1.Mention Hadoop distribution? Difference between CDH and CDP

Hadoop is an open-source software framework for distributed storage and processing of large data sets. There are several Hadoop distributions available.

Hortonworks Data Platform (HDP), MapR, Cloudera Distribution of Hadoop (CDH) and Cloudera Data Platform (CDP), both developed by Cloudera, a prominent player in the big data and analytics space.

Cloudera Distribution for Hadoop (CDH):

CDH has been a longstanding and popular Hadoop distribution developed by Cloudera.

It encompasses various Apache Hadoop ecosystem components such as HDFS, MapReduce, YARN, Hive, HBase, Spark, Impala, and many others.

CDH follows a traditional release cycle, offering both major and minor updates to ensure stability and compatibility across the Hadoop ecosystem.

CDH is well-suited for on-premises deployments and traditional big data architectures.

Cloudera Data Platform (CDP):

CDP is a newer offering from Cloudera designed to provide a more comprehensive and unified data management platform.

Unlike CDH, CDP is built to operate in both on-premises and cloud environments, providing flexibility for hybrid and multi-cloud deployments.

CDP integrates a broader range of data management and processing services beyond Hadoop, including Apache Kafka, Apache Spark, and others, offering enhanced capabilities for data analytics, streaming, and machine learning.

Additionally, CDP incorporates features for data governance, security, and management to address enterprise-level requirements for data privacy and compliance.

CDP aims to simplify the deployment and management of big data and analytics workloads across different environments, offering a more modern and unified approach compared to traditional Hadoop distributions like CDH.

2.Explain Hadoop Architecture

Hadoop is an open source framework for writing and running distributed applications that process large data.

-it provides both storage and computational capabilities.

-it is deployed on a cluster of machines which are grouped in racks.

Hadoop Components: HDFS ,MapReduce,YARN:resource negotiator,common utilities or Hadoop common

1. HDFS-Hadoop Distributed File System

-responsible for storage adopted from Google's file system (GFS). HDFS built to support only very large files and not small ones. store millions of files. designed to support functionality of mapreduce. based on write once and read many times.(reading is used much more than writing)

Replication:

-each file is split into blocks of fixed size (64/128MB etc)(size is not fixed. it could be changed)

-each block is replicated (3replicas by default)( replicas can change. we can have 2 replicas / 4 replicas)

-these replicas are distributed to different machines in the cluster

Advantages of having replicas:

-if a replica on a machine is lost or fails it's not a big problem because the same copy is in another machine, so that we can use that to complete our process.

-having different replicas (copies) means multiple machine failures are easily tolerated

-having different replicas means several versions of data are available for reading, so a data block can be read that is closest to an application on the network, which in turn speeds up processing.

-HDFS is designed to track and manage the number of available replicas of a block, so if the no of copies of a block drops bcoz of failure, the file system automatically makes a new copy from one of the remaining replicas.

HDFS Cluster: Name Node, Data Node (Node-computer, we have cluster of different computers)

-HDFS uses master-slave architecture, master is name node nd slave is data node.

-we have one/two name nodes and many data nodes.

Name Node:-Master mind(imp node is the whole file system)

-if it fails whole system fails.

-older version have one name node consistuted Single point failure SPOFso if name node fails the whole system fails

-this was handled in new versions

-if name node fails the data resides on data node is still there.except that we dont know which copy is in which place.each file where it is located

-there are techniques to recover the data whre it is.. but these are complicated

-name node stores filesystem metadata,stores file to block map and provides a global picture of the file system

-HDFS has atleast one name node,it can have two(active / standby) to avoid system failure.this release is called HDFS-High Availability

Data Node:

-where chuncks of data-file content-are stored.system has many data nodes.

-is has direct local access to one or more disks.

-each data node regularly contact name node to report thier status.sends HEARTBEAT to tell namenode that everything is allright ,its up nd running.

-at any given time,name node has a complete view of all datanodes in the cluster,their current health-whether they are down or up nd what blocks available.

-if data node fails name node knows something happen.the name node will do somthing to recover that.

HDFS write operation:

-client will contact name node(1st contact of file system) to tell that he wants to write on the file.

-name node contains the location adress of all data nodes.

-name node sends the location to the client to write the files.

-client will contact the 1st data node that is decided by name node.if it has 3 replicas

-in the 1st data node the data is written and1st data node contacts 2nd data node and write blocks of

data and 2nd data node contacts 3rd data node nd write data and it contacts previous data node nd client

to inform the file is stored succesfully.

HDFS READ OPERATION: easy one

-client contact name node. it will tell client the nearest available data node and client directly contacts data node.

2. MapReduce

-responsible for computing part of HADOOP

-it is designed to distribute computation over multiple nodes.

-it has two constructs: Mappers and reducers

-all computations are expressed in terms of map and reduce which manipulate key-value pairs.

-MapReduce operates at higher level where the programmer thinks in terms of key and value pairs and data flow is implicit.

-the system has many mappers nd atleast one reducer.

-main advantage-once programer expreses computation interms of mapping nd reducing,scalling the application to run over hundreds /thousands/even tenthousands of machines

in a cluster is a simple configuration change.so this model is attractive.

-sutable to analyse dataset in a batch fasion.

-works well on unstructured or semi-structured data bcoz it is based on concept of key value pairs.

-based on share nothing architecture.bcoz each computation are independt on one another.

-this share-nothing property enables mapreduce to run a program across thousands or millions of unreliable machines in parallel nd to complete a task in a very short time.

How MapReduce Works.

5 steps.

1.splitting

2.mapping(distriution)

3.shuffling and sorting

4.reducing

5.Aggregating.

• Hadoop Daemons - Job Tracker - Task Tracker

daemon is a operations running on background application that user have no control over that.

Job Tracker :its a mediator between application and hadoop

-job tracker monitor whole process.once code is submitted to the cluster,the job tracker determines execution plan by determining which files to

process , assigns nodes to different tasks,and monitors all tasks as they're running.

-if a task fails,the job tracker will automatically relaunch it , possibly on a different node.

-there is only one job trackert daemon per hadoop cluster.

-it oversees the overall execution of mapreduce.

Task Trackers:

-task tracker manages the execution of individual tasks on each slave node.

-each task tracker is responsible for executing the individual tasks that the job tracker assigns.

-usually there is a single task tracker per slave node.

-task tracker constantly communicates with the job tracker.if JT fails to recieve a heartbeatfrom a TT,

it will assume TT has crashed nd will resubmit the corresponding task to other nodes in the cluster.

3. YARN (Yet Another Resource Negotiator)-management component of hadoop (HDFS-storage,Mapreduce-computing,YARN-management component)

-in hadoop 1- we dont have yarn,mapreduce itself takes care of management of resources there.

-in laterversions hadoop 2.0 mapreduce underwent some modifications resulting in MAPREDUCE 2.0 or YARN

-YARN is seperated into 2parts.

Resource Manager:responsible for tracking and arbitrating resources among applications

Node Manager:responsible for launching tasks and monitoring the resource usage per slave node.

both form the data computation framework.

Resource manager -

-responsible for creating nd allocating resources to multiple applications

-we dont have one application running on hadoop cluster.we have running different applications at the same time on a cluster.

we dont dedicate whole cluster for running one application.

-instead of having one centralized jobtracker ,each application has its own 'jobtracker' called application master.

that runs on one of the workers of the cluster.

Yarn ,there is no centralized job tracker.

Node Manager

-replaces task tracker.node manager is more advanced nd generic as it runs different applications,dedictated for application,in an aplication container.

-bcoz it uses node manager,its ability to run arbitrary applications,one can write non-mapreduce applications and run them on YARN.

4. Hadoop common or Common Utilities

Hadoop common or Common utilities are nothing but our java library and java files or we can say the java scripts that we need for all the other components present in a Hadoop cluster. these utilities are used by HDFS, YARN, and MapReduce for running the cluster. Hadoop Common verify that Hardware failure in a Hadoop cluster is common so it needs to be solved automatically in software by Hadoop Framework.

3.Configuration files used during hadoop installation

The key configuration files used during Hadoop installation are:

1. core-site.xml

Specifies the Hadoop file system URI (fs.defaultFS) which typically points to the NameNode's address and port. Defines the default file system scheme (e.g. hdfs) to use.

2. hdfs-site.xml

Specifies configuration options for HDFS daemons like NameNode, DataNode, etc. Sets replication factor (dfs.replication) which determines how many replicas are maintained for each HDFS block.

Configures NameNode metadata storage directories.

3. yarn-site.xml

Configures parameters for YARN daemons like ResourceManager, NodeManager. Sets the ResourceManager address and scheduler to use. Specifies application log directories, node manager memory settings, etc.

4. mapred-site.xml

Specifies MapReduce job configuration parameters like the JobHistory server details.

Configures the framework for running MapReduce jobs (pre-YARN).

5. workers (or slaves)

A simple file that lists the hostnames of DataNodes/NodeManagers in the cluster.

6. log4j.properties

Configures logging levels and log rolling policies for Hadoop daemons.

These XML files are typically located in the $HADOOP\_HOME/etc/hadoop directory and need to be properly configured based on the cluster setup before starting Hadoop services.

4.Difference between Hadoop fs and hdfs dfs

The difference between Hadoop fs and HDFS dfs lies in their scope and functionality:

Hadoop fs

* Hadoop fs (or hadoop.fs) refers to the Hadoop FileSystem API, which is a generic abstraction layer for accessing different file systems.
* It provides a unified interface for interacting with various file systems like local file system, HDFS, Amazon S3, Azure Blob Storage, etc.
* Applications can use the same code to read/write data from/to different file systems by specifying the appropriate URI scheme (e.g., [file:///](file://\), hdfs://, s3a://, etc.).

HDFS dfs

* HDFS dfs (or hdfs.dfs) specifically refers to the Hadoop Distributed File System (HDFS), which is the primary distributed file system used in Hadoop clusters.
* It is designed to store and manage large datasets across multiple machines in a fault-tolerant and scalable manner.
* HDFS dfs provides APIs and commands for interacting with the HDFS file system, such as creating directories, uploading/downloading files, checking file status, and more.
* It is optimized for batch processing of large datasets with high throughput for data access patterns like write-once, read-many-times.

5.Difference between Hadoop 2 and Hadoop 3

Hadoop 2

JAVA 7 is the minimum compatible version.

Replication is the only way to handle fault tolerance which is not space optimized. 3x Replication Scheme is used.

HDFS balancer is used for Data Balancing.

200% of HDFS is consumed in Hadoop 2.x

Uses timeline service with scalability issue.

Limited Scalability, can have upto 10000 nodes in a cluster.

Manual intervention is needed for the namenode recovery.

Compatible File System - HDFS(default), FTP, Amazon S3 and Windows Azure Storage Blobs (WASB) file system.

Hadoop 3

JAVA 8 is the minimum compatible version.

Erasure coding is used for handling fault tolerance. Uses eraser encoding in HDFS.

Intra-data node balancer is used which is called via HDFS disk-balancer command-line interface.

50% used in Hadoop 3.x means we have more space to work.

Improve the time line service along with improving scalability and reliability of this service.

Scalability is improved, can have more then 10000 nodes in a cluster.

No need of Manual intervention for name node recovery.

Compatible File System- All file systems including Microsoft Azure Data Lake filesystem.

6.What is replication factor ? why its important

Replication ensures the availability of the data. Replication is nothing but making a copy of something and the number of times you make a copy of that particular thing can be expressed as its Replication Factor. As we have seen in File blocks that the HDFS stores the data in the form of various blocks at the same time Hadoop is also configured to make a copy of those file blocks. By default the Replication Factor for Hadoop is set to 3 which can be configured means you can change it Manually as per your requirement like in above example we have made 4 file blocks which means that 3 Replica or copy of each file block is made means total of 4×3 = 12 blocks are made for the backup purpose.

we need this replication for our file blocks because for running Hadoop we are using commodity hardware (inexpensive system hardware) which can be crashed at any time. We are not using a supercomputer for our Hadoop setup. That is why we need such a feature in HDFS which can make copies of that file blocks for backup purposes, this is known as fault tolerance.

After making so many replica’s of our file blocks we are wasting so much of our storage but for the big brand organization the data is very much important than the storage. So nobody care for this extra storage.

You can configure the Replication factor in you hdfs-site.xml file.

7.What if Datanode fails?

Namenode periodically receives a heartbeat and a Block report from each Datanode in the cluster. Every Datanode sends heartbeat message after every 3 seconds to Namenode. The health report is just information about a particular Datanode that is working properly or not. In the other words we can say that particular Datanode is alive or not. A block report of a particular Datanode contains information about all the blocks on that resides on the corresponding Datanode. When Namenode doesn’t receive any heartbeat message for 10 minutes(ByDefault) from a particular Datanode then corresponding Datanode is considered Dead or failed by Namenode. Since blocks will be under replicated, the system starts the replication process from one Datanode to another by taking all block information from the Block report of corresponding Datanode. The Data for replication transfers directly from one Datanode to another without data passing through Namenode.

8.What if Namenode fails?

if the NameNode fails, it can have serious consequences for the Hadoop Distributed File System (HDFS) and the overall Hadoop cluster. Here are the key points regarding NameNode failure:

Single Point of Failure: In older versions of Hadoop, there was only a single NameNode, which constituted a Single Point of Failure (SPOF). If the NameNode failed, the entire HDFS would become unavailable, and no data could be read or written.

Data Availability: While the actual data blocks are stored on the DataNodes, the NameNode stores the metadata and the file-to-block mapping information. If the NameNode fails, the system loses track of where the data blocks are located, making the data inaccessible, even though the blocks themselves are still present on the DataNodes.

Recovery Mechanisms: To address the issue of NameNode failure, Hadoop introduced High Availability (HA) for the NameNode in later versions. In an HA setup, there are two NameNodes: an Active NameNode and a Standby NameNode. The Standby NameNode maintains an up-to-date copy of the metadata and can take over as the Active NameNode if the primary NameNode fails.

Data Recovery: Even with HA, if both the Active and Standby NameNodes fail, the data on the DataNodes is still present, but the system loses track of the file-to-block mapping. In such cases, complicated recovery procedures are required to reconstruct the metadata and make the data accessible again.

9.Why is block size 128 MB ? what if I increase or decrease the block size

The 128 MB block size is a trade-off between minimizing the overhead of managing too many small files and avoiding excessively large files that could impact data transfer times. It is designed to work well with the typical hardware configurations of commodity servers and disk drives.

Smaller block sizes would result in more metadata overhead and more seek operations during reads/writes, impacting performance.

Larger block sizes could lead to inefficient use of storage space, as the last block of a file may be significantly underutilized.

But it is configurable and can be increased or decreased. The choice of block size has implications on the performance and storage efficiency of the HDFS system.

Increase the block size:

Increasing the block size can improve data throughput for large sequential reads and writes, as there will be fewer block boundaries to cross.

However, larger block sizes can also lead to increased storage overhead, as the last block of a file may have a significant portion of unused space.

With larger blocks, the cost of transferring a single block across the network or disk increases, potentially impacting performance for smaller files or random access patterns.

Decrease the block size:

Decreasing the block size can reduce the storage overhead for small files, as the last block will have less unused space.

Smaller block sizes can also improve performance for random access patterns or workloads involving many small files, as there will be less data to transfer per block.

However, smaller block sizes can lead to increased metadata overhead, as there will be more blocks to manage for the same amount of data.

Additionally, smaller block sizes can result in more seek operations during reads and writes, potentially impacting performance for large sequential workloads.

10.Small file problem

HDFS is designed to handle large files efficiently, it can face challenges with a large number of small files due to storage inefficiency, metadata overhead, seek time overhead, and performance issues with NameNode operations. Various techniques, such as merging small files, using appropriate file formats, archiving, or leveraging object stores, can help mitigate the small file problem in HDFS

11.What is Rack awareness?

The rack is a physical collection of nodes in our Hadoop cluster. A large Hadoop cluster is consists of many Racks. With the help of this Racks information, Namenode chooses the closest Datanode to achieve maximum performance while performing the read/write information which reduces the Network Traffic. A rack can have multiple data nodes storing the file blocks and their replica’s. The Hadoop itself is so smart that it will automatically write a particular file block in 2 different Data nodes in Rack. If you want to store that block of data into more than 2 Racks then you can do that.

A large Hadoop cluster contains multiple Racks, in each rack there are lots of data nodes are available. Communication between the Datanodes that are present on the same rack is quite much faster than the communication between the data node present at the 2 different racks. The name node has the feature of finding the closest data node for faster performance for that Name node holds the ids of all the Racks present in the Hadoop cluster. This concept of choosing the closest data node for serving a purpose is Rack Awareness.

12.What is SPOF ? how its resolved ?

Name node is the important node in the whole file system. It is called the master mind of the system. If the name node fails, the whole system fails. In older versions of Hadoop, because we had one name node, if it fails, whole system is failed which is a single-point-of-failure (SPOF), but in later versions, this problem was solved. It is important to understand that if name node fails, the data that resides on the data node is still there. the data itself is not lost but the name node fails, we dont know what is going on and where the data is. except that we dont know which file where it is located. there are techniques to recover the data by accessing metadata reconstituting the whole system. To avoid system failure, HDFS have two name nodes (active & stand-by) which is called HDFS-High Availability.

13.Explain zookeeper?

Zookeeper is a coordination service for distributed application that enables synchronization across a cluster. Zookeeper in Hadoop can be viewed as centralized repository where distributed applications can put data and get data out of it. It is used to keep the distributed system functioning together as a single unit, using its synchronization, serialization and coordination goals. For simplicity's sake Zookeeper can be thought of as a file system where we have znodes that store data instead of files or directories storing data. Zookeeper is a Hadoop Admin tool used for managing the jobs in the cluster.

14.Difference between -put and -CopyFromLocal?

both the -put and -copyFromLocal commands are used to copy files from the local file system to HDFS (Hadoop Distributed File System).

-put is a traditional command available in earlier versions of Hadoop (such as Hadoop 1.x).

-copyFromLocal is a newer command introduced in later versions of Hadoop (such as Hadoop 2.x and later). It serves the same purpose as –put.

15.What is erasure coding?

Erasure coding is used to recover the data when the computer hard disk fails. It is a high-level RAID(Redundant Array of Independent Disks) technology used by so many IT company’s to recover their data. Hadoop file system HDFS i.e. Hadoop Distributed File System uses Erasure coding to provide fault tolerance in the Hadoop cluster. Since we are using commodity hardware to build our Hadoop cluster, failure of the node is normal. Hadoop 2 uses a replication mechanism to provide a similar kind of fault-tolerance as that of Erasure coding in Hadoop 3.

In Hadoop 2 replicas of the data, blocks are made which is then stored on different nodes in the Hadoop cluster. Erasure coding consumes less or half storage as that of replication in Hadoop 2 to provide the same level of fault tolerance. With the increasing amount of data in the industry, developers can save a large amount of storage with erasure coding. Erasure encoding minimizes the requirement of hard disk and improves the fault tolerance by 50% with the similar resources provided.

16.What is speculative execution?

Speculative execution is a technique used in distributed computing systems, particularly in frameworks like Hadoop, to improve job completion times and overall system efficiency.

It involves running multiple copies of the same task simultaneously on different nodes in a distributed environment. The main goal of speculative execution is to mitigate performance bottlenecks caused by slow-running or straggling tasks

When a job is submitted to a distributed computing system, tasks are divided and assigned to different nodes (or workers) for execution. If a particular task is progressing significantly slower than expected (due to factors like resource contention or hardware issues), speculative execution kicks in.

A speculative task is a duplicate copy of a task that is already running but appears to be slow. The system identifies such tasks based on their progress and resource consumption.

The system starts executing the speculative task on another node, concurrently with the original task. The system does not wait for the slow task to complete before initiating the speculative task.

Whichever task (original or speculative) completes first successfully becomes the result. The result of the other task is discarded to prevent duplicate processing. Speculative execution helps in reducing job completion times by addressing stragglers or slow tasks effectively. It improves overall system throughput and resource utilization in distributed computing environments.

17.Explain Yarn Architecture

YARN (Yet Another Resource Negotiator) is the resource management layer of Apache Hadoop. It's responsible for managing and allocating resources in a Hadoop cluster, allowing multiple data processing frameworks to run concurrently on the same infrastructure. YARN separates the resource management and job scheduling/monitoring functions of MapReduce, enabling a more flexible and efficient use of cluster resources. Here's an overview of the YARN architecture:

ResourceManager (RM):

The ResourceManager is the master daemon in the YARN architecture. It consists of two main components: the Scheduler and the ApplicationManager.

Scheduler: The Scheduler is responsible for allocating resources to applications based on resource requirements, queue configurations, and policies. It manages resources across all the nodes in the cluster.

ApplicationManager: The ApplicationManager is responsible for accepting job submissions, negotiating appropriate resources with the ResourceManager, and managing the lifecycle of applications.

NodeManager (NM):

The NodeManager is a per-node daemon that runs on each node in the cluster. It's responsible for managing resources on individual nodes and monitoring their health.

Resource Manager: The Resource Manager component in the NodeManager is responsible for monitoring resource usage (CPU, memory, etc.) on the node and reporting it back to the ResourceManager.

ApplicationManager: The ApplicationManager component in the NodeManager is responsible for launching and managing application containers on the node.

ApplicationMaster (AM):

The ApplicationMaster is a framework-specific component responsible for managing the execution of a specific application. Each application running on YARN has its own ApplicationMaster.

The ApplicationMaster negotiates resources with the ResourceManager, requests containers for executing tasks, monitors their progress, handles failures, and releases resources upon completion.

Application-specific logic, such as task scheduling, fault tolerance, and data locality optimization, is implemented within the ApplicationMaster.

Container:

A container is a fundamental unit of resource allocation in YARN. It represents a set of allocated resources (CPU, memory, etc.) on a node where application tasks are executed.

Each ApplicationMaster negotiates with the ResourceManager to obtain one or more containers for executing its tasks.

Containers provide isolation and resource enforcement for tasks running on the node.

18.How does ApplicationManager and Application Master differ

ApplicationManager:

The ApplicationManager is a component of the ResourceManager in YARN.

It is responsible for accepting job submissions from clients, negotiating resources with the ResourceManager on behalf of the application, and managing the lifecycle of applications.

It handles the initial submission of an application, requests resources from the ResourceManager based on the application's requirements, and tracks the application's progress until completion.

Each application submitted to YARN has its own ApplicationManager instance associated with it.

ApplicationMaster:

The ApplicationMaster is a framework-specific component responsible for managing the execution of a specific application within YARN.

It is launched by the ResourceManager on behalf of the application when resources are allocated to it.

The ApplicationMaster is responsible for coordinating tasks, monitoring their progress, handling failures, and requesting additional resources as needed.

It interacts with the NodeManagers to launch and manage containers for executing application tasks.

Each application running on YARN has its own ApplicationMaster instance, which runs on one of the nodes in the cluster.

19.Explain Mapreduce working?

A MapReduce system is usually composed of three steps (even though it's generalized as the combination of Map and Reduce operations/functions). The MapReduce operations are:

Map: The input data is first split into smaller blocks. The Hadoop framework then decides how many mappers to use, based on the size of the data to be processed and the memory block available on each mapper server. Each block is then assigned to a mapper for processing. Each ‘worker’ node applies the map function to the local data, and writes the output to temporary storage. The primary (master) node ensures that only a single copy of the redundant input data is processed.

Shuffle, combine and partition: worker nodes redistribute data based on the output keys (produced by the map function), such that all data belonging to one key is located on the same worker node. As an optional process the combiner (a reducer) can run individually on each mapper server to reduce the data on each mapper even further making reducing the data footprint and shuffling and sorting easier. Partition (not optional) is the process that decides how the data has to be presented to the reducer and also assigns it to a particular reducer.

Reduce: A reducer cannot start while a mapper is still in progress. Worker nodes process each group of <key,value> pairs output data, in parallel to produce <key,value> pairs as output. All the map output values that have the same key are assigned to a single reducer, which then aggregates the values for that key. Unlike the map function which is mandatory to filter and sort the initial data, the reduce function is optional.

20.How many mappers are created for 1 GB file?

The number of mappers created for a 1 GB file in a MapReduce job depends on several factors, including the input format, block size, and cluster configuration. In Hadoop, by default, each file is divided into blocks, and each block is processed by a separate mapper.

Assuming the default Hadoop block size of 128 MB:

1 GB file = 1024 MB

Number of blocks = 1024 MB / 128 MB = 8 blocks

Each block will be processed by a separate mapper. Therefore, for a 1 GB file with the default block size, there will be 8 mappers created in a MapReduce job.

It's important to note that this calculation assumes no data compression or input format considerations. If the file is compressed or if a different input format is used, the number of mappers may vary. Additionally, the number of available slots in the cluster's nodes may also affect the number of mappers that can run concurrently.

21.How many reducers are created for 1 GB file?

The number of reducers created for a 1 GB file in a MapReduce job depends on various factors, including the cluster configuration, job settings, and input data characteristics. In Hadoop, the number of reducers can be specified by the user or set by default based on the cluster's configuration.

By default, if the number of reducers is not explicitly specified, Hadoop will use a default number of reducers, which is usually determined by the total number of available reduce slots in the cluster.

Assuming a default configuration with a moderate number of reduce slots available, Hadoop might create a smaller number of reducers, such as 1 or 2, for processing a 1 GB file.

22.What is combiner?

A combiner is a functionality in MapReduce that acts as a mini-reducer on the output of the Mapper function before sending the data to the Reducers. It helps reduce the amount of data transferred between the Mapper and Reducer tasks by performing a local aggregation of intermediate key-value pairs on the mapper nodes.

23.What is partitioner?

A partitioner is a component in Hadoop's MapReduce framework responsible for determining which reducer will receive the output key-value pairs from the mappers. It performs this task based on the key of each pair. The primary purpose of a partitioner is to ensure that all key-value pairs with the same key end up at the same reducer, enabling efficient aggregation and processing of related data.