Introduction to Hadoop (T1): Introduction, Hadoop and its Ecosystem, Hadoop Distributed File System, MapReduce Framework and Programming Model, Hadoop Yarn, Hadoop Ecosystem Tools

Hadoop Distributed File System Basics (T2): HDFS Design Features, Components, HDFS User Commands.

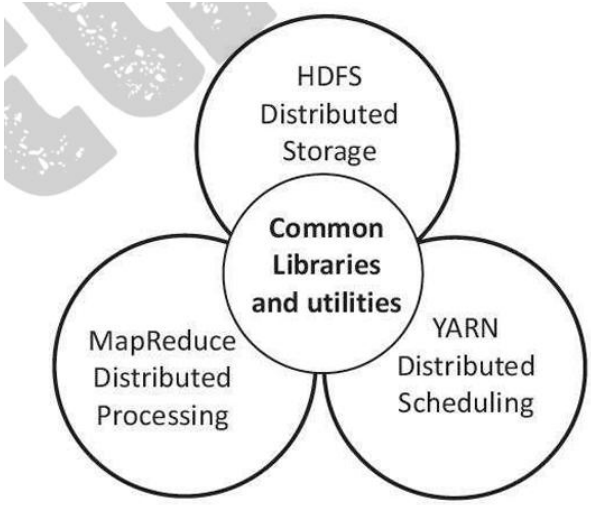
Essential Hadoop Tools (T2): Using Apache Pig, Hive, Sqoop, Flume, Oozie, HBase.

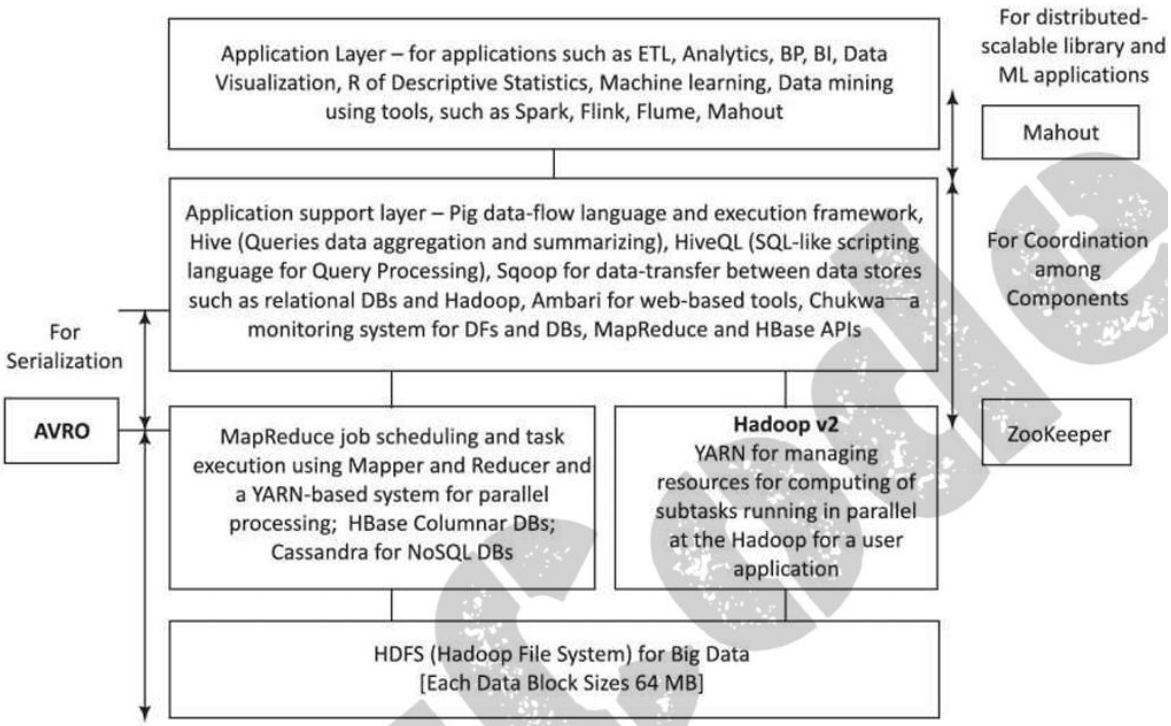
Module-2

a	What is Hadoop? Explain Hadoop eco-system with neat diagram
b	Explain with neat diagram HDFS Components.
c	Write short note on Apache hive.

a	Explain Apache Sqoop Import and Export methods.
b	Explain Apache Oozie with neat diagram.
c	Explain YARN application framework.

1) With Neat diagram explain Hadoop main components and ecosystem components





Hadoop main components and ecosystem components

1. HDFS (Hadoop Distributed File System)
    - The foundation of the Hadoop ecosystem, HDFS provides reliable, scalable, and distributed storage for Big Data. It stores data across multiple nodes in 64 MB data blocks. HDFS ensures fault tolerance by replicating each data block across multiple nodes, meaning data remains accessible even if individual nodes fail.
2. MapReduce
    - MapReduce is a programming model and processing engine that allows Hadoop to process large datasets in parallel. It involves:
        - Mapper: Breaks down tasks into smaller sub-tasks, which are distributed across multiple nodes.
        - Reducer: Aggregates the results from the Mapper to produce the final output.
    - MapReduce v1: The initial version of MapReduce, which handles batch processing on a large scale.
    - MapReduce v2: Enhanced in Hadoop 2, where it leverages YARN for better resource management and parallel processing.
3. YARN (Yet Another Resource Negotiator)
    - YARN is a resource management layer in Hadoop that allocates system resources like memory and CPU across different applications. By enabling multiple applications to run

simultaneously, YARN enhances Hadoop's ability to handle Big Data efficiently. It also allows for various tools and processing models to work within the same cluster.

4. Hadoop Common
    - This module includes essential libraries and utilities required by other Hadoop components. It provides the fundamental building blocks, like file-based data structures, serialization, and Java RPC (Remote Procedure Call) capabilities, necessary for Hadoop's distributed file system and data processing tasks.

Ecosystem Components

1. **Application Layer**
    - This layer includes tools and applications used for data analytics, ETL (Extract, Transform, Load), Business Intelligence (BI), Machine Learning, and Data Visualization. Popular tools include:
        - Spark: A fast data processing engine.
        - Flink: Suitable for real-time data processing.
        - Flume: Used for collecting and transferring large volumes of log data.
        - Mahout: Provides distributed and scalable machine learning libraries.

2. **Application Support Layer**
    - Provides additional support for data processing, storage, and monitoring within the Hadoop ecosystem. Key components include:
        - Pig: A scripting language for data processing.
        - Hive: A data warehouse tool that uses HiveQL (similar to SQL) for querying and managing data within Hadoop.
        - Sqoop: For transferring data between Hadoop and relational databases.
        - Ambari: A web-based tool for managing and monitoring Hadoop clusters.
        - Chukwa: A data collection system for monitoring large distributed systems.
3. **AVRO**
    - AVRO is a data serialization system within Hadoop, enabling efficient data storage and communication across different systems. AVRO serializes data into a compact binary format, making it faster and more space-efficient for data exchange.
4. **ZooKeeper**
    - ZooKeeper provides coordination and synchronization across distributed applications in Hadoop. It manages configuration information, naming, and distributed synchronization, ensuring reliable and organized interactions between different Hadoop components.

2) Brief out the features of Hadoop HDFS? Also Explain the functions of Name Node and data Node

Hadoop HDFS Features

1. Data Storage and Distribution: Data is distributed across multiple clusters, with each cluster containing several racks of DataNodes. Each DataNode has several data blocks that store portions of data files.
2. Replication: Data blocks are replicated across multiple DataNodes, with a default replication factor of 3, which enhances fault tolerance.
3. Data Block Size: The default data block size in HDFS is 64 MB, although it can be adjusted as needed.
4. File Operations: HDFS supports basic file operations, including create, append, delete, and renaming, though it follows a "write-once, read-many" approach.
5. Large Files: HDFS is optimized for handling very large files, with average sizes often exceeding 500 MB.
6. **Scalability**: HDFS can scale up to thousands of nodes, managing petabytes of data across distributed systems.
7. **Fault Tolerance**: Data is automatically replicated across multiple nodes, ensuring data recovery in case of hardware failures.
8. **High Throughput**: Designed for large-scale data processing, HDFS provides high throughput by allowing data access across multiple nodes simultaneously.
9. **Data Integrity**: Data blocks in HDFS have checksums to detect any corruption, which helps maintain data integrity.

Functions of NameNode and DataNode

1. **NameNode**:
   - Acts as the **master node** in HDFS, managing the metadata of all files and directories.
   - Stores information like the **directory structure, permissions, and file locations** of blocks.
   - Controls **access** to files and manages the **block allocation** to DataNodes.
   - Does not store the actual data but only the **metadata** about where data is stored on the cluster.
   - Responsible for **coordinating and managing** the DataNodes to ensure data availability and fault tolerance.
2. **DataNode**:
   - Acts as the **worker node** in HDFS, storing the actual data blocks on the physical hardware.

- Follows instructions from the NameNode to **store and replicate data blocks**.
- Regularly **reports** to the NameNode with information on the blocks it holds.
- Performs **block creation, deletion, and replication** as directed by the NameNode.
- Ensures **data integrity** by periodically verifying checksums of blocks and reporting any issues to the NameNode.

3) Explain HDFS Commands with example

1. **List Files in HDFS**

   To list files in a directory within HDFS, use the `-ls` command:

   ```bash
   $ hdfs dfs -ls /
   ```

   Example Output:

   ```bash
   Found 2 items
   drwxr-xr-x   - hdfs hdfs          0 2024-01-01 12:00 /app-logs
   drwxr-xr-x   - hdfs hdfs          0 2024-01-01 12:00 /user
   ```

2. **Create a Directory in HDFS**

   To create a new directory, use the `-mkdir` command:

   ```bash
   $ hdfs dfs -mkdir /example_dir
   ```

   Description: This creates a new directory `/example_dir` in HDFS.

3. **Copy Files from Local to HDFS**

   To copy a local file into HDFS, use the `-put` command:

   ```bash
   $ hdfs dfs -put local_file /example_dir/
   ```

   Description: This copies the file `local_file` from your local filesystem into the HDFS directory `/example_dir`.

4. **Retrieve Files from HDFS to Local**

   To copy a file from HDFS back to your local filesystem, use the `-get` command:

   ```bash
   $ hdfs dfs -get /example_dir/local_file ./local_copy
   ```

   Description: This retrieves `local_file` from HDFS and saves it as `local_copy` in your current local directory.

5. **Copy Files within HDFS**

   To duplicate a file within HDFS, use the `-cp` command:

   ```bash
   $ hdfs dfs -cp /example_dir/file1 /example_dir/file2
   ```

   Description: This copies `file1` to a new file `file2` within the `/example_dir` directory in HDFS.

6. **Delete a File in HDFS**

   To delete a file from HDFS, use the `-rm` command:

   ```bash
   $ hdfs dfs -rm /example_dir/local_file
   ```

   Description: This removes the file `local_file` from the `/example_dir` directory in HDFS.

7. **Check Disk Usage in HDFS**

   To check the disk usage of a directory or file in HDFS, use the `-du` command:

   ```bash
   $ hdfs dfs -du /example_dir/
   ```

   Description: This displays the amount of space consumed by the files in `/example_dir`.

4) Explain Hadoop Distributed file systems

HDFS is a core component of Apache Hadoop, designed for scalable and reliable storage of large datasets. It is inspired by the Google File System (GFS) and optimized for high-throughput access to large volumes of data.

Key Features of HDFS

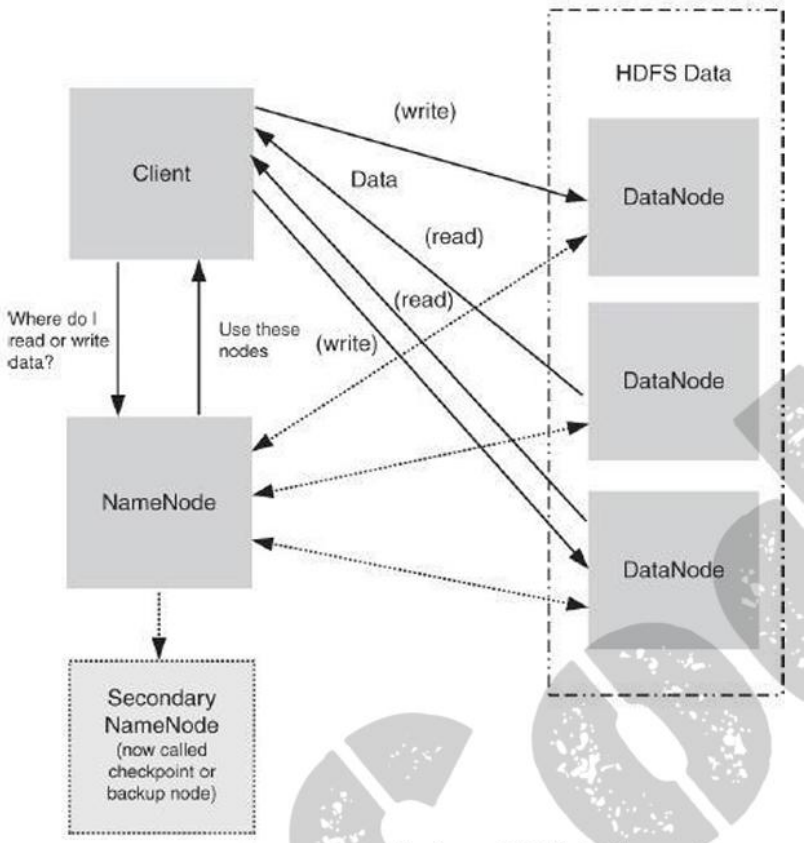
1. Distributed Storage:
   HDFS divides large files into smaller blocks (default size is 64 MB or 128 MB) and distributes these blocks across multiple nodes in a cluster.
2. Fault Tolerance:
   Each data block is replicated across multiple nodes (default replication factor is 3). If one node fails, the data can be retrieved from the replicas.
3. Write Once, Read Many:
   Data in HDFS is typically written once but can be read many times. This approach is ideal for big data analytics.
4. Scalability:
   HDFS is designed to scale out by adding more nodes to the cluster, enabling storage of petabytes of data.
5. Data Locality:
   Computation tasks are moved to the nodes where data resides, reducing network congestion and improving processing speed.
6. High Throughput:
   HDFS is optimized for sequential data access, enabling high-throughput data read/write operations.
7. Compatibility with Commodity Hardware:
   HDFS is built to run on inexpensive, off-the-shelf hardware, reducing costs.

Components of HDFS

1. NameNode:
   - Acts as the master node.
   - Manages metadata about the file system (e.g., file locations, directory structure, replication info).
   - Coordinates access to files and handles namespace operations like opening, renaming, and deleting files.
2. DataNode:
   - Acts as the slave node.
   - Stores actual data blocks and serves read/write requests from clients.
   - Sends periodic "heartbeats" to the NameNode to indicate its availability.
3. Secondary NameNode:
   - Assists the NameNode by creating periodic snapshots of its metadata.
   - It is not a failover node but helps in recovery in case of NameNode failure.

How HDFS Works

1. File Storage:
    - A file is split into blocks (e.g., 128 MB each).
    - These blocks are distributed across multiple DataNodes in the cluster.
2. Replication:
    - Each block is replicated across multiple nodes for fault tolerance.
    - The NameNode decides where replicas are stored.
3. File Retrieval:
    - A client requests a file from the NameNode.
    - The NameNode provides the locations of the blocks.
    - The client then retrieves the blocks directly from the DataNodes.



various system roles in an HDFS deployment

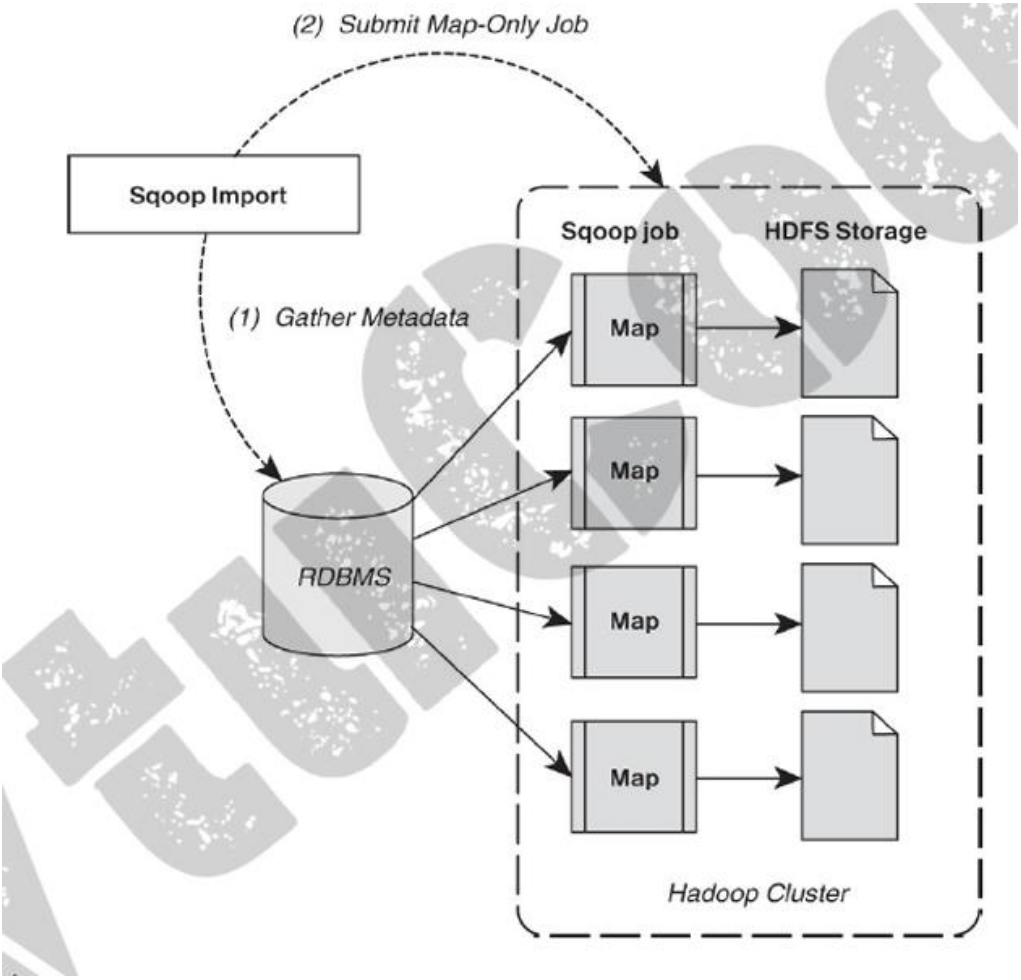
5) Explain Apache Sqoop import and export method with neat diagram

Apache Sqoop is a tool designed for efficiently transferring bulk data between relational databases (RDBMS) and the Hadoop ecosystem (primarily HDFS, Hive, and HBase). It leverages the parallel processing capabilities of MapReduce to handle large-scale data transfers, which improves performance and reduces the time taken for data movement.

1. Sqoop Import Method

The import method in Apache Sqoop is used to transfer data from a relational database (e.g., MySQL, Oracle, PostgreSQL) into HDFS, Hive, or HBase. The main goal of this operation is to move data from the RDBMS into the Hadoop ecosystem for processing and analysis.

- Process: Sqoop uses JDBC (Java Database Connectivity) to connect to the relational database. Once the connection is established, it extracts data from a specified table or the results of a query and moves the data into Hadoop. The data is imported in parallel using multiple mappers, which makes the process more efficient for large datasets.
- Custom Queries: Instead of importing entire tables, users can specify custom SQL queries to extract only the required data, allowing for more flexible and precise data extraction.
- Data Formats: During import, the data is typically transferred into formats compatible with Hadoop storage, such as plain text, Avro, or Parquet, depending on the target system (HDFS, Hive, or HBase).
- Use Case: The import method is particularly useful when you need to analyze large datasets stored in traditional relational databases in a Hadoop-based framework for big data processing.

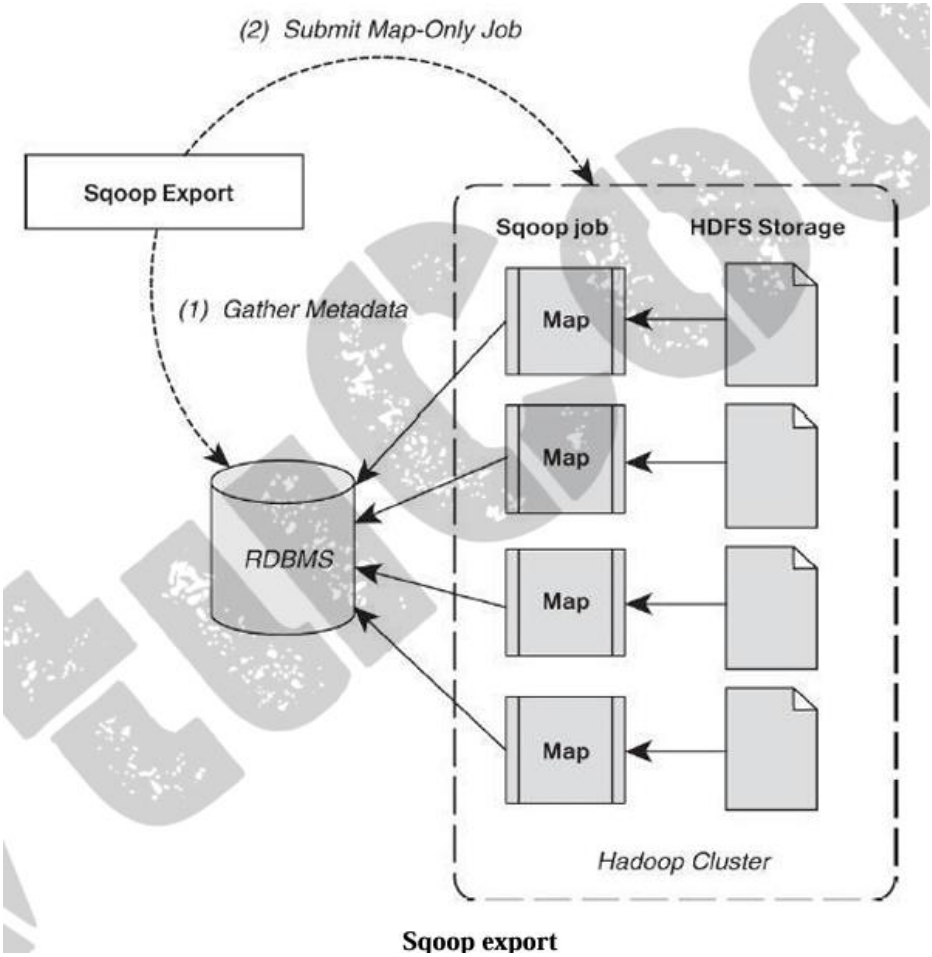


2. Sqoop Export Method

The export method allows users to transfer data from HDFS (or any other Hadoop ecosystem component like Hive or HBase) back into a relational database. This is useful when data processing or analysis is

completed within the Hadoop ecosystem, and the results need to be moved back to an RDBMS for further use or reporting.

- Process: The export process reads data from a specified HDFS directory (or other sources like Hive), formats it into rows that match the schema of the target database, and writes it to the relational database. Like imports, the data export is also handled in parallel using MapReduce, ensuring efficiency during the data transfer.
- Incremental Exports: Sqoop allows for incremental data export, which means that only the data that has changed or been added since the last export can be transferred. This feature is useful for regular data updates to the database without transferring the entire dataset again.
- Handling Large Data: For large datasets, Sqoop splits the data into manageable chunks, known as splits, and each chunk is processed by a separate mapper. This parallelism helps speed up the export process.
- Use Case: The export method is commonly used when you want to store the results of Hadoop-based processing (like data analytics or transformations) into a traditional RDBMS for reporting, backup, or further operations.



Sqoop export

6) Explain the working of the Hadoop map reduce frame work

The **Hadoop MapReduce framework** is a programming model and processing engine used to process large datasets in a distributed computing environment, leveraging the power of Hadoop's distributed file

system (HDFS). The model divides tasks into two phases: **Map** and **Reduce**, enabling parallel processing across a cluster of machines.

Here's a breakdown of how the Hadoop MapReduce framework works:

---

1. Input Data

- The data to be processed by MapReduce is stored in **HDFS** (Hadoop Distributed File System).
- A typical input for a MapReduce job is a file or a directory in HDFS, which is divided into blocks (usually 64 MB or 128 MB).
- Each block of data is processed by a different **map task**, allowing for **parallel processing**.

---

2. Map Phase (Mapper)

- The **Map** phase processes the input data by splitting it into **key-value pairs**. Each chunk of data is processed by a **Mapper**, which is a user-defined function.

Steps in the Map Phase:

- **Input Splitting**: The input data is split into smaller chunks (blocks) by the **InputFormat**. These chunks are assigned to mappers for parallel processing.
- **Mapping**: Each **Mapper** processes a chunk of input data and generates a set of intermediate key-value pairs. These pairs are not the final output but represent the results of the processing step.
- **Output of Mapper**: After processing, the mapper produces **key-value pairs** as output, which are shuffled and sorted by key before passing to the next phase.

Example: If you are counting the number of occurrences of words in a document, the mapper would process the text, outputting key-value pairs like:

- ("word", 1), where "word" is the key and 1 is the value.

---

3. Shuffle and Sort Phase

- Once all the **map tasks** have completed, the Hadoop framework performs a **shuffle and sort** operation:
    - **Shuffle**: The system groups the intermediate data (key-value pairs) by key. All pairs with the same key are grouped together.
    - **Sort**: The groups are sorted by the key so that each reducer gets all values corresponding to the same key in sorted order.
- This phase is handled automatically by Hadoop and ensures that data is prepared for the reduce phase.

---

4. Reduce Phase (Reducer)

- The **Reduce** phase takes the output from the Map phase (key-value pairs) and processes it to generate the final output. The **Reducer** is responsible for aggregating, summarizing, or transforming the data based on the keys.

Steps in the Reduce Phase:

- **Input to Reducer**: The **Reducer** receives the intermediate key-value pairs. For each unique key, the reducer gets a list of all values associated with that key.
- **Processing**: The reducer performs the required computation on the values associated with the key. This could include operations like summing values, counting occurrences, or other forms of aggregation.
- **Output**: The final output of the reducer is the processed key-value pairs, which are written to **HDFS** as the result of the MapReduce job.

Example: Continuing with the word count example, if the mapper outputs ("word", 1) for each occurrence of a word, the reducer will take all ("word", 1) pairs and sum them to produce the final count for each word.

---

5. Final Output

- After the reduce phase, the final output is written back to **HDFS**, typically in files where each line contains a key-value pair representing the final result.

7) Explain Apache Pig , Hive , Flume , Hbase, Oziee?

1. Apache Pig

Apache Pig is a high-level platform built on top of Hadoop for processing and analyzing large datasets. It is designed to work with Hadoop and uses a language called **Pig Latin**, which is a simplified scripting language similar to SQL but with additional data transformation capabilities. Pig is widely used for data transformation tasks like filtering, grouping, and joining large datasets.

Key Features:

- **Data Flow Language**: Pig Latin allows users to write complex data processing tasks without having to write a lot of low-level code.
- **Extensibility**: Pig allows users to write their own functions (UDFs) for custom operations.
- **Optimization**: Pig can optimize the execution of queries by automatically converting them into MapReduce jobs for efficient parallel processing.
- **Schema Flexibility**: It can handle both structured and semi-structured data formats, making it suitable for big data processing tasks.

Use Cases:

- ETL (Extract, Transform, Load) processes.

- Data preparation for analytics and reporting.
- Data cleaning and transformation for downstream applications.

---

2. Apache Hive

Apache Hive is a data warehousing and SQL-like query language system built on top of Hadoop. It provides a high-level interface for querying and managing large datasets stored in **HDFS** using a language called **HiveQL**, which is very similar to SQL. Hive is used for data summarization, querying, and analysis.

Key Features:

- **SQL-Like Query Language**: Hive uses **HiveQL** (a query language similar to SQL) to interact with data, making it easier for analysts familiar with SQL to use.
- **Data Warehousing**: It provides features for partitioning and bucketing data to improve query performance.
- **Extensibility**: Users can define their own functions (UDFs) to process data in custom ways.
- **Support for Large Datasets**: Hive is designed to handle petabytes of data across a distributed system.
- **Integration with BI Tools**: Hive integrates with various Business Intelligence (BI) tools, providing a bridge for Hadoop and data analysts.

Use Cases:

- Large-scale data analytics.
- Querying structured data with SQL-like syntax.
- Data transformation and aggregation.

---

3. Apache Flume

Apache Flume is a distributed and reliable service designed for collecting, aggregating, and transporting large volumes of streaming data into Hadoop. It is mainly used for ingesting log data and streaming data from different sources like web servers, social media, and sensor data into HDFS, Hive, or HBase.

Key Features:

- **Distributed Architecture**: Flume is designed to scale horizontally across many nodes, making it suitable for high-volume data ingestion.
- **Reliable Data Transfer**: It provides fault-tolerant mechanisms to ensure that data is reliably transferred, even in case of network failures.
- **Flexible Data Flow**: Flume supports multiple **sources**, **channels**, and **sinks**, making it highly customizable for different use cases.
- **Support for Multiple Data Sources**: It can aggregate data from multiple sources such as log files, social media feeds, and more.

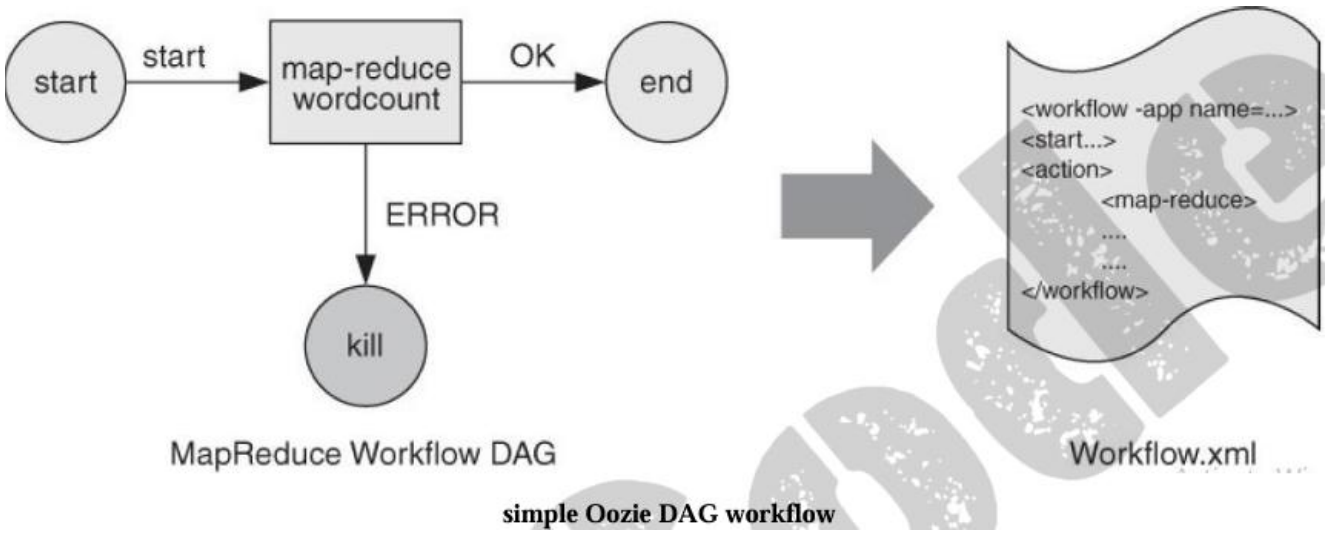
Use Cases:

- Ingesting large amounts of streaming data like log files into Hadoop.
- Real-time data processing.
- Aggregating and transporting sensor data, social media streams, etc.

---

4. Apache Oozie

Apache Oozie is a workflow scheduler system designed to manage and coordinate Hadoop jobs. It allows users to define a series of actions, such as MapReduce jobs, Pig scripts, Hive queries, and others, and manage their execution in a workflow. Oozie is essential for automating and scheduling the execution of complex data processing workflows.



simple Oozie DAG workflow

Key Features:

- **Workflow Management**: Oozie workflows are represented as Directed Acyclic Graphs (DAGs), where each action (e.g., MapReduce job, Pig job) depends on the completion of the previous action.
- **Time and Data Triggers**: Workflows can be triggered based on time (cron-like scheduling) or when new data arrives.
- **Coordination**: Oozie can coordinate various Hadoop jobs that depend on each other, ensuring that jobs run in a sequence or in parallel as required.
- **Error Handling**: It provides built-in mechanisms to handle errors and retries in case of job failures.

Use Cases:

- Automating the execution of complex data processing workflows.
- Scheduling and managing data processing tasks, such as running MapReduce, Pig, and Hive jobs in sequence.
- Coordinating the dependencies between jobs for large-scale data pipelines.

---

5. Apache HBase

Apache HBase is a distributed, scalable, NoSQL database built on top of Hadoop and HDFS. It is designed to handle massive amounts of sparse data, which makes it suitable for applications that require random read and write access to large datasets. HBase is modeled after Google's **Bigtable** and is typically used for real-time read/write access to data.

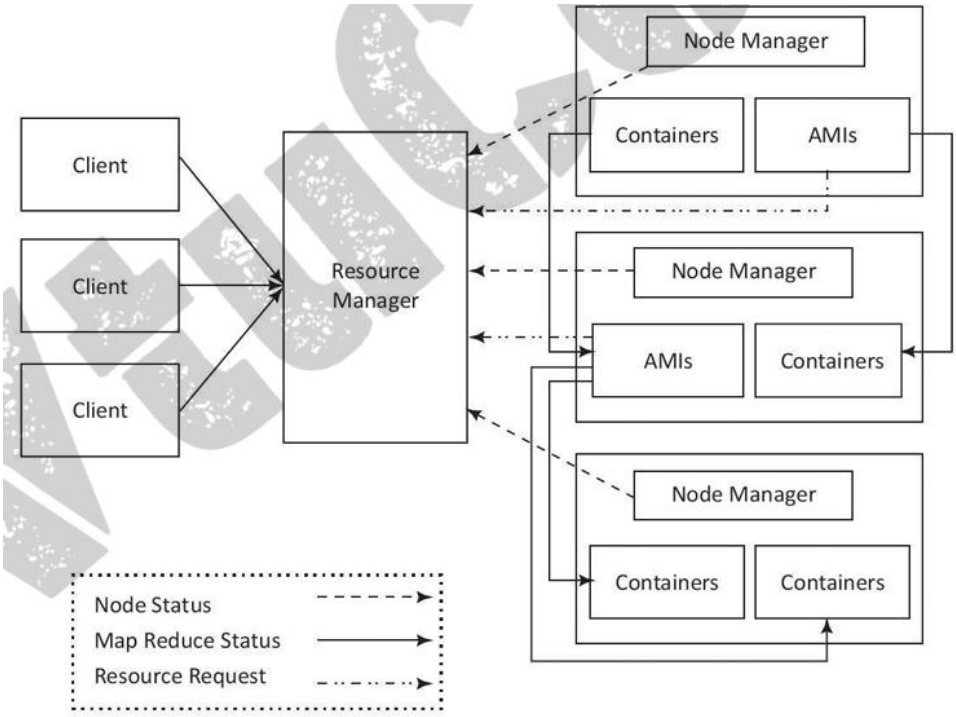
Key Features:

- **Column-Oriented**: HBase stores data in **tables** with rows and columns, but it is **column-family-based**. This allows for efficient storage and retrieval of sparse data.
- **Scalability**: HBase can handle petabytes of data and scales horizontally by adding more nodes to the cluster.
- **Real-Time Access**: It supports real-time, low-latency access to data, making it ideal for applications requiring fast read and write operations.
- **Integration with Hadoop**: HBase integrates seamlessly with **HDFS**, storing its data in HDFS and allowing the use of Hadoop's ecosystem tools for processing.

Use Cases:

- Storing and managing large amounts of unstructured data (e.g., sensor data, clickstream data).
- Real-time data processing applications (e.g., recommendation engines, real-time analytics).
- Storing large datasets for random, low-latency access (e.g., time-series data, logs).

YARN APPLICATION FRAMEWORK



YARN based execution model

YARN as a Resource Management Platform:

- YARN is responsible for managing computational resources within a Hadoop cluster.
- It allocates resources like CPU, memory, and storage to applications running on the cluster.
- This allows for efficient utilization of resources and enables multiple applications to run concurrently.

Hadoop 2 YARN Execution Model:

- Components:
  - Client: Submits application requests to the Resource Manager.
  - Resource Manager (RM): The central authority in YARN, it manages resource allocation and schedules application execution.
  - Node Manager (NM): Manages resources on individual nodes within the cluster.
  - Application Master (AM): Responsible for negotiating resource allocation with the RM and managing the execution of a specific application.
  - Container: An isolated unit of resources (CPU, memory, etc.) allocated to an application for running tasks.

Execution Flow:

1. Client Submits Application: The client submits an application request (e.g., a MapReduce job) to the Resource Manager.
2. Resource Manager Starts Application Master: The RM allocates resources for an Application Master on a Node Manager.
3. Application Master Negotiates Resources: The Application Master negotiates with the RM for the resources required by the application (containers).
4. Resource Manager Allocates Containers: Based on availability and scheduling policies, the RM allocates containers on different nodes.
5. Application Master Launches Tasks: The Application Master sends instructions to the allocated containers on the Node Managers to launch and run application tasks.
6. Node Managers Manage Containers: Node Managers monitor the containers running on their nodes and report their status to the Application Master.
7. Task Execution and Monitoring: The application tasks execute within the allocated containers. The Application Master monitors their progress and handles failures.