**Third semester MCA Degree Examination, Dec.2025/Jan.2026**

**Big Data Analytics**

**Time: 3 hrs** **Max.Marks: 100**

**1. a. Define Big data and explain its evolution. Discuss the characteristics of Big data with suitable examples.**

Big Data refers to vast and rapidly growing volumes of data that are too large and complex for traditional data processing tools to manage. This data comes in many forms structured (e.g., tables), semi-structured (e.g., JSON, XML), and unstructured (e.g., text, images, video).

With the explosion of devices, sensors, online services, and digital platforms, data is now generated at an unprecedented rate. This growth makes it essential for organizations to adopt advanced tools and technologies to capture, store, analyze, and utilize this data effectively.

**Practical Uses of Big Data**
Organizations use Big Data to:

- Make smarter decisions by identifying trends and patterns
- Predict customer behavior and personalize user experiences
- Improve operational efficiency by finding process inefficiencies
- Innovate faster by identifying new business opportunities
- Enhance risk management by detecting fraud or security threats

Big Data transforms raw information into actionable insights that help companies gain a competitive edge.

**The 5 V's of Big Data**

- **Volume**: Refers to the huge amount of data generated every second-ranging from terabytes to petabytes. Example: YouTube uploads 500+ hours of video every minute.
- **Velocity**: The speed at which data is created, shared, and processed. Data streams in from sensors, social media, and transactions in real-time.
- **Variety**: Data comes in multiple formats-text, audio, images, videos, logs, sensor data, etc. Handling all these types together is complex
- **Veracity**: Refers to the trustworthiness and accuracy of the data. Inconsistent, duplicated, or noisy data can lead to wrong insights.
- **Value**: Not all data is useful. The key is extracting relevant data and turning it into business value through analytics.

**Evolution of Big Data**

- **Early Data Processing (Pre-2000s):** Focus on structured data in relational databases (SQL), managed by systems like mainframes, handling smaller, organized datasets for business intelligence.
- **Emergence of Web Scale (Early 2000s):** Explosion of internet data (text, images, logs) challenged traditional systems; concepts like data warehousing grew, but processing remained slow and batch-oriented.

- **The Hadoop Era (Mid-2000s):** Apache Hadoop (MapReduce) emerged, enabling distributed storage (HDFS) and processing of massive, diverse datasets, making big data analytics accessible.
- **Real-time & NoSQL (2010s):** Rise of NoSQL databases (MongoDB, Cassandra) for unstructured data, and stream processing (Spark Streaming) for real-time analytics, handling high velocity and variety.
- **Cloud & AI Integration (Late 2010s-Present):** Cloud platforms (AWS, Azure, GCP) offer scalable storage/compute; advanced AI/ML techniques (deep learning) analyze complex patterns, making data actionable for predictive insights and automated decisions.

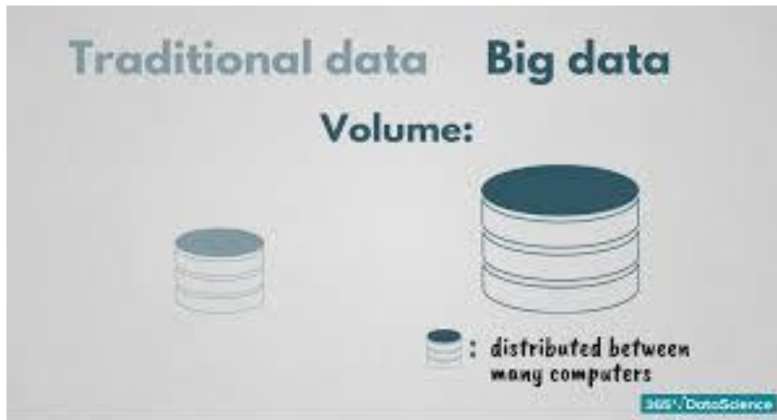
**B. Differentiate between traditional data processing systems and Big data systems. Explain why traditional systems are inadequate for handling Big data.**

| Feature | Traditional Data Systems | Big Data Systems |
|---|---|---|
| **Data Volume** | Gigabytes to Terabytes (GB-TB) | Petabytes to Zettabytes (PB-ZB) |
| **Data Type** | Primarily structured (fixed format, relational) | Structured, Semi-structured, Unstructured (text, video, images) |
| **Architecture** | Centralized, single server (vertical scaling) | Distributed, scale-out (horizontal scaling across many servers) |
| **Schema** | Fixed, predefined schema | Dynamic, flexible schema |
| **Velocity** | Processed in batches (hourly, daily) | Real-time or near real-time (per second) |
| **Tools** | Relational Databases (SQL), standard analytics | Hadoop, Spark, NoSQL, Machine Learning |

**Traditional Systems Are Inadequate for Big Data**

- **Scalability:** Traditional systems scale vertically (bigger single machine), hitting physical limits with massive data; Big Data needs horizontal scaling (more machines).
- **Variety:** Fixed schemas break with diverse, unstructured data (videos, social feeds) that don't fit neat rows and columns.

- **Velocity & Volume:** Centralized systems can't ingest or process data streams at the speed and sheer quantity Big Data generates.
- **Cost & Complexity:** Expensive, complex hardware for large storage/processing becomes prohibitive, while Big Data leverages commodity hardware in distributed clusters.
- **Analysis Limitations:** Traditional tools struggle to find patterns in petabytes of varied data, requiring advanced analytics (ML/NLP) inherent in Big Data frameworks.



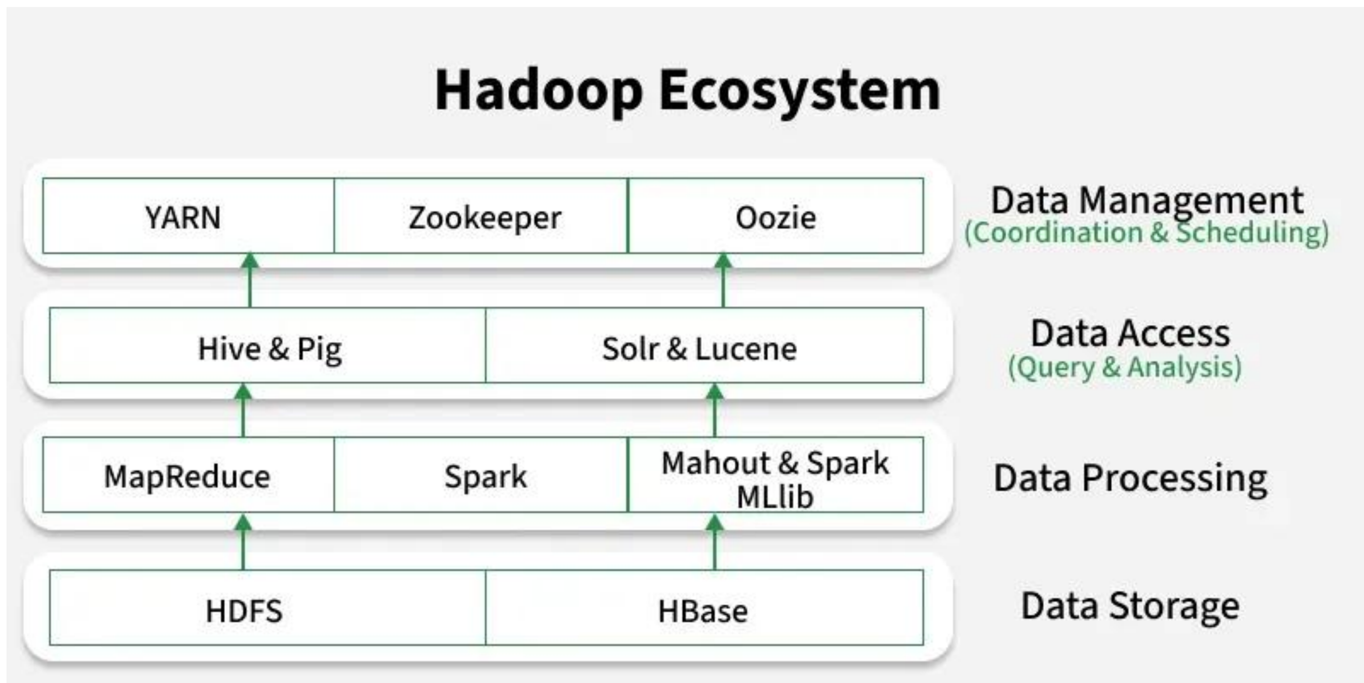

**2.A. Explain the Hadoop ecosystem. Describe the architecture and components of Hadoop including HDFS, YARN and Mapreduce.**

Hadoop is an open-source framework for storing and processing large-scale data across distributed clusters using commodity hardware. The Hadoop Ecosystem is a suite of tools and technologies built around Hadoop's core components (HDFS, YARN, MapReduce and Hadoop Common) to enhance its capabilities in data storage, processing, analysis and management.

**Components of Hadoop Ecosystem**

Hadoop Ecosystem comprises several components that work together for efficient big data storage and processing:

- **HDFS (Hadoop Distributed File System):** Stores large datasets across distributed nodes.
- **YARN (Yet Another Resource Negotiator):** Manages cluster resources and job scheduling.
- **MapReduce:** A programming model for batch data processing.
- **Spark:** Provides fast, in-memory data processing.
- **Hive & Pig:** High-level tools for querying and analyzing large datasets.
- **HBase:** A NoSQL database for real-time read/write access.
- **Mahout & Spark MLlib:** Libraries for scalable machine learning.
- **Solr & Lucene:** Tools for full-text search and indexing.
- **Zookeeper:** Manages coordination and configuration across the cluster.
- **Oozie:** A workflow scheduler for managing Hadoop jobs.




**HDFS**

HDFS is a core component of Hadoop ecosystem, designed to store large volumes of structured or unstructured data across multiple nodes. It manages metadata through log files and splits storage tasks between two main parts:

- **NameNode (master)**: Stores metadata (data about data) and requires fewer resources.
- **DataNodes (slaves):** Store actual data on commodity hardware, making Hadoop cost-effective.

HDFS handles coordination between clusters and hardware, serving as the backbone of entire Hadoop system.

**YARN**

YARN (Yet Another Resource Negotiator) is resource management layer of Hadoop, responsible for scheduling and allocating resources across the cluster. It has three key components:

- **ResourceManager:** Allocates resources to various applications in the system.
- **NodeManager:** Manages resources (CPU, memory, etc.) on individual nodes and reports to ResourceManager.
- **ApplicationMaster:** Acts as a bridge between ResourceManager and NodeManager, handling resource negotiation for each application.

Together, they ensure efficient resource utilization and smooth execution of jobs in the Hadoop cluster.

**MapReduce**

MapReduce enables distributed and parallel data processing on large datasets. It allows developers to write programs that transform big data into manageable results.

- **Map():** Processes input data by filtering, sorting and organizing it into key-value pairs.

- **Reduce():** Takes the output from Map(), aggregates the data and summarizes it into a smaller, consolidated set of results.

Together, they efficiently handle large-scale data transformations across the Hadoop cluster.

**PIG**

Pig is a platform developed by Yahoo for analyzing large datasets using **Pig Latin**, a SQL-like scripting language designed for data processing.

- It simplifies complex data flows and handles MapReduce operations internally.
- The processed results are stored in HDFS.
- **Pig Latin** runs on **Pig Runtime**, similar to Java on JVM.
- It enhances programming ease and optimization, making it a key part of Hadoop ecosystem.

**HIVE**

Hive uses a SQL-like interface (**HQL: Hive Query Language**) to read and write large datasets.

- It supports both real-time and batch processing, making it highly scalable.
- Hive supports all standard SQL datatypes, easing query operations.

It has two main components:

- **JDBC/ODBC drivers:** manage data connections and access permissions.
- **Hive Command Line:** used for query execution and processing.

**B. Discuss the limitations of Hadoop. Explain the reasons for the shift from Hadoop Mapreduce to Apache Spark.**

**Hadoop** is a platform that got its start as a Yahoo project in 2006, which became a top-level Apache open-source project afterward. This framework handles large datasets in a distributed fashion. The Hadoop ecosystem is highly fault-tolerant and does not depend upon hardware to achieve high availability. This framework is designed with a vision to look for the failures at the application layer. It's a general-purpose form of distributed processing that has several components:

- **Hadoop Distributed File System (HDFS):** This stores files in a Hadoop-native format and parallelizes them across a cluster. It manages the storage of large sets of data across a Hadoop Cluster. Hadoop can handle both structured and unstructured data.
- **YARN:** YARN is Yet Another Resource Negotiator. It is a schedule that coordinates application runtimes.
- **MapReduce:** It is the algorithm that actually processes the data in parallel to combine the pieces into the desired result.
- **Hadoop Common:** It is also known as Hadoop Core and it provides support to all other components it has a set of common libraries and utilities that all other modules depend on.

Hadoop is built in Java, and accessible through many programming languages, for writing MapReduce code, including Python, through a Thrift client. It's available either open-source through the Apache distribution, or through vendors such as Cloudera (the largest Hadoop vendor by size and scope), MapR, or HortonWorks.

**What is Spark?**

Apache Spark is an open-source tool. It is a newer project, initially developed in 2012, at the AMPLab at UC Berkeley. It is focused on processing data in parallel across a cluster, but the biggest difference is that it works in memory. It is designed to use RAM for caching and processing the data. Spark performs different types of big data workloads like:

- Batch processing.

- Real-time stream processing.
- Machine learning.
- Graph computation.
- Interactive queries.

**There are five main components of Apache Spark:**

- **Apache Spark Core:** It is responsible for functions like scheduling, input and output operations, task dispatching, etc.
- **Spark SQL:** This is used to gather information about structured data and how the data is processed.
- **Spark Streaming:** This component enables the processing of live data streams.
- **Machine Learning Library:** The goal of this component is scalability and to make machine learning more accessible.
- **GraphX:** This has a set of APIs that are used for facilitating graph analytics tasks.

Limitations of Hadoop

- **Slow Processing Speed:**

Hadoop's MapReduce model requires reading and writing data to disk between processing stages, which is a slow disk-seek operation, making it unsuitable for real-time or interactive data processing.

- **High Latency:**

The disk-centric nature of MapReduce results in high latency, as data must be written to disk after the "map" phase and re-read for the "reduce" phase.

- **Ineffective for Iterative Processing:**

Hadoop is not efficient for iterative algorithms, like those used in machine learning, because each iteration requires loading data from disk, increasing processing time.

- **Complex Programming Environment:**

Writing and debugging complex MapReduce jobs requires a high level of technical expertise, leading to a steep learning curve.

- **Small File Problem:**

Hadoop is not well-suited for processing a large number of small files because it stores data in large blocks, which increases overhead.

The Shift to Spark

Apache Spark emerged to address Hadoop's shortcomings, offering a more performant and flexible framework.

- **In-Memory Processing:**

Spark performs computations in RAM, avoiding the disk I/O bottleneck of Hadoop MapReduce and resulting in significantly faster processing speeds, especially for iterative and interactive workloads.

- **Real-time and Streaming Capabilities:**

Spark can process data in real-time from live events, a capability lacking in Hadoop's batch-oriented approach.

- **Unified API and Simplified Programming:**

Spark provides high-level APIs in Scala, Python, and Java, offering more concise and user-friendly dataframes and datasets compared to Hadoop's verbose Java MapReduce jobs.

- **Support for Multiple Workloads:**

Spark's unified engine supports a wide range of tasks, including SQL queries, machine learning, and graph processing, via libraries like Spark SQL, MLlib, and GraphX.

- **Improved Iterative Processing:**

By keeping intermediate data in memory across iterations, Spark dramatically accelerates machine learning algorithms and other iterative processes compared to Hadoop.

**3. A. Describe the design and operations of Hadoop Distributed File System(HDFS). Describe the roles of Namenode, Datanode and secondary Namenode.**

The Hadoop Distributed File System (HDFS) is a key component of the Apache Hadoop ecosystem, designed to store and manage large volumes of data across multiple machines in a distributed manner. It provides high-throughput access to data, making it suitable for applications that deal with large datasets, such as big data analytics, machine learning, and data warehousing. This article will delve into the architecture of HDFS, explaining its key components and mechanisms, and highlight the advantages it offers over traditional file systems.

**HDFS Architecture**

HDFS is designed to be highly scalable, reliable, and efficient, enabling the storage and processing of massive datasets. Its architecture consists of several key components:

1. NameNode
2. DataNode
3. Secondary NameNode
4. HDFS Client
5. Block Structure

**NameNode**

The NameNode is the master server that manages the filesystem namespace and controls access to files by clients. It performs operations such as opening, closing, and renaming files and directories. Additionally, the NameNode maps file blocks to DataNodes, maintaining the metadata and the overall structure of the file system. This metadata is stored in memory for fast access and persisted on disk for reliability.

**Key Responsibilities:**

- Maintaining the filesystem tree and metadata.
- Managing the mapping of file blocks to DataNodes.
- Ensuring data integrity and coordinating replication of data blocks.

**DataNode**

DataNodes are the worker nodes in HDFS, responsible for storing and retrieving actual data blocks as instructed by the NameNode. Each DataNode manages the storage attached to it and periodically reports the list of blocks it stores to the NameNode.

**Key Responsibilities:**

- Storing data blocks and serving read/write requests from clients.
- Performing block creation, deletion, and replication upon instruction from the NameNode.
- Periodically sending block reports and heartbeats to the NameNode to confirm its status.

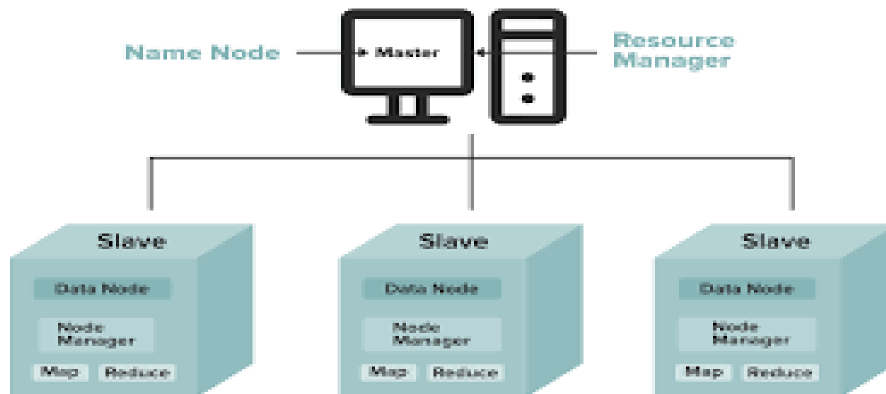
**Secondary NameNode**

The Secondary NameNode acts as a helper to the primary NameNode, primarily responsible for merging the EditLogs with the current filesystem image (FsImage) to reduce the potential load on the NameNode.

It creates checkpoints of the namespace to ensure that the filesystem metadata is up-to-date and can be recovered in case of a NameNode failure.

**Key Responsibilities:**

- Merging EditLogs with FsImage to create a new checkpoint.
- Helping to manage the NameNode's namespace metadata.




**B. Discuss the Hadoop Mapreduce programming model and also explain the job execution flow of a Mapreduce Program.**

MapReduce is a parallel, distributed programming model in the Hadoop framework that can be used to access the extensive data stored in the Hadoop Distributed File System (HDFS). The Hadoop is capable of running the MapReduce program written in various languages such as Java, Ruby, and Python. One of the beneficial factors that MapReduce aids is that MapReduce programs are inherently parallel, making the very large scale easier for data analysis.

When the MapReduce programs run in parallel, it speeds up the process. The process of running MapReduce programs is explained below.

- **Dividing the input into fixed-size chunks:** Initially, it divides the work into equal-sized pieces. When the file size varies, dividing the work into equal-sized pieces isn't the straightforward method to follow, because some processes will finish much earlier than others while some may take a very long run to complete their work. So one of the better approaches is that one that requires more work is said to split the input into fixed-size chunks and assign each chunk to a process.
- **Combining the results:** Combining results from independent processes is a crucial task in MapReduce programming because it may often need additional processing such as aggregating and finalizing the results.

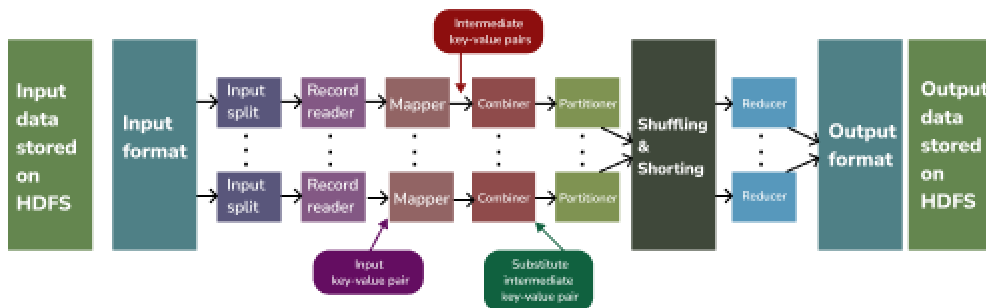
**Key components of MapReduce**

There are two key components in the MapReduce. The MapReduce consists of two primary phases such as the map phase and the reduces phase. Each phase contains the key-value pairs as its input and output and it also has the map function and reducer function within it.

- **<u>Mapper</u>:** Mapper is the first phase of the MapReduce. The Mapper is responsible for processing each input record and the key-value pairs are generated by the InputSplit and RecordReader. Where these key-value pairs can be completely different from the input pair. The MapReduce output holds the collection of all these key-value pairs.
- **<u>Reducer</u>:** The reducer phase is the second phase of the MapReduce. It is responsible for processing the output of the mapper. Once it completes processing the output of the mapper, the reducer now generates a new set of output that can be stored in HDFS as the final output data.

**Execution workflow of MapReduce**

Now let's understand how the MapReduce Job execution works and what all the components it contains. Generally, MapReduce processes the data in different phases with the help of its different components. Take a look at the below figure which illustrates the steps of the job execution workflow of MapReduce in Hadoop.




- **Input Files:** The data for MapReduce tasks are present in the input files. These input files reside in HDFS. The format for input files is arbitrary, while line-based log files and binary format can also be used.
- **InputFormat:** The InputFormat is used to define how the input files are split and read. It selects the files or objects that are used for input. In general, the InputFormat is used to create the Input Split.
- **Record Reader:** The RecordReader can communicate with the Input Split in the Hadoop MapReduce. It can also convert the data into key-value pairs so that the mapper can read. By default, the RecordReader utilizes the TextInputFormat for converting data into key-value pairs. The Record Reader communicates with the Input Split until the file reading is completed. It then assigns a byte offset (unique number) to each line present in the file. Then these key-value pairs are sent to the mapper for further processing.
- **Mapper:** From the RecordReader, the mapper receives the input records. The Mapper is responsible for processing those input records from the RecordReader and it generates the new key-value pair. The Key-value pair generated by the mapper can be completely different from the input pair. The output of the mapper which is intermediate output is said to be stored in the local disk since it is the temporary data.

- **Combiner:** The Combiner in MapReduce is also known as Mini-reducer. The Hadoop MapReduce combiner performs the local aggregation on the mapper's output which minimizes the data transfer between the mapper and reducer. Once the Combiner completes its process, the output of the combiner is passed to the partitioner for further work.

- **Partitioner:** In Hadoop MapReduce, the partitioner is used when we are working with more than one reducer. The Partitioner extracts the output from the combiner and then it performs partitioning. The partitioning of output takes place based on the key and then it is sorted. With the help of a hash function, the key (or subset of the key) is used to derive the partition. Since MapReduce execution works with the help of key-value, each combiner output is partitioned and a record having the same key value moves into the same partition and then each partition is sent to the reducer. Partitioning of the output of the combiner allows the even distribution of the map output over the reducer.
- **Shuffling and Sorting:** The Shuffling performs the shuffling operation on the mapper’s output before it is sent to the reducer phase. Once all the mapper has completed their work and their output is said to be shuffled on the reducer nodes, then this intermediate output is merged and sorted. This sorted output is passed as input to the reducer phase.
- **Reducer:** It takes the set of intermediate key-value pairs from the mapper as the input and then it runs the reducer function on each of the key-value pairs to generate the output. This output of the reducer phase is the final output and it is stored in the HDFS.
- **Record Writer:** The Record Writer holds the power of writing these output key-value pairs from the reducer phase to the output files.
- Output format: The Output format determines how these output values are written in the output files by the record reader. The Output format instances provided by Hadoop are generally used to write files on either HDFS or on the local disk. Thus, the final output of the reducer is written on the HDFS by output format instances.

**4. A. Explain the steps involved in writing Mapreduce programs with reference to Word Count and sorting algorithms.**

Counting the number of words in any language is a piece of cake like in C, C++, Python, Java, etc. MapReduce also uses Java but it is very easy if you know the syntax on how to write it. It is the basic of MapReduce. You will first learn how to execute this code similar to "Hello World" program in other languages. So here are the steps which show how to write a MapReduce code for Word Count.

**Mapper Code:**

```
// Importing libraries
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class WCMapper extends MapReduceBase implements Mapper<LongWritable,
```

```
                                        Text, Text, IntWritable> {

    // Map function
    public void map(LongWritable key, Text value, OutputCollector<Text,
                IntWritable> output, Reporter rep) throws IOException
    {

        String line = value.toString();

        // Splitting the line on spaces
        for (String word : line.split(" "))
        {
            if (word.length() > 0)
            {
                output.collect(new Text(word), new IntWritable(1));
            }
        }
    }
}
```

**Reducer Code:**

```
// Importing libraries
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

public class WCReducer extends MapReduceBase implements Reducer<Text,
                                IntWritable, Text, IntWritable> {

    // Reduce function
```

```
    public void reduce(Text key, Iterator<IntWritable> value,
                OutputCollector<Text, IntWritable> output,
                        Reporter rep) throws IOException
    {

        int count = 0;

        // Counting the frequency of each words
        while (value.hasNext())
        {
            IntWritable i = value.next();
            count += i.get();
        }

        output.collect(key, new IntWritable(count));
    }
}
```

**B. Explain advanced Hadoop features such as Coimbiners, Partioners and Counters. Discuss Hadoop streaming and its integration with python.**

Hadoop MapReduce was originally built for Java, which limited its accessibility to developers familiar with other languages. To address this, Hadoop introduced Streaming a utility that enables writing MapReduce programs in any language that supports standard input and output, such as Python, Bash or Perl.

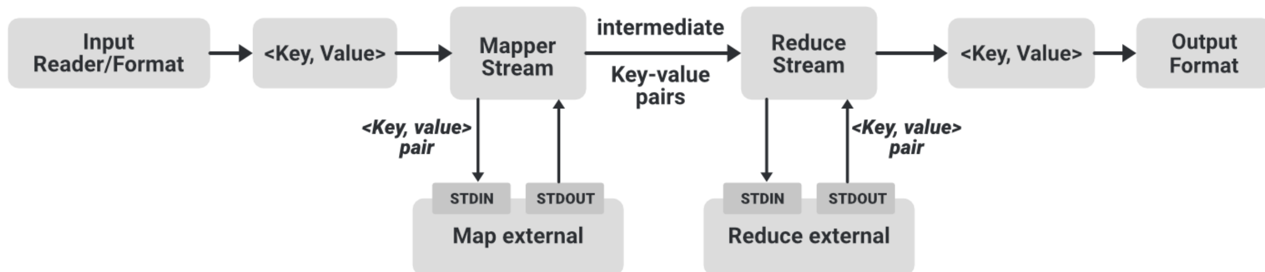
Hadoop Streaming, available since version 0.14.1, allows external scripts to be used as Mapper and Reducer tasks. These scripts process input from STDIN and produce output to STDOUT, enabling non-Java programs to participate fully in Hadoop's distributed data processing.

**Use Cases of Hadoop Streaming**

1. Suitable for developers preferring Python, Perl, Bash or other non-Java languages
2. Enables reuse of existing legacy scripts in MapReduce workflows
3. Facilitates rapid prototyping of data processing tasks using simple scripts
4. Supports development of custom mappers and reducers for non-standard or binary data formats

**Data Flow in Hadoop Streaming**

Hadoop Streaming processes key-value pairs through external mapper and reducer scripts using standard input and output. Let's see how data flows through each stage in the diagram below.




**1. Combiners (Mini-Reducers)**

A Combiner, often called a "mini-reducer," is an optional, localized reducer class that runs immediately after the Mapper phase and before the data is shuffled to the Reducer. Its primary purpose is to reduce network congestion by reducing the volume of data transferred between mappers and reducers.

- **Function:** It aggregates the intermediate key-value pairs generated by the Mapper locally on each mapper node.
- **Use Case:** Ideal for commutative and associative operations (e.g., sum, max, min) where local aggregation doesn't change the final result.
- **Advantages:** Significantly reduces network bandwidth usage and improves overall MapReduce performance by minimizing intermediate data.
- **Limitations:** It is not guaranteed to run by the framework; thus, a job cannot rely on it to run a specific number of times.

**2. Partitioners**

A Partitioner in Hadoop controls the partitioning of intermediate map outputs. It acts as a routing mechanism, deciding which reducer will process a particular intermediate key-value pair.

- **Function:** The Partitioner takes the key (or a subset of it) and the number of reducers to compute a partition index. It ensures all records with the same key are sent to the same reducer.
- **Default Behavior:** The default HashPartitioner uses a hash function to calculate the partition: (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks.
- **Custom Partitioner:** If the default partitioning leads to skewed data (some reducers getting much more data than others), a custom partitioner can be implemented to ensure even distribution.
- **Significance:** Crucial for balancing load across multiple reducers.

**3. Counters**

Counters are mechanisms used in Hadoop MapReduce to collect, measure, and report statistics about job execution, which are essential for quality control and problem diagnosis.

- **Built-in Counters:** Hadoop maintains built-in counters for every job, such as:
  - **MapReduce Task Counters:** Information about tasks (e.g., MAP_INPUT_RECORDS, REDUCE_OUTPUT_RECORDS).
  - **FileSystem Counters:** Bytes read or written by the file system.
  - **Job Counters:** Job-level statistics managed by the Application Master (e.g., TOTAL_LAUNCHED_MAPS).
- **User-Defined (Custom) Counters:** Users can define their own counters using Java enum to track application-specific events (e.g., counting the number of null values, invalid records, or specific business events).
- **Usage:** They are useful for checking if the expected amount of data was processed and for debugging.

**5.A.Explain in-memory big data processing using Apache Spark.Describe the Spark architecture and its main components.**

**Spark** is a cluster computing system. It is faster as compared to other cluster computing systems (such as Hadoop). It provides high-level APIs in Python, Scala, and Java. Parallel jobs are easy to write in Spark. In this article, we will discuss the different components of Apache Spark.

Spark processes a huge amount of datasets and it is the foremost active Apache project of the current time. Spark is written in Scala and provides API in Python, Scala, Java, and R. The most vital feature of Apache Spark is its in-memory cluster computing that extends the speed of the data process. Spark is an additional general and quicker processing platform. It helps us to run programs relatively quicker than Hadoop (i.e.) a hundred times quicker in memory and ten times quicker even on the disk. The main features of spark are:

1. **Multiple Language Support:** Apache Spark supports multiple languages; it provides API's written in Scala, Java, Python or R. It permits users to write down applications in several languages.
2. **Quick Speed:** The most vital feature of Apache Spark is its processing speed. It permits the application to run on a Hadoop cluster, up to one hundred times quicker in memory, and ten times quicker on disk.
3. **Runs Everywhere:** Spark will run on multiple platforms while not moving the processing speed. It will run on Hadoop, Kubernetes, Mesos, Standalone, and even within the Cloud.
4. **General Purpose:** It is powered by plethora libraries for machine learning (i.e.) MLlib, DataFrames, and SQL at the side of Spark Streaming and GraphX. It is allowed to use a mix of those libraries which are coherently associated with the application. The feature of mix streaming, SQL, and complicated analytics, within the same application, makes Spark a general framework.

5. **Advanced Analytics:** Apache Spark also supports "Map" and "Reduce" that has been mentioned earlier. However, at the side of MapReduce, it supports Streaming data, SQL queries, Graph algorithms, and Machine learning. Thus, Apache Spark may be used to perform advanced analytics.

The above figure illustrates all the spark components. Let's understand each of the components in detail:

1. **Spark Core:** All the functionalities being provided by Apache Spark are built on the highest of the Spark Core. It delivers speed by providing in-memory computation capability. Spark Core is the foundation of parallel and distributed processing of giant dataset. It is the main backbone of the essential I/O functionalities and significant in programming and observing the role of the spark cluster. It holds all the components related to scheduling, distributing and monitoring jobs on a cluster, Task dispatching, Fault recovery. The functionalities of this component are:
   1. It contains the basic functionality of spark. (Task scheduling, memory management, fault recovery, interacting with storage systems).
   2. Home to API that defines RDDs.
2. **Spark SQL Structured data:** The Spark SQL component is built above the spark core and used to provide the structured processing on the data. It provides standard access to a range of data sources. It includes Hive, JSON, and JDBC. It supports querying data either via SQL or via the hive language. This also works to access structured and semi-structured information. It also provides powerful, interactive, analytical application across both streaming and historical data. Spark SQL could be a new module in the spark that integrates the relative process with the spark with programming API. The main functionality of this module is:
   1. It is a Spark package for working with structured data.
   2. It Supports many sources of data including hive tablets, parquet, json.
   3. It allows the developers to intermix SQK with programmatic data manipulation supported by RDDs in python, scala and java.
3. **Spark Streaming:** Spark streaming permits ascendible, high-throughput, fault-tolerant stream process of live knowledge streams. Spark can access data from a source like a flume, TCP socket. It will operate different algorithms in which it receives the data in a file system, database and live dashboard. Spark uses Micro-batching for real-time streaming. Micro-batching is a technique that permits a method or a task to treat a stream as a sequence of little batches of information. Hence spark streaming groups the live data into small batches. It delivers it to the batch system for processing. The functionality of this module is:
   1. Enables processing of live streams of data like log files generated by production web services.
   2. The API's defined in this module are quite similar to spark core RDD API's.
4. **Mllib Machine Learning:** MLlib in spark is a scalable Machine learning library that contains various machine learning algorithms. The motive behind MLlib creation is to make the implementation of

machine learning simple. It contains machine learning libraries and the implementation of various algorithms. For example, clustering, regression, classification and collaborative filtering.

5. **GraphX graph processing:** It is an API for graphs and graph parallel execution. There is network analytics in which we store the data. Clustering, classification, traversal, searching, and pathfinding is also possible in the graph. It generally optimizes how we can represent vertex and edges in a graph. GraphX also optimizes how we can represent vertex and edges when they are primitive data types. To support graph computation, it supports fundamental operations like subgraph, joins vertices, and aggregate messages as well as an optimized variant of the Pregel API.

**B. Explain Resilient Distributed Datasets(RDDs). Discuss RDD transformations and actions with examples.**

A Resilient Distributed Dataset (RDD) is an immutable, fault-tolerant collection of elements that can be distributed across multiple cluster nodes to be processed in parallel. RDDs are the basic data structure within the open source data processing engine Apache Spark.

Spark was developed to address shortcomings in MapReduce, a programming model for “chunking” a large data processing task into smaller parallel tasks.

MapReduce can be slow and inefficient. It requires replication (maintaining multiple copies of data in different locations), serialization (coordinating access to resources used by more than one program) and intense I/O (input/output of disk storage).

Spark specifically reduces unnecessary processing. Whereas MapReduce writes intermediate data to disk, Spark uses RDDs to cache and compute data in memory. The result is that Spark’s analytics engine can process data 10–100 times faster than MapReduce.

RDD and Apache Spark

Apache Spark is a fast, open source, large-scale data-processing engine often used for machine learning (ML) and artificial intelligence (AI) applications. Spark can be viewed as an improvement on Hadoop, more specifically, on Hadoop's native data processing framework, MapReduce.

Spark scales by distributing data-processing workflows across large clusters of computers, with built-in support for parallel computing on multiple nodes and fault tolerance.

It includes application programming interfaces (APIs) for common data science and data engineering programming languages, including Java™, Python (PySpark), Scala and R.

Spark uses RDDs to manage and process data. Each RDD is divided into logical partitions, which can be computed on different cluster nodes simultaneously. Users can perform 2 types of RDD operations: transformations and actions.

- Transformations are operations that create a new RDD.
- Actions instruct Spark to apply computation and pass the result back to the Spark driver, the process that manages Spark jobs.

Spark performs transformations and actions on RDDs in memory—the key to Spark's speed. Spark can also store data in memory or write the data to disk for added persistence.

How RDD works

Resilient Distributed Datasets are resilient and distributed. That means:

Resilient

RDDs are called "resilient" because they track data lineage information so that lost data can be rebuilt if there is a failure, making RDDs highly fault-tolerant.

As an example of this data resilience, consider an executor core that is lost during the processing of an RDD partition. The driver would detect that failure, and that partition would be reassigned to a different executor core.

Distributed

RDDs are called "distributed" because they are split into smaller groups of data that can be distributed to different compute nodes and processed simultaneously.

In addition to these 2 core characteristics, RDD has other features that contribute to its importance and operations in Spark.

**6.A. Explain Dataframes and Spark SQL Describe their role in structured data processing.**

A DataFrame can be constructed from an array of different sources such as Hive tables, Structured Data files, external databases, or existing RDDs. This API was designed for modern Big Data and data science applications taking inspiration from DataFrame in R Programming and Pandas in Python.

Features of DataFrame

Here is a set of few characteristic features of DataFrame −

- Ability to process the data in the size of Kilobytes to Petabytes on a single node cluster to large cluster.
- Supports different data formats (Avro, csv, elastic search, and Cassandra) and storage systems (HDFS, HIVE tables, mysql, etc).
- State of art optimization and code generation through the Spark SQL Catalyst optimizer (tree transformation framework).

Can be easily integrated with all Big Data tools and frameworks via Spark-Core.

Provides API for Python, Java, Scala, and R Programming.

SQLContext

SQLContext is a class and is used for initializing the functionalities of Spark SQL. SparkContext class object (sc) is required for initializing SQLContext class object.

The following command is used for initializing the SparkContext through spark-shell.

```
$ spark-shell
```

By default, the SparkContext object is initialized with the name **sc** when the spark-shell starts.

Use the following command to create SQLContext.

```
scala> val sqlcontext = new org.apache.spark.sql.SQLContext(sc)
```

Example

Let us consider an example of employee records in a JSON file named **employee.json**. Use the following commands to create a DataFrame (df) and read a JSON document named **employee.json** with the following content.

**employee.json** − Place this file in the directory where the current **scala>** pointer is located.

```
{
  {"id" : "1201", "name" : "satish", "age" : "25"}
  {"id" : "1202", "name" : "krishna", "age" : "28"}
  {"id" : "1203", "name" : "amith", "age" : "39"}
  {"id" : "1204", "name" : "javed", "age" : "23"}
  {"id" : "1205", "name" : "prudvi", "age" : "23"}
}
```

DataFrame Operations

DataFrame provides a domain-specific language for structured data manipulation. Here, we include some basic examples of structured data processing using DataFrames.

Follow the steps given below to perform DataFrame operations −

Read the JSON Document

First, we have to read the JSON document. Based on this, generate a DataFrame named (dfs).

Use the following command to read the JSON document named **employee.json**. The data is shown as a table with the fields − id, name, and age.

```
scala> val dfs = sqlContext.read.json("employee.json")
```

**Output** − The field names are taken automatically from **employee.json**.

```
dfs: org.apache.spark.sql.DataFrame = [age: string, id: string, name: string]
```

Show the Data

If you want to see the data in the DataFrame, then use the following command.

```
scala> dfs.show()
```

**Output** − You can see the employee data in a tabular format.

```
<console>:22, took 0.052610 s
+----+------+--------+
|age | id   |  name  |
+----+------+--------+
| 25 | 1201 | satish |
| 28 | 1202 | krishna|
| 39 | 1203 | amith  |
| 23 | 1204 | javed  |
| 23 | 1205 | prudvi |
+----+------+--------+
```

Use printSchema Method

If you want to see the Structure (Schema) of the DataFrame, then use the following command.

```
scala> dfs.printSchema()
```

**Output**

```
root
   |-- age: string (nullable = true)
   |-- id: string (nullable = true)
   |-- name: string (nullable = true)
```

Use Select Method

Use the following command to fetch **name**-column among three columns from the DataFrame.

```
scala> dfs.select("name").show()
```

**Output** − You can see the values of the **name** column.

```
<console>:22, took 0.044023 s
+--------+
```

```
|  name  |
+--------+
| satish |
| krishna|
| amith  |
| javed  |
| prudvi |
+--------+
```

**B. Explain Spark MLib and its role in machine learning. Discuss the limitations of traditional RDBMS in handling Big Data.**

MLlib is a machine learning framework built on top of Apache Spark. It is designed to make machine learning tasks faster and more efficient. MLlib supports various ML algorithms and utilities.

**Workflow of MLlib Models**

1. **Data Ingestion**: Load data using Spark DataFrames.
2. **Data Preprocessing**: Cleaning, handling missing values. Feature selection and transformation
3. **Model Selection**: Choose a machine learning algorithm based on data.
4. **Training**: Fit the model on training data.
5. **Prediction**: Use the trained model to make predictions on test or new data.
6. **Evaluation**: Evaluate model performance using various metrics.
7. **Pipeline Deployment**: Build a pipeline combining all steps.

The above image illustrates the Workflow of a Model using MLlib.

**Major Algorithms in MLlib**

**1. Classification Models**

- Used to categorize data into predefined labels.
- Used in Email spam detection
- Handle binary and multiclass problems

**2. Regression Models**

- Used to predict continuous values.
- Used in House price prediction
- Minimize error between predicted and actual values

**3. Clustering Models**

- Used to group data points without labels.
- Used in Customer segmentation
- Unsupervised learning based on similarity

**4. Recommendation Algorithms**

- Used for personalized content delivery.
- Used in Movie recommendations
- Collaborative filtering based on user-item interactions

**5. Dimensionality Reduction**

- Used to reduce feature space while preserving data variance.
- Used in Visualization or preprocessing
- Helps in improving performance and reducing noise

**6. Feature Transformation**

- Essential for preparing raw data into a usable format.
- Encoding categorical variables and scaling features
- Required before model training

MLlib simplifies large-scale machine learning by combining Spark with ML algorithms. It allows seamless integration of preprocessing, training and evaluation in one environment and is ideal for production-level systems where both scalability and speed are crucial.

**Strengths of MLlib**

- **Comprehensive ML support**: It provides a wide range of algorithms for classification, regression, clustering, recommendation and more making it suitable for various machine learning tasks.
- **Scalable and fault-tolerant**: Designed to handle large-scale datasets across distributed systems. It ensures fault tolerance so processes continue smoothly even in case of hardware failures.
- **Simplified API for common ML tasks**: MLlib offers an easy-to-use API that abstracts the complexity of distributed computing and makes it easier for users to implement common machine learning tasks like training models and making predictions.
- **Good speed**: It uses Apache Spark for distributed processing, providing high-speed model training and inference making it suitable for big data applications.
- **Easy integration**: It integrates seamlessly with Spark's data processing framework allowing it to work well with Spark's other components like Spark SQL, Spark Streaming and ML pipelines.

**7.A. Explain the need for NOSQL databases. Discuss the limitations of traditional RDBMS in handling Big data.**

NoSQL (Not Only SQL) databases are designed to handle large volumes of unstructured and semi-structured data. Unlike traditional relational databases that rely on fixed schemas and tables, NoSQL offers flexible data models and supports horizontal scaling. This makes them well-suited for modern applications that require high performance, scalability, and the ability to manage diverse data types efficiently.

**Features of NoSQL Databases**

- **Dynamic schema:** Allow flexible shaping of data to meet new requirements without the need to migrate or change schemas.
- **Horizontal scalability:** They scale horizontally for adding more nodes into the existing ones and acquire enough storage for even bigger datasets and much higher traffic by distributing the load on multiple servers.
- **Document-based:** Data are presented in flexible, semi-structured formats like JSON/BSON (e.g., MongoDB).
- **Key-value-based:** They possess a simple but fast access pattern (e.g., Redis) by storing data as pairs of keys and values.
- **Column-based:** Data are organized into columns instead of rows (e.g., CASSANDRA).
- **Distributed and high availability:** They are designed to be highly available and to automatically handle node failures and data replication across multiple nodes in a database cluster.
- **Flexibility:** Allow developers to store and retrieve data in a flexible and dynamic manner, with support for multiple data types and changing data structures.
- **Performance:** Perfect for big data and real-time analytics and high volume applications.

| Feature | SQL (Relational DB) | NoSQL (Non-Relational DB) |
|---|---|---|
| **Data Model** | Structured, Tabular | Flexible (Documents, Key-Value, Graphs) |
| **Scalability** | Vertical Scaling | Horizontal Scaling |
| **Schema** | Predefined | Dynamic & Schema-less |
| **ACID Support** | Strong | Limited or Eventual Consistency |
| **Best For** | Transactional applications | Big data, real-time analytics |
| **Examples** | MySQL, PostgreSQL, Oracle | MongoDB, Cassandra, Redis |

**Use of NoSQL**

- **Big Data Applications:** Efficiently stores and processes massive amounts of unstructured and semi-structured data.
- **Real-Time Analytics:** Supports fast queries and analysis for use cases like recommendation engines or fraud detection.
- **Scalable Web Applications:** Handles high traffic and large user bases by scaling horizontally across servers.
- **Flexible Data Storage:** Manages diverse data formats (JSON, key-value, documents, graphs) without rigid schemas.

**B. Explain the types of NOSQL databases Keyvalue, Document, Column and graph with suitable examples.**

Databases store organized data for easy access and management. Traditional relational databases use structured tables, but modern applications and big data have driven the rise of NoSQL systems. NoSQL databases handle large volumes of unstructured and semi-structured data, offering the scalability and flexibility today's diverse workloads demand.

**Types of NoSQL Database**

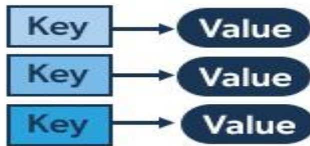
NoSQL databases can be classified into **four main types**, based on their data storage and retrieval methods:

1. Document-based databases
2. Key-value stores
3. Column-oriented databases
4. Graph-based databases

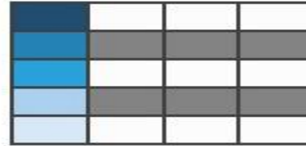
Each type has unique advantages and use cases, making NoSQL a preferred choice for big data applications, real-time analytics, cloud computing and distributed systems.

NoSQL

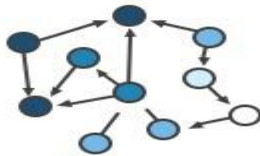
Key-Value




Column-Family



Graph



Document

**1. Document-Based Database**

The document-based database is a nonrelational database. Instead of storing the data in rows and columns (tables), it uses the documents to store the data in the database. A document database stores data in JSON, BSON or XML documents.

Documents can be stored and retrieved in a form that is much closer to the data objects used in applications which means less translation is required to use these data in the applications. In the Document database, the particular elements can be accessed by using the index value that is assigned for faster querying.

Collections are the group of documents that store documents that have similar contents. Not all the documents are in any collection as they require a similar schema because document databases have a flexible schema.

| **Database** | **Use Case** |
|---|---|
| **MongoDB** | Content management, product catalogs, user profiles |
| **CouchDB** | Offline applications, mobile synchronization |
| **Firebase Firestore** | Real-time apps, chat applications |

**2. Key-Value Stores**

A key-value store is a nonrelational database. The simplest form of a NoSQL database is a **key-value store**. Every data element in the database is stored in key-value pairs. The data can be retrieved by using a unique key allotted to each element in the database. The values can be simple data types like strings,

numbers or complex objects. A key-value store is like a relational database with only two columns which is the key and the value.

| Database | Use Case |
| --- | --- |
| **Redis** | Caching, real-time leaderboards, session storage |
| **Memcached** | High-speed in-memory caching |
| **Amazon DynamoDB** | Cloud-based scalable applications |

**3. Column Oriented Databases**

A column-oriented database is a non-relational database that stores the data in columns instead of rows. That means when we want to run analytics on a small number of columns, we can read those columns directly without consuming memory with the unwanted data. Columnar databases are designed to read data more efficiently and retrieve the data with greater speed. A columnar database is used to store a large amount of data.

| Database | Use Case |
| --- | --- |
| **Apache Cassandra** | Real-time analytics, IoT applications |
| **Google Bigtable** | Large-scale machine learning, time-series data |
| **HBase** | Hadoop ecosystem, distributed storage |

**4. Graph-Based Databases**

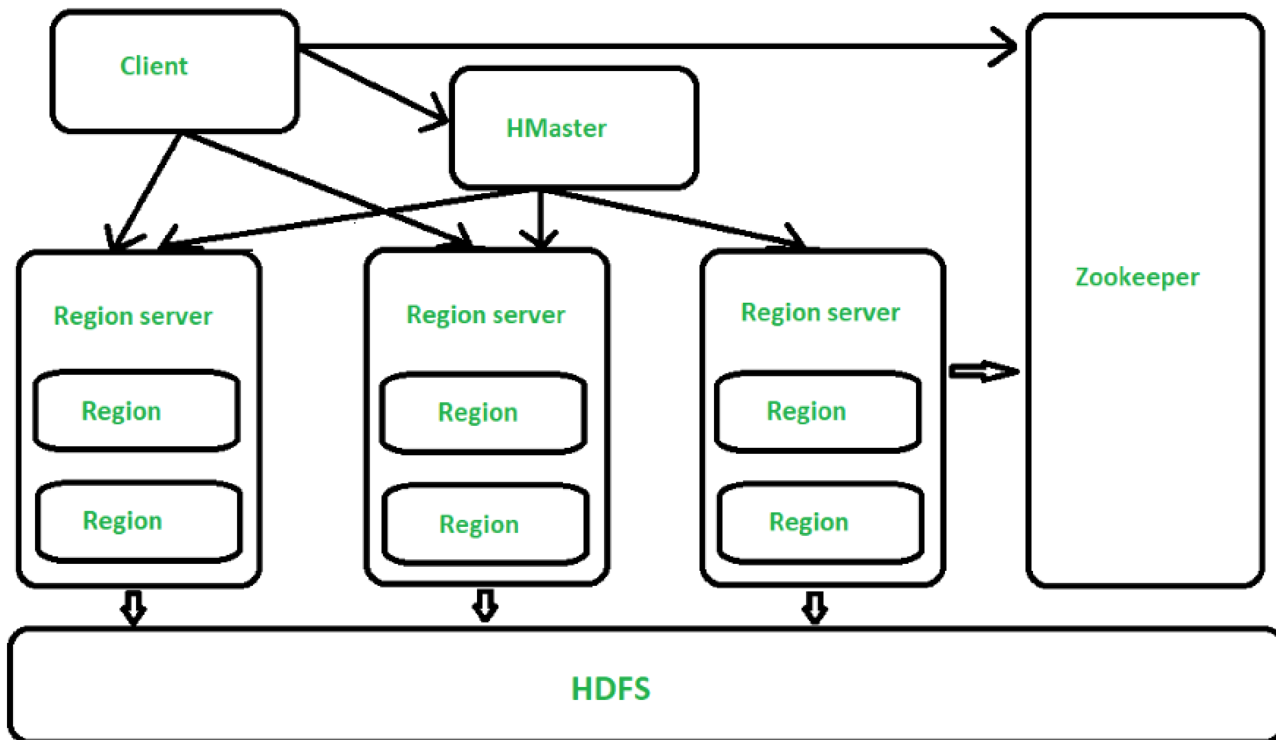
Graph-based databases focus on the relationship between the elements. It stores the data in the form of nodes in the database. The connections between the nodes are called links or relationships, making them ideal for complex relationship-based queries.

- Data is represented as nodes (objects) and edges (connections).
- Fast graph traversal algorithms help retrieve relationships quickly.
- Used in scenarios where relationships are as important as the data itself.

| Database | Use Case |
| --- | --- |
| **Neo4j** | Fraud detection, social networks |

| Database | Use Case |
|---|---|
| **Amazon Neptune** | Knowledge graphs, AI recommendations |
| **ArangoDB** | Multi-model database, cybersecurity |

**8.A. Explain HBase architecture. Describe thr CRUD operations performed in HBase.**
HBase is a distributed, scalable, NoSQL database built on top of Hadoop. It is designed to store huge amounts of structured or semi-structured data and provide fast, random read/write access. To achieve this, HBase relies on three main components in its architecture: HMaster, Region Server, and ZooKeeper.




**HMaster**
The **HMaster** acts as the **main coordinator** of the HBase cluster.
Think of it as the *manager* that oversees how data is distributed and how the cluster functions.

**Key Roles of HMaster**

- Assigns regions (data chunks) to Region Servers
- Manages table operations like **create**, **delete**, and **modify**
- Monitors the health of Region Servers
- Balances load across servers
- Handles failover when a server crashes

In large clusters, multiple backup HMasters run to ensure high availability.

**Region Server**

HBase tables are very large, so they are divided horizontally into smaller parts called Regions. A Region Server is responsible for managing these regions.

In HBase, the fundamental **CRUD** operations (Create, Read, Update, Delete) are performed at the **row level** using the HBase Shell or the client APIs (Java, REST, etc.). Operations are atomic within a single row.

Here are the specific operations and their corresponding commands in the HBase Shell:

**Create (or Insert) and Update**: The put command is used for both creating new data and updating existing data.

**Syntax**: put 'table_name', 'row_key', 'column_family:column_qualifier', 'value'

*Note*: When updating, if a row and column combination already exists, the new value is stored with a later timestamp (or a specified one), but the old value may be retained as a previous version depending on table schema settings.

**Read (Retrieve)**: Data retrieval is primarily done using the get or scan commands.

**get**: Retrieves specific cells from a single row.

**Syntax**: get 'table_name', 'row_key', 'column_family:column_qualifier' (or for a whole row: get 'table_name', 'row_key')

**scan**: Retrieves data across multiple rows, useful for listing all data or a range of data within a table.

**Syntax**: scan 'table_name' (can use filters for specific ranges or columns)

**Delete**: The delete and deleteall commands are used to remove data.

**delete**: Deletes a specific cell value (the latest version by default) in a row.

**Syntax**: delete 'table_name', 'row_key', 'column_family:column_qualifier'

**deleteall**: Deletes all columns (cells) in a specified row.

**Syntax**: deleteall 'table_name', 'row_key'

*Note*: In HBase, a delete operation writes a special "tombstone" marker, and the actual data is only removed during a major compaction process.

For programmatic access, the HBase Java client API uses corresponding classes and methods such as Put (for create/update), Get and Scan (for read), and Delete (for delete operations).

**B. Explain Cassandra and MongoDB. Discuss data modeling techniques used for achieving scalability and performance in Big data systems.**

**1. Apache Cassandra**

Cassandra is a distributed, wide-column store designed to handle massive amounts of data across many servers without a single point of failure. Originally developed by Facebook and maintained by Apache, it uses a **peer-to-peer architecture** where every node is equal.

- **Data Model:** Wide-column, row-oriented model. Data is stored in column families (tables).
- **Key Features:**
  - **High Write Throughput:** Capable of handling millions of writes per second, making it ideal for IoT, time-series data, and logging.
  - **Linear Scalability:** Adds nodes easily to increase capacity with no downtime.
  - **Tunable Consistency:** Allows tuning the tradeoff between speed and consistency (e.g., consistency level ONE for speed, QUORUM for balance).
  - **Distributed Architecture:** No master node, offering high availability across multiple data centers.

**2. MongoDB**

MongoDB is a document-oriented database that stores data in flexible, JSON-like BSON (Binary JSON) documents. It is a general-purpose NoSQL database developed by MongoDB Inc., designed for ease of development and scalability.

- **Data Model:** Document-oriented. Data is stored in collections of nested BSON documents.
- **Key Features:**
  - **Schema Flexibility:** No predefined schema, allowing for rapid iteration and dynamic data structures.
  - **Rich Queries & Aggregation:** Supports ad-hoc queries, indexing, and complex data processing via an aggregation pipeline.
  - **Horizontal Scaling:** Achieved through sharding (distributing data across multiple replica sets).
  - **Strong Consistency:** Strong consistency is the default, ensuring data integrity.

**Data Modeling for Scalability and Performance**

To achieve high scalability (linear growth) and performance (low latency), data modeling in NoSQL differs from traditional RDBMS.

**A. Cassandra Data Modeling Techniques**

Cassandra modeling is **query-driven**, meaning data structures are designed based on the specific read queries needed, rather than the entities.

- **Denormalization:** Duplicating data into multiple tables to support different query patterns is standard practice to avoid joins, which are not supported in Cassandra.

- **Partition Key Selection:** Choosing a good partition key is critical for balancing data distribution across nodes. A poor key leads to "hotspots," where one node takes most of the traffic.
- **Clustering Columns:** Used to sort data *within* a partition, allowing for efficient range queries (e.g., sorting messages by time within a user ID partition).
- **Avoiding Secondary Indexes:** Secondary indexes can be inefficient in a distributed system; creating tailored table designs (materialized views or extra tables) is preferred for performance.

**B. MongoDB Data Modeling Techniques**

MongoDB modeling focuses on optimizing for read/write performance by leveraging document structure.

- **Embedding Data:** Embedding related data within a single document (nested structures) reduces the need for complex joins, improving read performance because related data is fetched in one query.
- **Referencing (Linking) Data:** For one-to-many relationships where the "many" side grows indefinitely, referencing (using _id to link documents) is used to avoid exceeding the 16MB document size limit.
- **Sharding Key Strategy:** Sharding horizontally partitions data. Choosing an effective shard key (e.g., using a hashed key for random distribution) is essential for uniform performance.
- **The Subset Pattern:** Storing only the most frequently accessed data (e.g., the last 5 comments on a post) within the main document, rather than the entire history, to keep the working set small and memory-resident.

**9.A. Explain the role of Big Data tools. Such as Hive, Pig, Sqoop, Flume ad Sqoop with suitable examples.**

**PIG**

Pig is a platform developed by Yahoo for analyzing large datasets using **Pig Latin**, a SQL-like scripting language designed for data processing.

- It simplifies complex data flows and handles MapReduce operations internally.
- The processed results are stored in HDFS.
- **Pig Latin** runs on **Pig Runtime**, similar to Java on JVM.
- It enhances programming ease and optimization, making it a key part of Hadoop ecosystem.

**HIVE**

Hive uses a SQL-like interface (**HQL: Hive Query Language**) to read and write large datasets.

- It supports both real-time and batch processing, making it highly scalable.
- Hive supports all standard SQL datatypes, easing query operations.

It has two main components:

- **JDBC/ODBC drivers:** manage data connections and access permissions.
- **Hive Command Line:** used for query execution and processing.
  - **Sqoop (Data Transfer)**
    - **Role:** Designed for efficiently importing and exporting data between Hadoop (HDFS) and structured, relational databases like MySQL or Oracle.

    - **Example:** Transferring user profile data from a production MySQL database into HDFS for further analysis by Spark or Hive.

- **Flume (Data Ingestion)**
    - **Role:** A distributed, reliable service for efficiently collecting, aggregating, and moving large amounts of streaming log data into HDFS.
    - **Example:** Collecting real-time web server logs or social media feeds and streaming them directly into HDFS for immediate analysis.

**Summary of Use Cases**

| **Tool** | **Primary Role** | **Best For** |
|---|---|---|
| **Hive** | SQL Interface | Structured data analysis & Data Warehousing |
| **Pig** | Scripting (ETL) | Unstructured/Semi-structured data transformation |
| **Sqoop** | Data Transfer | Batch loading data from RDBMS to HDFS |
| **Flume** | Data Ingestion | Streaming log data ingestion |

**B. Explain data ingestion in Big Data systems. Describe the working of Flume and Sqoop with suitable examples.**

Data ingestion is the process of collecting raw data from various source systems, transforming it into a usable format, and loading it into a target big data environment, such as a data warehouse or data lake. This crucial first step in the big data pipeline facilitates data analysis and storage

Key elements of data ingestion include data sources (databases, files, real-time streams), data formats (structured, semi-structured, unstructured), and ingestion methods (batch vs. real-time streaming)

**Apache Flume**

Apache Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data from various sources to a centralized data store .It is primarily designed for streaming data, particularly log files and event data, from systems like web servers, social media, and application servers into HDFS or other storage systems.

**Working Principle**

Flume uses a simple **agent** architecture. An agent is a Java virtual machine (JVM) process that hosts the core components: Source, Channel, and Sink .

- **Source:** Receives data (events) from external systems.
- **Channel:** Acts as a temporary store for events, bridging the Source and Sink (e.g., a file channel, memory channel, or Kafka channel).
- **Sink:** Consumes events from the channel and delivers them to the final destination .

**Example**

To ingest real-time web server log data into HDFS:

1. **Source:** A SyslogTcpSource is configured on the log server to listen for incoming log entries.
2. **Channel:** A FileChannel stores the events reliably on disk until they are committed.
3. **Sink:** An HDFSSink writes the collected logs to files in the Hadoop Distributed File System (HDFS).

This setup ensures that as new log entries are generated, they are continuously streamed into HDFS for processing .

**Apache Sqoop**

Apache Sqoop is a tool designed to transfer data efficiently between Apache Hadoop and relational databases (like MySQL, PostgreSQL, Oracle, or enterprise data warehouses) . It is optimized for batch processing and handles structured data effectively .

**Working Principle**

Sqoop automates the creation of MapReduce jobs to parallelize the data transfer process.

- **Import:** When importing data from a relational database management system (RDBMS) to Hadoop, Sqoop translates the database schema into a Java class, generates the necessary MapReduce jobs to read the data in parallel, and writes it to the target location (HDFS, Hive, or HBase) .
- **Export:** When exporting data from Hadoop to an RDBMS, Sqoop reads the data in parallel from HDFS and writes it to the database table(s) .

**Example**

To perform a daily import of a products table from a MySQL database into HDFS:

1. **Connection:** The user provides the database connection details (JDBC URL, username, password).
2. **Command:** A Sqoop command specifies the table name (products), the target HDFS directory, and the columns to import [3].
3. **Process:** Sqoop creates a MapReduce job that parallelizes the import across multiple map tasks, reading chunks of the products table simultaneously and saving them as files in HDFS.

**10.**A. Discuss real world applications of Big Data in healthcare, finance, e-commerce, IOT and social media.

Big Data applications revolutionize key industries by analyzing massive, complex datasets to drive insights, efficiency, and personalization. Key applications include predictive patient care in healthcare,

real-time fraud detection in finance, hyper-personalized recommendations in e-commerce, predictive maintenance in IoT, and sentiment analysis for marketing on social media.

**1. Healthcare**

- **Preventive Care & Diagnostics:** MDPI and Touro University Illinois note that big data analyzes patient data for early detection of diseases and improved treatment plans.
- **Operational Efficiency:** DelveInsight states it optimizes hospital staffing by forecasting patient admission trends.
- **Drug Discovery:** Softeq explains that AI analyzes large genomic datasets to accelerate research.

**2. Finance**

- **Fraud Detection:** Turing reports that, according to , banks use big data to analyze transactions instantly, detecting unusual patterns to prevent fraud.
- **Risk Management:** ACTE Technologies mentions that financial institutions use it to evaluate credit risks and manage compliance.

**3. E-commerce**

- **Personalization:** Shopify explains that companies analyze browsing behavior and purchase history to provide tailored product recommendations.
- **Supply Chain Optimization:** Retailers use predictive analytics to manage inventory levels, forecasting demand to avoid stockouts.

**4. IoT (Internet of Things)**

- **Predictive Maintenance:** Sensors on machinery, as mentioned by ACTE Technologies, predict equipment failures before they occur, reducing downtime.
- **Smart Cities/Connected Homes:** HashStudioz notes that data from connected devices improves energy efficiency and urban management.

**5. Social Media**

- **Sentiment Analysis:** Shopify indicates that businesses analyze social media trends, comments, and interactions to understand customer sentiment.
- **Targeted Advertising:** Platforms like Facebook and Twitter use user data (likes, shares, demographics) to deliver highly targeted ads, as explained in ACTE Technologies.

**B. Explain real time analytics using Kafka and spark streaming. Discuss the ethical issues and challenges in Big data analytics.**

Real-time analytics leverages platforms like Apache Kafka and Spark Streaming to process data as it is produced, allowing for immediate insights and actions . This approach is crucial for applications such as fraud detection, IoT monitoring, and personalized user experiences.

**Real-Time Analytics with Kafka and Spark Streaming**

**Apache Kafka** acts as a high-throughput, fault-tolerant messaging system, serving as the central nervous system of the data pipeline .

- **Data Ingestion:** Producers send data (e.g., website clicks, sensor readings, financial transactions) to Kafka topics.
- **Data Durability:** Kafka stores records reliably and durably, enabling multiple consumers to process the same data stream at different times without data loss.

**Spark Streaming** (now structured as Structured Streaming in modern Apache Spark) processes the continuous data streams delivered by Kafka .

- **Stream Processing:** It reads data from Kafka and processes it using the full power of the Spark engine (e.g., SQL queries, machine learning algorithms).
- **Real-time Insights:** The processed results can be immediately stored in real-time databases, used to trigger alerts, or displayed on dashboards.

This combination allows organizations to ingest, process, and analyze vast quantities of data with very low latency, turning data into actionable intelligence almost instantly .

**Ethical Issues and Challenges in Big Data Analytics**

The power of big data analytics brings significant ethical responsibilities and practical challenges:

**Ethical Issues**

- **Privacy and Surveillance:** The ability to collect and analyze massive amounts of personal data raises major privacy concerns. Continuous monitoring of individuals can lead to a "surveillance society" where people's actions are constantly tracked and analyzed .
- **Algorithmic Bias and Discrimination:** Machine learning models trained on historical data may perpetuate or even amplify existing societal biases (e.g., racial, gender) . Biased algorithms in areas like hiring, credit scoring, or criminal justice can lead to unfair outcomes.
- **Data Ownership and Consent:** There is often a lack of transparency about what data is being collected and how it is used. Users often give consent via long, complex terms-of-service agreements they do not fully understand, challenging the notion of informed consent .

**Challenges**

- **Data Security and Governance:** Managing vast datasets increases the risk of data breaches. Robust governance frameworks are necessary to ensure data is handled responsibly and in compliance with regulations like GDPR or CCPA .

- **Data Quality and Reliability:** Big data often comes from disparate sources and can be messy, incomplete, or inaccurate. Poor data quality can lead to flawed analyses and poor decision-making .
- **Interpretability and Explainability:** Many advanced machine learning models (e.g., deep learning) used in big data are "black boxes," meaning their decision-making processes are difficult to understand. This lack of explainability makes it hard to trust the outcomes, especially in high-stakes applications