySQL's replication setup often uses a master-slave model where the master node handles all write operations, and the slaves replicate the data for backup and read operations.

Peer-to-Peer Architecture:

Explanation:
In a peer-to-peer (P2P) architecture, all nodes (peers) are equal and can perform any role within the system. There is no central authority, and nodes communicate directly with each other to share data and tasks.

- **Node Equality:** Each node can act as both a client and a server, sharing resources and responsibilities.
- **Data Distribution:** Data is distributed across all nodes, and each node contributes to the overall system's functionality.

Advantages:

- No single point of failure, improving system reliability.
- Better load balancing and scalability, as each node shares the load.

Disadvantages:

- More complex to implement and manage.
- Increased network communication overhead and potential consistency issues.

Example:
Distributed file systems like BitTorrent use a peer-to-peer architecture where every node can download and upload data, making the system highly resilient and scalable.

Characteristic	Master-Slave Architecture	Peer-to-Peer Architecture
Control and Decision-Making	One designated master node controls and decides.	All nodes (peers) have equal status and can communicate directly.
Communication	Master communicates with and directs slaves.	Peers can communicate with any other peer without a central controller.
Responsibilities	Master manages coordination, task distribution.	Peers share responsibilities; no central authority.
Scalability	Scalability can be a challenge due to a potential bottleneck at the master.	Generally more scalable as there is no single point of control. Adding peers increases overall capacity.
Examples	Database systems (write operations by master, read by slaves), distributed systems (task distribution).	File-sharing networks (BitTorrent), some blockchain networks (Bitcoin).

HBase Overview

HBase is a distributed, scalable, and high-performance database built on top of the Hadoop Distributed File System (HDFS). It is part of the Hadoop ecosystem and designed for handling large amounts of sparse data efficiently. Unlike traditional relational databases, HBase is a NoSQL database that follows a column-oriented approach, making it suitable for applications that require real-time read/write access to large datasets.

Key Features of HBase:

- **Scalability**: HBase can scale horizontally by adding more servers to handle increased data loads.

- **Fault Tolerance**: It is designed to handle hardware failures gracefully, leveraging HDFS for data storage and replication.
- **Real-time Access**: Provides low-latency access to large volumes of data, suitable for applications that require fast access and real-time analytics.

HBase Data Model

To better understand it, let us take an example and consider the table below.

Customer ID	Name	Address	Product ID	Product Name
1	Paul Walker	US	231	Gallardo
2	Vin Diesel	Brazil	520	Mustang

If this table is stored in a row-oriented database. It will store the records as shown below:

1, Paul Walker, US, 231, Gallardo,

2, Vin Diesel, Brazil, 520, Mustang

In row-oriented databases data is stored on the basis of rows or tuples as you can see above. While the column-oriented databases store this data as:

1,2, Paul Walker, Vin Diesel, US, Brazil, 231, 520, Gallardo, Mustang

HBase Data Model organizes data in a way that allows efficient storage and retrieval, even for very large datasets. The data model is fundamentally different from traditional relational databases due to its column-oriented nature.

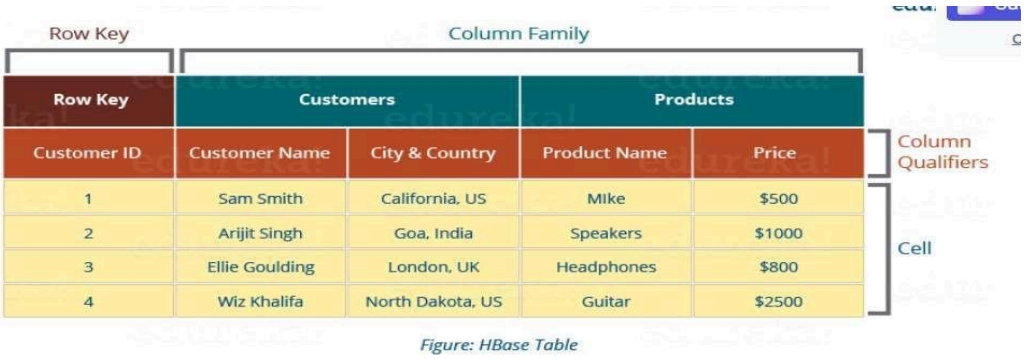


Figure: HBase Table

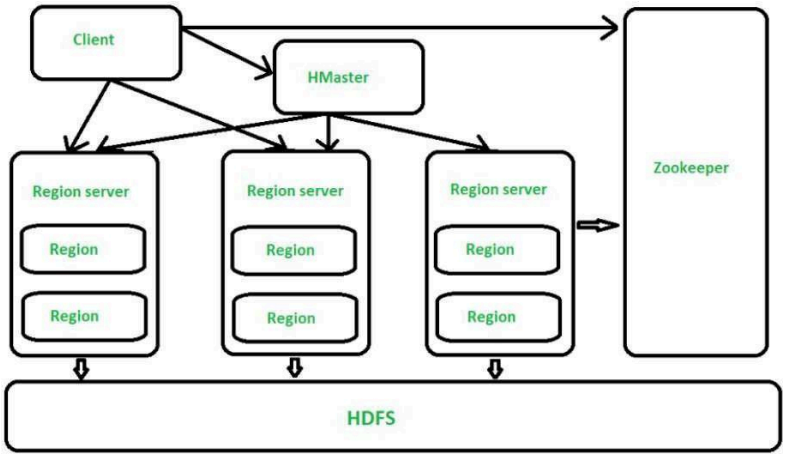
- **Tables**: Data is stored in a table format in HBase. But here tables are in column-oriented format.
- **Row Key**: Row keys are used to search records which make searches fast. You would be curious to know how? I will explain it in the architecture part moving ahead in this blog.
- **Column Families**: Various columns are combined in a column family. These column families are stored together which makes the searching process faster because data belonging to same column family can be accessed together in a single seek.
- **Column Qualifiers**: Each column's name is known as its column qualifier.
- **Cell**: Data is stored in cells. The data is dumped into cells which are specifically identified by rowkey and column qualifiers.
- **Timestamp**: Timestamp is a combination of date and time. Whenever data is stored, it is stored with its timestamp. This makes easy to search for a particular version of data.

1. **Tables**:
    - Data in HBase is organized into tables, each of which is identified by a unique name.
    - Tables are divided into rows and columns, but HBase is optimized for operations on columns rather than rows.
2. **Rows**:
    - Each row in an HBase table is uniquely identified by a **row key**.
    - The row key is a unique identifier for each row and determines the order of rows within a table.
    - Data is stored and retrieved based on this row key, which allows for efficient querying.
3. **Column Families**:
    - Columns in HBase are grouped into **column families**.
    - Each column family contains a set of columns that are stored together.
    - Column families are defined at the time of table creation and cannot be modified afterward.
    - Columns within a family are not predefined, allowing flexibility in storing data.
4. **Columns**:
    - Within column families, columns are identified by a **column qualifier**.

- Data in a column family can be sparsely populated, meaning not every row needs to have values for every column.

5. **Cells**:
    - The intersection of a row, column family, and column qualifier forms a cell where data is stored.
    - Each cell contains a value and a timestamp indicating when the value was last updated.
6. **Versions**:
    - HBase supports multiple versions of a cell value.
    - The number of versions stored can be configured to keep historical data or allow versioned access.

HBase Architecture



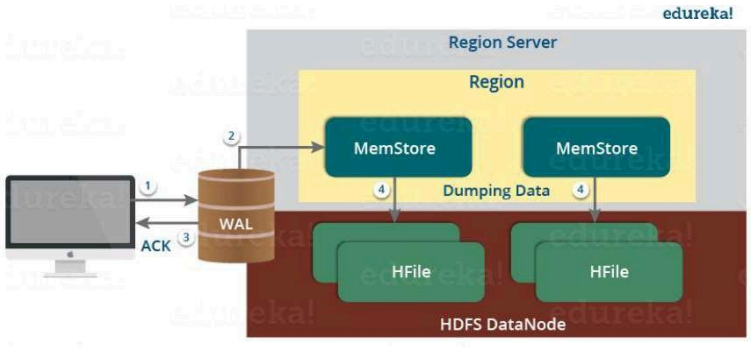
HBase Architecture involves several components that work together to manage data efficiently:

1. **HMaster**:
    - **Role**: Acts as the master server in HBase, responsible for managing the cluster and coordinating activities.
    - **Functions**: Handles administrative tasks such as table creation, deletion, and region assignment. Monitors Region Servers and balances the load across them.

- **Responsibilities**: Oversees region management and ensures overall health and performance of the cluster.

2. **Region Servers**:
   - **Role**: Worker nodes that handle read and write requests for regions.
   - **Functions**: Each Region Server manages one or more regions, which are subsets of a table's data. It performs operations such as storing, retrieving, and updating data.
   - **Responsibilities**: Interacts with HDFS to store and retrieve data blocks and maintain local data stores.
3. **Zookeeper**:
   - **Role**: Coordinates the distributed components and manages metadata.
   - **Functions**: Keeps track of the cluster state, maintains configuration information, and provides distributed synchronization services.
   - **Responsibilities**: Ensures that the HBase cluster operates smoothly by helping with tasks like leader election and server coordination.

HBase Read and Write Architecture



Write Mechanism:

1. **Write-Ahead Log (WAL)**:
   - When a client performs a write operation, the data is first recorded in the WAL.
   - The WAL is a log file maintained by Region Servers to ensure data durability and recoverability in case of failures.
2. **MemStore**:

- After writing to the WAL, the data is also stored in the MemStore, which is an in-memory cache for recent writes.
- The MemStore collects updates for a specific column family and maintains them in a sorted order.

3. **Acknowledgment**:
   - Once the data is written to the WAL and MemStore, the client receives an acknowledgment that the write operation is complete.
4. **HFile**:
   - When the MemStore reaches a certain threshold, its data is flushed to disk into HFiles (HBase's file format for storing data).
   - This process involves creating new HFiles and updating metadata to reflect the changes.

Read Mechanism:

1. **Block Cache**:
   - During a read operation, HBase first checks the block cache, which stores recently read data in memory for quick access.
2. **MemStore**:
   - If the data is not found in the block cache, HBase checks the MemStore for recent updates that might not yet be flushed to HFiles.
3. **HFile and Bloom Filters**:
   - If the data is still not found, HBase looks up HFiles on disk, using Bloom filters to efficiently locate relevant data blocks.
   - Bloom filters help reduce the number of disk reads by quickly determining whether a key is present in an HFile.
4. **Data Retrieval**:
   - The data is retrieved from the appropriate HFile and returned to the client.