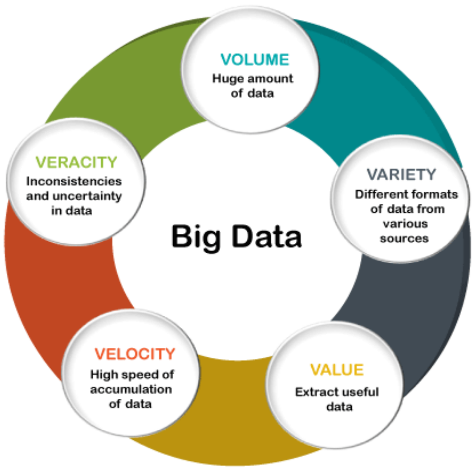
Q1. What is big data?

Ans: Big data refers to extremely large and diverse collections of structured, unstructured, and semi-structured data that continues to grow exponentially over time. These datasets are so huge and complex in volume, velocity, and variety, that traditional data management systems cannot store, process, and analyze them.

Big data describes large and diverse datasets that are huge in volume and also rapidly grow in size over time. Big data is used in machine learning, predictive modelling, and other advanced analytics to solve business problems and make informed decisions

Q2. Explain the characteristics of big data analytics. what are the five Vs of big data?

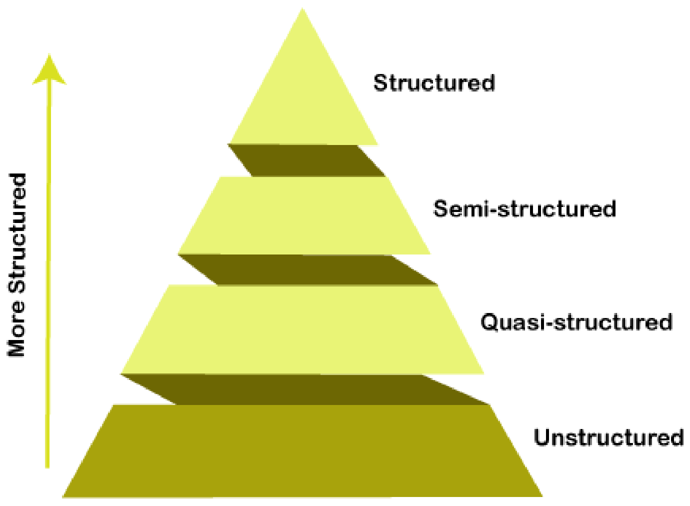

VOLUME
Huge amount of data
VARIETY
Different formats of data from various sources
VALUE
Extract useful data
VELOCITY
High speed of accumulation of data
VERACITY
Inconsistencies and uncertainty in data
Big Data

Ans:

Volume: Volumes of data generated from many sources daily, such as business processes, machines, social media platforms, networks, human interactions, etc.

Eg: Facebook can generate approximately a billion messages, 4.5 billion times that the "Like" button is recorded, and more than 350 million new posts are uploaded each day.

Variety: Big Data can be structured, unstructured, and semi-structured that are being collected from different sources. Data will only be collected from databases and sheets in the past, But these days the data will comes in array forms, that are PDFs, Emails, audios, SM posts, photos, videos etc.



The data is categorized as below:

Structured data: It is in a tabular form. Structured Data is stored in the relational database management system.

Semi-structured: In Semi-structured, the schema is not appropriately defined, e.g., JSON, XML, CSV, TSV, and email. OLTP (Online Transaction Processing) systems are built to work with semi-structured data.

Unstructured Data: All the unstructured files, log files, audio files, and image files are included in the unstructured data.

Quasi-structured Data: The data format contains textual data with inconsistent data formats that are formatted with effort and time with some tools.

Example: Web server logs, i.e., the log file is created and maintained by some server that contains a list of activities.

Veracity

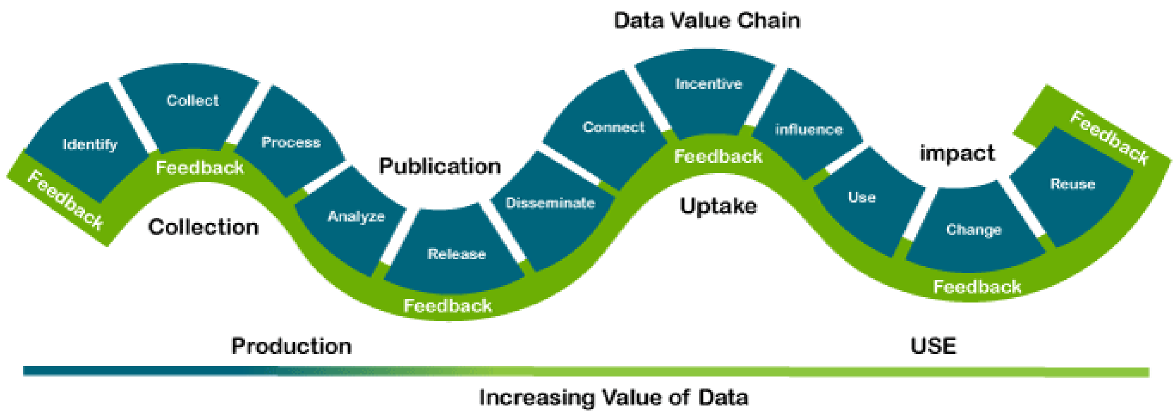
Veracity means how much the data is reliable. It has many ways to filter or translate the data. Veracity is the process of being able to handle and manage data efficiently.

For example: Facebook posts with hashtags.

Value

It is not the data that we process or store. It is valuable and reliable data that we store, process, and also analyze.

ADVERTISEMENT



Velocity

Big data velocity deals with the speed at the data flows from sources like application logs, business processes, networks, and social media sites, sensors, mobile devices, etc.

Q3. **What is Hadoop Eco System? What are its main components? (HDFS, YARN and Map reduce)**

Ans: ***The core components of Hadoop are HDFS, YARN, and Map Reduce.***

Data Ingestion

Data ingestion is the first layer of Big Data Architecture. The data is generated from various sources such as social media. Sensors, IoT devices, and SaaS platforms need to be collected and brought to a single warehouse or a database. There are three types of data ingestion techniques.

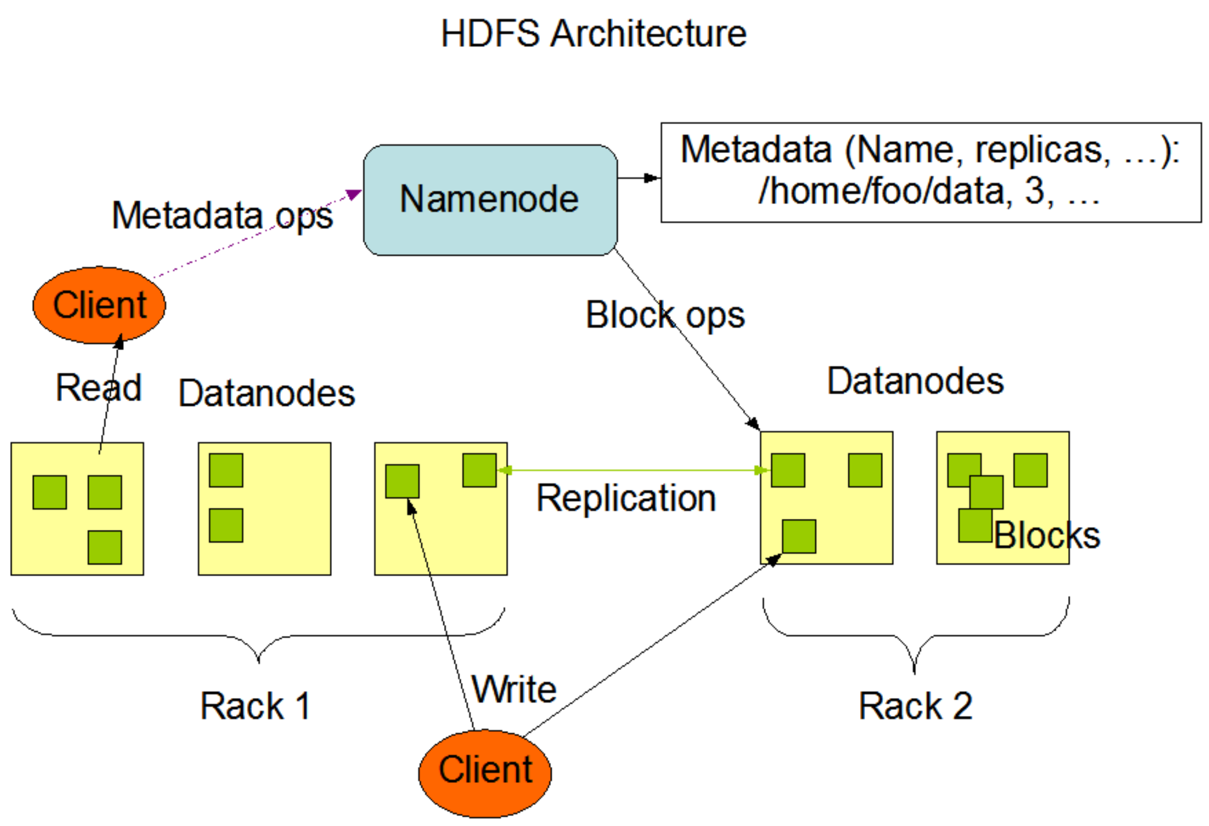
HDFS

HDFS stands for Hadoop Distributed File System, and it is designed to run on commodity servers. A typical HDFS architecture consists of a Name node and several data nodes. Nodes can be thought up as a single computer, and a collection of nodes constitute a cluster, and each cluster could boast 1000s of nodes.

When HDFS receives data, it converts the data file into small chunks of data, mostly 64 or 128 MB. The chunk size depends on the system configuration. While partitioning and replicating the data, HDFS follows a principle called rack awareness. A rack is a collection of 40-50 data nodes. Each copy of chunked data gets stored in different racks, thus making it highly fault-tolerant.

HDFS follows a master-slave architecture for data processing. The master node is also called the Name node, while the data nodes are secondary. The master server or the Name node manages the file system namespace and regulates access to files by clients. The data nodes contain the storage attached to the node and execute read and write operations from file system clients. Also, they are responsible for the deletion, replication and block creation upon request from the Name node.

The entire process can be summed up in the below picture



Source: https://hadoop.apache.org/

The master-slave architecture has one critical weakness: the cluster operation will halt if the Name node or Master node is compromised.

HBase

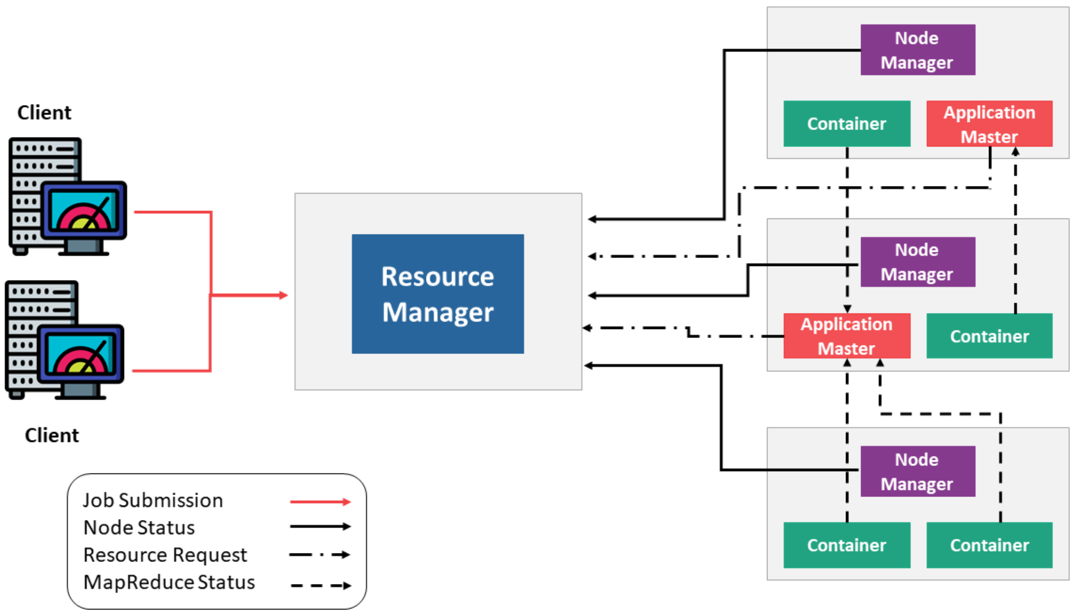
We discussed HDFS, and now we will move on to HBase, a column-oriented non-relational database management system that runs on top of HDFS. It operates similarly to HDFS; it has a master node to manage clusters, and slave nodes or region servers store the portion of the table and perform read and write operations. It has a unique concept of clubbing columns into a column family. These column families can be changed as and when needed making them flexible to changing application requirements. It is ideal for analysing real-time data and random read-write access to voluminous data.

YARN

YARN stands for Yet Another Resource Negotiator. In Hadoop 1.0, map-reduce was responsible for processing and job tracking tasks. But the utilisation of resources turned out to be highly inefficient. Then came YARN which took over the task of resource distribution and job scheduling from map-reduce. The YARN now sits in the middle of HDFS and map-reduce. There are four critical components in YARN.

- **Resource Manager:** The master node is chiefly responsible for resource allocation and directing Node managers to perform real processing. A scheduler is responsible for scheduling jobs, and resource distribution and an Application manager takes care of the job submissions and is accountable for running application masters in a cluster.

- **Node Manager:** A node manager runs a slave daemon. It is responsible for running assigned jobs on each node. It periodically sends heartbeats to signal the active status of the respective nodes.
- **Application Masters:** It negotiates resources from Resource Manager and coordinates with node managers to execute tasks.
- **Containers**: Containers are the hardware resources available in a cluster, such as CPUs, RAMs, disk space etc. It grants access to applications to use specific amounts of resources**.**



Oozie

Apache OOzie is an open-source Java web application for workflow scheduling in a distributed cluster. It combines multiple jobs into a single unit, and Oozie supports various jobs from Hive, Map Reduce, pig etc. There are three types of Oozie jobs.

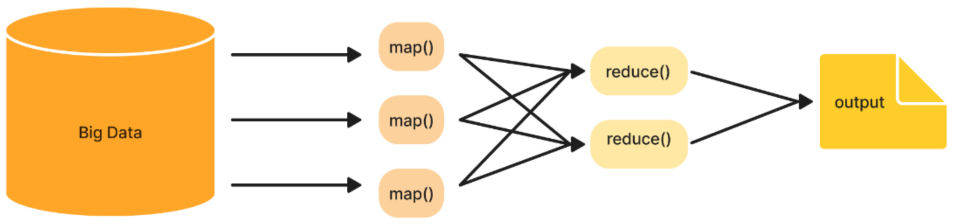
- **Oozie Workflow jobs:** These are Directed Acyclic Graphs (DAGs) which specify a sequence of actions to be executed**.**
- **Oozie Coordinator jobs:** These are recurrent Oozie Workflow jobs triggered by time and data availability.
- **Oozie Bundle:** It provides a way to package multiple coordinators and workflow jobs and manage the lifecycle of those jobs.

Data Processing and Analysis

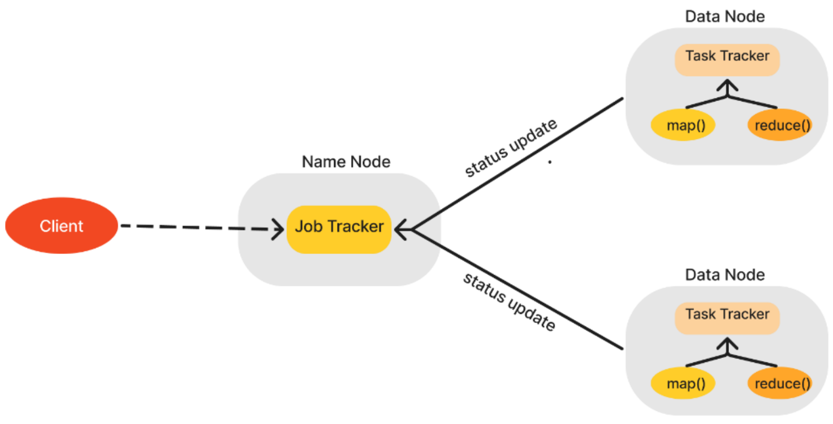
This could be thought of as the nervous system of Big Data architecture. Map Reduce, another core component of Hadoop, is primarily responsible for data processing. We will also discuss other software libraries that take part in data processing and analysis tasks.

Map Reduce

Map Reduce is responsible for processing a huge amount of data in a parallel distributed manner. It has two different jobs: Map and the other is Reduce. Just as the name Map always proceeds to Reduce. The data is processed and converted into key-value pairs or tuples in the Map stage. the output of the map job is fed to the reducer as inputs. Before being sent to the reducer, the intermediate data is sorted and organised, and the reducer then aggregates the key-value pair to output a smaller set of outputs. Final data is then stored in HDFS.



Just like HDFS, Map Reduce follows a master-slave design to accomplish tasks. Each Name node has a Job tracker, which divides and tracks the job submitted by the clients. Each job is then distributed among data nodes. These data nodes house task trackers, periodically sending a heartbeat indicating the node is alive. This way job tracker tracks the entire process. In case of a data node failure, the job tracker assigns the job to another node, thus making the system fault-tolerant.



Pig

Yahoo developed Apache Pig to analyse large amounts of data. This is what map-reduce does, too, but one fundamental problem with Map Reduce is it takes a lot of code to perform the intended jobs. This is the primary reason why Pig was developed. It has two significant components Pig Latin and Pig engine.

Pig Latin is a high-level language that is used to perform analysis tasks. 10 lines of Pig Latin code can achieve the same task as 200 lines of map-reduce code. The pig codes internally get converted to map-reduce jobs with the help of the pig engine. Thus making the entire process easier. The Pig Latin language is similar to SQL.

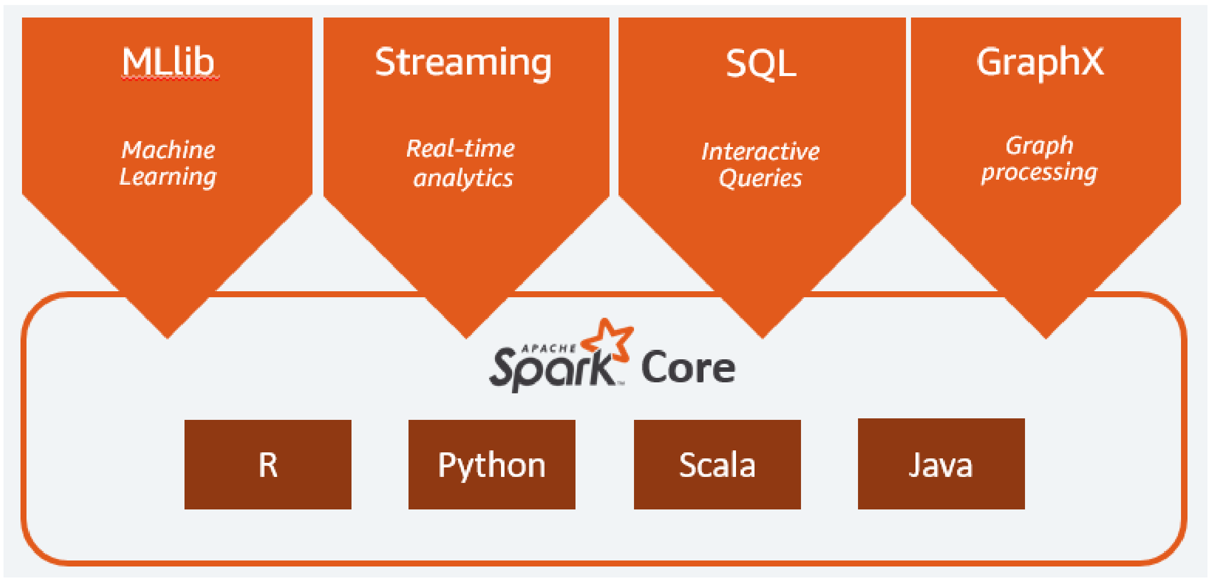
Spark

One of the critical concerns with map-reduce was that it takes a sequential multi-step process to run a job, and it has to read cluster data to do the operation and write it back to nodes to perform a job. Thus, map-reduce jobs have high latency, making them inefficient for real-time analytics.

To overcome the shortcoming, Spark was developed. The key features that set Apache Spark apart from map-reduce are its in-memory computation capability and reusability of data across parallel operations. This makes it almost 100 times faster than Hadoop map-reduce for large scale data processing.

The Spark framework includes The spark core, Spark SQL, MLlib, streaming and Graphx.

- **Spark Core:** This is responsible for memory management, scheduling, distributing, monitoring jobs, and fault recovery. And it was interacting with storage systems. It can be accessed by different programming languages such as Java, Scala, Python and R via APIs.
- **MLlib:** Library consisting of machine algorithms to do regression, classification, clustering etc.
- **Streaming:** It helps ingest real-time data from sources such as Kafka, Twitter, and Flume in mini-batches and perform real-time analytics on the same using codes written for batch analytics**.**
- **Spark SQL**: Distributed querying engine that provides highly optimised queries up to 100x faster than map-reduce. It supports various data sources out-of-the-box including Hive, Cassandra, HDFS etc.
- **Graphx:** It is a distributed graph processing unit that provides ETL, Graph computation and exploratory analysis at scale.



Spark is an ecosystem in itself. It has its cluster manager called standalone manager, Spark SQL for accessing data, streaming for batch and real-time data processing etc. Honestly, it deserves an article in itself.

Data Access

Once the data is ingested from different sources and stored in cluster nodes, the next step is to retrieve the right data for our needs. There are a bunch of software that helps us access the data efficiently as and when needed.

Hive

Hive is a data warehousing tool designed to work with voluminous data, and it works on top of HDFS and Map Reduce. The Hive query language is similar to SQL, making it user-friendly. The hive queries internally get converted into map-reduce or spark jobs which run on Hadoop's distributed node cluster.

Impala

Apache Impala is an open-source data warehouse tool for querying high volume data. Syntactically it is similar to HQL but provides highly optimised faster queries than Hive. Unlike Hive, it is not dependent on map-reduce; instead, it has its engine, which stores intermediate results in memory, thus providing faster query execution. It can easily be integrated with HDFS, Hbase and amazon s3. AS Impala is similar to SQL, and the learning curve is not very steep.

[Hue

Apache Hue is an open-source web interface for Hadoop components developed by Cloudera. It provides an easy interface to interact with Hive data stores, manage HDFS files and directories, and track map-reduce jobs and Oozie workflows. If you are not a fan of Command Line Interface, this is the right tool to interact with various Hadoop components.]

Zookeeper

Apache zookeeper is another essential member of the Hadoop family, responsible for cross node synchronisation and coordination. Hadoop applications may need cross-cluster services; deploying Zookeeper takes care of this issue. Applications create a znode within Zookeeper; applications can synchronise their tasks across the distributed cluster by updating their status in the znode. Zookeeper then can relegate information regarding a specific node's status change to other nodes.

Q5. What are the issues and challenges of big data?

Ans: **Some of the Big Data challenges are:**

1. ***Sharing and Accessing Data:***
    - The inaccessibility of data sets from external sources.
    - Sharing data can cause substantial challenges.
    - It includes the need for inter and intra- institutional legal documents.
    - Accessing data from public repositories leads to multiple difficulties.

2. ***Privacy and Security:***
    - It includes sensitive, conceptual, technical as well as legal significance.
    - Most of the organizations are unable to maintain regular checks due to large amounts of data generation. However, it should be necessary to perform security checks and observation in real time because it is most beneficial.
    - Some of the organization collects information of the people in order to add value to their business. This is done by making insights into their lives that they're unaware of.
3. ***Analytical Challenges:***
    - There are some huge analytical challenges in big data which arise some main challenges questions like how to deal with a problem if data volume gets too large?
    - Or how to find out the important data points?
    - Or how to use data to the best advantage?
    - These large amount of data on which these types of analysis are to be done can be structured (organized data), semi-structured (Semi-organized data) or unstructured (unorganized data). There are two techniques through which decision making can be done:
        - Either incorporate massive data volumes in the analysis.
        - Or determine upfront which Big-data is relevant.

4. ***Technical challenges:***

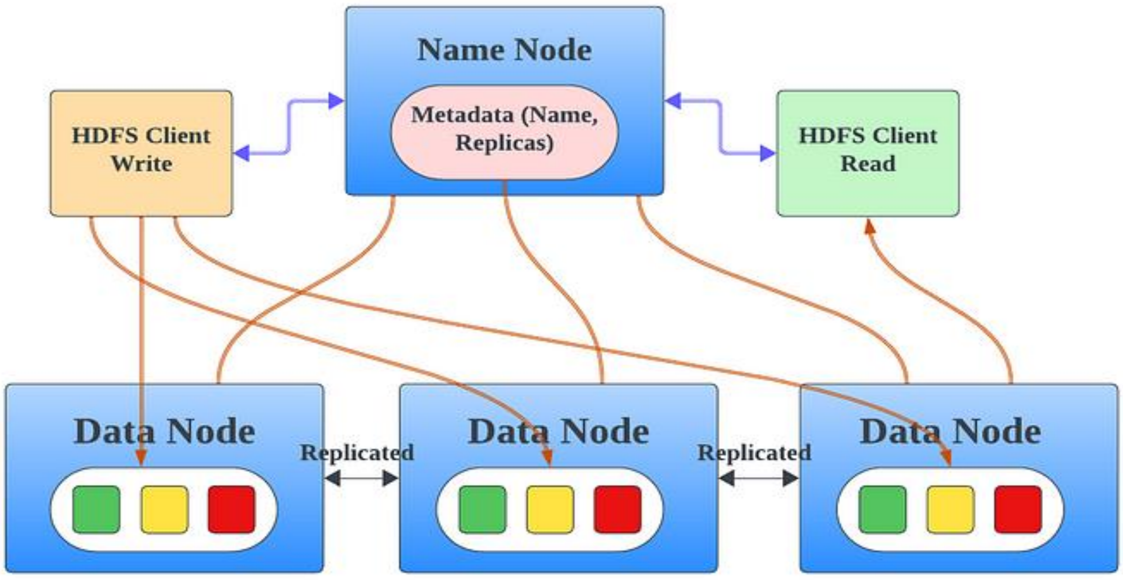
- **Quality of data:**
    - When there is a collection of a large amount of data and storage of this data, it comes at a cost. Big companies, business leaders and IT leaders always want large data storage.
    - For better results and conclusions, Big data rather than having irrelevant data, focuses on quality data storage.
    - This further arise a question that how it can be ensured that data is relevant, how much data would be enough for decision making and whether the stored data is accurate or not.
- **Fault tolerance:**
    - Fault tolerance computing is extremely hard, involving intricate algorithms.
    - The new technologies like cloud computing and big data always intended that whenever the failure occurs the damage done should be within the acceptable threshold that is the whole task should not begin from the scratch.
- **Scalability:**
    - Big data projects can grow and evolve rapidly. The scalability issue of Big Data has lead towards cloud computing.
    - It leads to various challenges like how to run and execute various jobs so that goal of each workload can be achieved cost-effectively.
    - It also requires dealing with the system failures in an efficient manner. This leads to a big question again that what kinds of storage devices are to be used.

Assignment II

Q1. Explain the concept of HDFS. Architectural design of HDFS.

HDFS Concepts? What is HDFS and what are its main components? (Name node and Data node)

1. **Ans: Blocks:** A Block is the minimum amount of data that it can read or write. HDFS blocks are 128 MB by default and this is configurable. Files n HDFS are broken into block-sized chunks, which are stored as independent units.
2. **Name Node:** The name node acts as master. Name Node is controller and manager of HDFS as it knows the status and the metadata of all the files in HDFS; the metadata information being file permission, names and location of each block. The metadata are small, so it is stored in the memory of name node, allowing faster access to data.
3. **Data Node:** They store and retrieve blocks by client or name node. They report back to name node periodically, with list of blocks that they are storing. The data node being a commodity hardware also does the work of block creation, deletion and replication as stated by the name node.



Q2. Enlist the commands used in HDFS ?

Command	**Description**
-rm	Removes file or directory
-ls	Lists files with permissions and other details
-mkdir	Creates a directory named path in HDFS
-cat	Shows contents of the file
-rmdir	Deletes a directory
-put	Uploads a file or folder from a local disk to HDFS
-rmr	Deletes the file identified by path or folder and subfolders
-get	Moves file or folder from HDFS to local file
-count	Counts number of files, number of directory, and file size
-df	Shows free space
-getmerge	Merges multiple files in HDFS
-chmod	Changes file permissions
-copyToLocal	Copies files to the local system
-Stat	Prints statistics about the file or directory
-head	Displays the first kilobyte of a file
-usage	Returns the help for an individual command
-chown	Allocates a new owner and group of a file

Q3. Explain data ingest with Flume and Sqoop?

- **Ans: . Apache Flume:** Flume is designed for efficiently collecting, aggregating, and transporting large amounts of log data or event data from various sources to a central repository like HDFS (Hadoop Distributed File System). **Flume** is great for streaming data into Hadoop.

Working:

- **Sources**: Flume sources gather data from different input channels. For example, you might have sources that read data from log files, syslogs, or even custom sources.
- **Channels**: Channels act as buffers that temporarily hold the data as it is being transferred from the source to the Destination (sink).
- **Destination**: Sinks write the data to the final destination, such as HDFS, HBase, or another storage system.

Example: Imagine you have a web server generating log files. You can use Flume to continuously monitor these log files, collect the data, and load it into HDFS for further processing.

- **2. Apache Sqoop:** Sqoop is used for transferring data between relational databases (like MySQL, PostgreSQL, Oracle) and Hadoop. It's particularly useful for importing large volumes of data from traditional databases into HDFS or exporting data from HDFS back to a relational database. **Sqoop** is ideal for batch importing/exporting data between databases and Hadoop.

Working:

- **Import**: Sqoop connects to a relational database, retrieves data, and imports it into HDFS. It supports parallel imports, which can speed up the process by splitting the work across multiple tasks.
- **Export**: It can also take data from HDFS and export it back to a relational database, useful for integrating Hadoop-based analysis results with operational databases.
- Each tool addresses different aspects of data ingestion and management within a Hadoop ecosystem, making it easier to handle and process large volumes of data.

Example: If we have customer data stored in a MySQL database and we want to analyze it using Hadoop's ecosystem tools (like Hive or Spark), we would use Sqoop to import this data into HDFS.

Q4. What are Hadoop archives?

Ans: Hadoop Archives (HAR) HAR is a way to organize and compress data within HDFS to reduce the number of files and improve performance when accessing large datasets. **HAR** helps manage and optimize the storage of a large number of small files in HDFS

- **Archive**: Hadoop Archives bundle a large number of files into a single archive file. This reduces the number of files in HDFS and can help improve performance by reducing the metadata overhead.
- **Access**: Data in HAR files can be accessed just like regular HDFS files, but the access patterns are optimized for performance.

Example: If we have millions of small files (e.g., log files or data chunks), managing these files can become inefficient due to the overhead of storing metadata. By archiving these files into HAR files, we can improve performance and manageability.

In Hadoop, **compression** and **serialization** are crucial techniques to optimize data storage and transmission. They help in reducing the size of data and enabling efficient data processing across the distributed Hadoop ecosystem, particularly HDFS (Hadoop Distributed File System) and MapReduce.

Q5. Where Compression is Used?

Why Serialization is Important in Hadoop?

Ans: Compression Explain Compression and serialization in Hadoop.in Hadoop: Compression reduces the size of data files, which is essential in a distributed environment like Hadoop, where large datasets are common. By using compression, less space is needed for storage, and data transfer becomes faster due to reduced input/output (I/O) overhead.

Use of Compression:

- **HDFS Storage:** Data stored on HDFS can be compressed to save disk space and improve I/O performance.
- **MapReduce Jobs:** Input/output data can be compressed to reduce the time spent reading from or writing to HDFS. Compression also helps reduce network traffic between mappers and reducers.

Advantages of Compression:

- **Less Disk Space Usage:** Reduces storage costs.
- **Faster Data Transmission:** Decreases the amount of data transferred across the network.
- **Improved Performance:** Speeds up processing tasks by reducing I/O bottlenecks.

---

2. Serialization in Hadoop:

Serialization is the process of converting an object into a byte stream to store it or transmit it to another system, where it can later be deserialized into its original form. Hadoop relies on serialization to efficiently process and move data across its distributed environment.

- **Data Exchange:** In a distributed system, data must be exchanged between nodes (across mappers, reducers, and data nodes). Serialization allows this data to be efficiently encoded and transferred.
- **Storage:** Data that is stored in HDFS or processed by MapReduce jobs needs to be serialized to ensure that it can be efficiently written to and read from disk.

Serialization Frameworks in Hadoop:

Writable: Hadoop's native serialization format, optimized for speed and compatibility with the Hadoop ecosystem.

Advantages: Lightweight, fast, and well-suited for Hadoop's distributed nature.

Disadvantage: Limited to Hadoop and Java environments; less portable.

❖ **Difference between Compression and serialization**

Feature	**Compression**	**Serialization**
Purpose	Reduce data size for storage and I/O	Encode data into a format for storage and transmission
Scope	Data files (HDFS, MapReduce)	In-memory data and data exchange
Common Formats	Gzip, Bzip2, Snappy, LZO	Writable, Avro, Protobuf, Thrift
Use Cases	Reducing disk space, speeding up jobs	Transmitting data between nodes, RPCs

Assignment III

Q1. Explain the Anatomy of a MapReduce Job.

Answer: Mapper

Record Reader:

The record reader translates an input split generated by input format into records. The purpose of record reader is to parse the data into record but doesn't parse the record itself. It passes the data to the mapper in form of key/value pair. Usually, the key in this context is positional information and the value is a chunk of data that composes a record.

Map:

Map function is the heart of mapper task, which is executed on each key/value pair from the record reader to produce zero or more key/value pair, called intermediate pairs. The decision of what is key/value pair depends on what the MapReduce job is accomplishing. The data is grouped on key and the value is the information pertinent to the analysis in the reducer.

Combiner:

It is an optional component but highly useful and provides extreme performance gain of MapReduce job without any downside. Combiner is not applicable to all the MapReduce algorithms but where ever it can be applied it is always recommended to use. It takes the intermediate keys from the mapper and applies a user-provided method to aggregate values in a small scope of that one mapper.

Partitioner:

The Hadoop partitioner takes the intermediate key/value pairs from mapper and split them into shards, one shard per reducer. This randomly distributes the key space evenly over the reducer, but still ensures that keys with the same value in different mappers end up at the same reducer. The partitioned data is written to the local filesystem for each map task and waits to be pulled by its respective reducer.

Reducer

Shuffle and Sort:

The reduce task start with the shuffle and sort step. This step takes the output files written by all of the Hadoop partitioners and downloads them to the local machine in which the reducer is running. These individual data pipes are then sorted by keys into one larger data list. The purpose of this sort is to group equivalent keys together so that their values can be iterated over easily in the reduce task.

Reduce:

The reducer takes the grouped data as input and runs a reduce function once per key grouping. The function is passed the key and an iterator over all the values associated with that key. A wide range of processing can happen in this function, the data can be aggregated, filtered, and combined in a number of ways. Once it is done, it sends zero or more key/value pair to the final step, the output format.

Output Format:

The output format translates the final key/value pair from the reduce function and writes it out to a file by a record writer. By default, it will separate the key and value with a tab and separate record with a new line character.

Q2. What is MapReduce type and its format?

Ans: MapReduce is a programming model in Big Data that uses a combination of map and reduce functions to process large volumes of data.

MapReduce functions: The map function reads data from a database and transfers it to a more accessible format, while the reduce function combines the final classification results.

MapReduce input and output types: The map input key and value types (K1 and V1) are usually different from the map output types (K2 and V2).

MapReduce input formats: Some examples of MapReduce input formats include FileInputFormat, TextInputFormat, KeyValueTextInputFormat, SequenceFileInputFormat, SequenceFileAsTextInputFormat, and NlineInputFormat.

MapReduce data types:

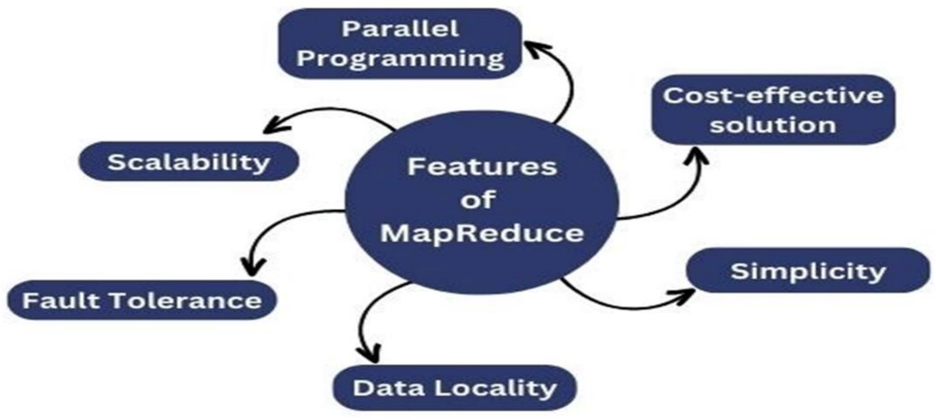
Some basic data types used in MapReduce include IntWritable and Text, which support serialization and comparability.

MapReduce file formats: Some common file formats used in MapReduce include XML, JSON, SequenceFiles, Avro, and Parquet.

MapReduce applications: MapReduce applications run in parallel to process large volumes of data stored on clusters.

Q3. What are the features of Map Reduce?

Ans:



Scalability

MapReduce can scale to process vast amounts of data by distributing tasks across a large number of nodes in a cluster. This allows it to handle massive datasets, making it suitable for Big Data applications.

Fault Tolerance

MapReduce incorporates built-in fault tolerance to ensure the reliable processing of data. It automatically detects and handles node failures, rerunning tasks on available nodes as needed.

Data Locality

MapReduce takes advantage of data locality by processing data on the same node where it is stored, minimizing data movement across the network and improving overall performance.

Simplicity

The MapReduce programming model abstracts away many complexities associated with distributed computing, allowing developers to focus on their data processing logic rather than low-level details.

Cost-Effective Solution

Hadoop's scalable architecture and MapReduce programming framework make storing and processing extensive data sets very economical.

Parallel Programming

Tasks are divided into programming models to allow for the simultaneous execution of independent operations. As a result, programs run faster due to parallel processing, making it easier for a process to handle each job. Parallel processing makes it easy; it distributes the tasks and perform the multiple processors. Therefore, all software runs faster.

Q4. What are the failures in MapReduce?

Ans: **Failures in MapReduce**

1. Task Failure:

The **task failure** is similar to an employee mak

ing a mistake while doing a task. Consider you are working on a large project that has been broken down into smaller jobs and assigned to different employees in your team. If one of the team members fails to do their task correctly, the entire project may be compromised. Similarly, in Hadoop, if a job fails due to a mistake or issue, it could affect overall data processing, causing delays or faults in the final result.

Limited memory: A task can fail if it runs out of memory while processing data.
Failures of disk: If the disk that stores data or intermediate results fails, tasks that depend on that data may fail.
Issues with software or hardware: Bugs, mistakes, or faults in software or hardware components can cause task failures.

How to Overcome Task Failure

Increase memory allocation: Assign extra memory to jobs to ensure they have the resources to process the data.
Implement fault tolerance mechanisms: Using data replication and checkpointing techniques to defend against disc failures and retrieve lost data.
Regularly update software and hardware: Keep the Hadoop framework and supporting hardware up to date to fix bugs, errors, and performance issues that can lead to task failures.

2. **TaskTracker Failure:**

A **TaskTracker** in Hadoop is similar to an employee responsible for executing certain tasks in a large project. If a TaskTracker fails, it signifies a problem occurred while an employee worked on their assignment. This can interrupt the entire project, much as when a team member makes a mistake or encounters difficulties with their task, producing delays or problems with the overall project's completion. To avoid TaskTracker failures, ensure the TaskTracker's hardware and software are in excellent working order and have the resources they need to do their jobs successfully.
Hardware issues: Just as your computer's parts can break or stop working properly, the TaskTracker's hardware (such as the processor, memory, or disc) might fail or stop operating properly. This may prohibit it from carrying out its duties.
Software problems or errors: The software operating on the TaskTracker may contain bugs or errors that cause it to cease working properly. It's similar to when an app on your phone fails and stops working properly.
Overload or resource exhaustion: It may struggle to keep up if the TaskTracker becomes overburdened with too many tasks or runs out of resources such as memory or processing power. It's comparable to being overburdened with too many duties or running out of storage space on your gadget.
Update software and hardware on a regular basis: Keep the Hadoop framework and associated hardware up to date to correct bugs, errors, and performance issues that might lead to task failures.

Upgrade or replace hardware: If TaskTracker's hardware is outdated or insufficiently powerful, try upgrading or replacing it with more powerful components. It's equivalent to purchasing a new, upgraded computer to handle jobs more efficiently.

Restart or reinstall the program: If the TaskTracker software is causing problems, a simple restart or reinstall may be all that is required. It's the same as restarting or reinstalling an app to make it work correctly again.

Alright!! Moving forward, let us learn about JobTracker Failure.

3. JobTracker Failure:

A **JobTracker** in Hadoop is similar to a supervisor or manager that oversees the entire project and assigns tasks to TaskTrackers (employees). If a JobTracker fails, it signifies the supervisor is experiencing a problem or has stopped working properly. This can interrupt the overall project's coordination and development, much as when a supervisor is unable to assign assignments or oversee their completion. To avoid JobTracker failures, it is critical to maintain the JobTracker's hardware and software, ensure adequate resources, and fix any issues or malfunctions as soon as possible to keep the project going smoothly.
Database connectivity: The JobTracker stores job metadata and state information in a backend database (usually Apache Derby or MySQL). JobTracker failures can occur if there are database connectivity issues, such as network problems or database server failures.
Security problems: JobTracker failures can be caused by security issues such as authentication or authorization failures, incorrectly configured security settings or key distribution and management issues.

How to Overcome JobTracker Failure

Avoiding Database Connectivity: To avoid database connectivity failures in the JobTracker, ensure optimized database configuration, robust network connections, and high availability techniques are implemented. Retrying connections, monitoring, and backups are all useful.
To overcome security-related problems: implement strong authentication and authorization, enable SSL/TLS for secure communication, keep software updated with security patches, follow key management best practices, conduct security audits, and seek x expert guidance for vulnerability mitigation and compliance with security standards.

Q5. Explain different types of MapReduce formats.

Ans: **Types of InputFormat in MapReduce**

1. FileInputFormat:

It serves as the foundation for all file-based InputFormats. FileInputFormat also provides the input directory, which contains the location of the data files. When we start a MapReduce task, FileInputFormat returns a path with files to read. This InpuFormat will read all files. Then it divides these files into one or more InputSplits.

2. TextInputFormat:

It is the standard InputFormat. Each line of each input file is treated as a separate record by this InputFormat. It does not parse anything. TextInputFormat is suitable for raw data or line-based records, such as log files. Hence:

- Key: It is the byte offset of the first line within the file (not the entire file split). As a result, when paired with the file name, it will be unique.
- Value: It is the line's substance. It does not include line terminators.

3. KeyValueTextInputFormat:

It is comparable to TextInputFormat. Each line of input is also treated as a separate record by this InputFormat. While TextInputFormat treats the entire line as the value, KeyValueTextInputFormat divides the line into key and value by a tab character ('/t'). Hence:

- Key: Everything up to and including the tab character.

- Value: It is the remaining part of the line after the tab character.

4. SequenceFileInputFormat:

It's an input format for reading sequence files. Binary files are sequence files. These files also store binary key-value pair sequences. These are block-compressed and support direct serialization and deserialization of a variety of data types. Hence Key & Value are both user-defined.

5. SequenceFileAsTextInputFormat:

It is a subtype of SequenceFileInputFormat. The sequence file key values are converted to Text objects using this format. As a result, it converts the keys and values by running 'toString()' on them. As a result, SequenceFileAsTextInputFormat converts sequence files into text-based input for streaming.

6. NlineInputFormat

It is a variant of TextInputFormat in which the keys are the line's byte offset. And values are the line's contents. As a result, each mapper receives a configurable number of lines of TextInputFormat and KeyValueTextInputFormat input. The number is determined by the magnitude of the split. It is also dependent on the length of the lines. So, if we want our mapper to accept a specific number of lines of input, we use NLineInputFormat.

N- It is the number of lines of input received by each mapper.

Each mapper receives exactly one line of input by default (N=1).

Assuming N=2, each split has two lines. As a result, the first two Key-Value pairs are distributed to one mapper. The second two key-value pairs are given to another mapper.

7. DBInputFormat

Using JDBC, this InputFormat reads data from a relational Database. It also loads small datasets, which might be used to connect with huge datasets from HDFS using multiple inputs. Hence:

- Key: LongWritables

- Value: DBWritables.

Output Format in MapReduce

The output format classes work in the opposite direction as their corresponding input format classes. The TextOutputFormat, for example, is the default output format that outputs records as plain text files, although key values can be of any type and are converted to strings by using the toString () method. The tab character separates the key-value character, but this can be changed by modifying the separator attribute of the text output format.

SequenceFileOutputFormat is used to write a sequence of binary output to a file for binary output. Binary outputs are especially valuable if they are used as input to another MapReduce process.

DBOutputFormat handles the output formats for relational databases and HBase. It saves the compressed output to a SQL table.

Assignment IV

Q1. Explain the new feature of name node with high availability in Hadoop 2.0.

Ans: Hadoop 2.0's Name Node High Availability feature addresses the single point of failure problem in Hadoop clusters by introducing a second Name Node:

1. **Active Name Node**: Handles all client operations in the cluster
2. **Passive Standby Name Node**: A standby Name Node that has similar data to the active Name Node
3. **Automatic failover**: If the active Name Node fails, the Hadoop administrator can manually intervene or automatically switch to the passive Name Node
4. **Hot standby**: The passive Name Node maintains enough state to provide a fast failover
5. **Replication factor**: Data is replicated across multiple nodes, and the replication factor can be configured by the administrator

Other features of Hadoop 2.0 include:

1. YARN, which can process terabytes and petabytes of data
2. Splitting the Work Tracker's roles into the Task Master and the Global Resource Manager
3. HDFS Federation, which increases the horizontal scalability of the Name Node
4. HDFS Snapshot

5. Support for Windows
6. NFS access is also provided.

Q2. What is MRv2.

Ans: MRv2, also known as MapReduce NextGen or YARN, is a new architecture in Hadoop that separates the two main functions of the Job Tracker into separate daemons:

1. **Resource management**

   The Resource Manager manages the global assignment of compute resources to applications.

2. **Job life-cycl e management**

   The Application Master manages the application's scheduling and coordination.

   The YARN architecture also includes Node Managers, which manage the user processes on a machine.

3. **Data storage**

   MapReduce uses the Hadoop Distributed File System (HDFS) to store data.

4. **Parallel containers**

   YARN uses multiple parameters to determine the number of parallel containers.

5. **Performance tuning**

   We can tune parameters at the Hadoop cluster level to improve performance.

6. **Containers**

   A container is the physical instance of a process executed by YARN on a worker node within the cluster.

Q3. Write the difference between MRv1 and MRv2 in YARN.

Ans: MapReduce version 1 and MapReduce version 2 are different versions of the MapReduce computation framework that run on YARN, a component of the Hadoop data-processing ecosystem:

- **MRv1**: The original version of MapReduce, which runs as an application on top of YARN.
- **MRv2**: The rewritten version of MRv1 that runs on YARN.

Differences between MRv1 and MRv2:

1. **Resource management**

YARN separates resource management and job management, which were both handled by the Job Tracker in Hadoop 1.x.

2. **Resource allocation**

   YARN allows resources to be allocated and reallocated for different applications sharing a cluster.

3. **Backwards compatibility**

   YARN is backwards-compatible with MapReduce, so all jobs that run against MapReduce also run in a YARN cluster.

4. **Application submission**

   The syntax to submit applications is similar to the MRv1 framework, but the yarn command is still preferred.

5. **Configuration parameters**

   MRv2 has different configuration parameters than MRv1.

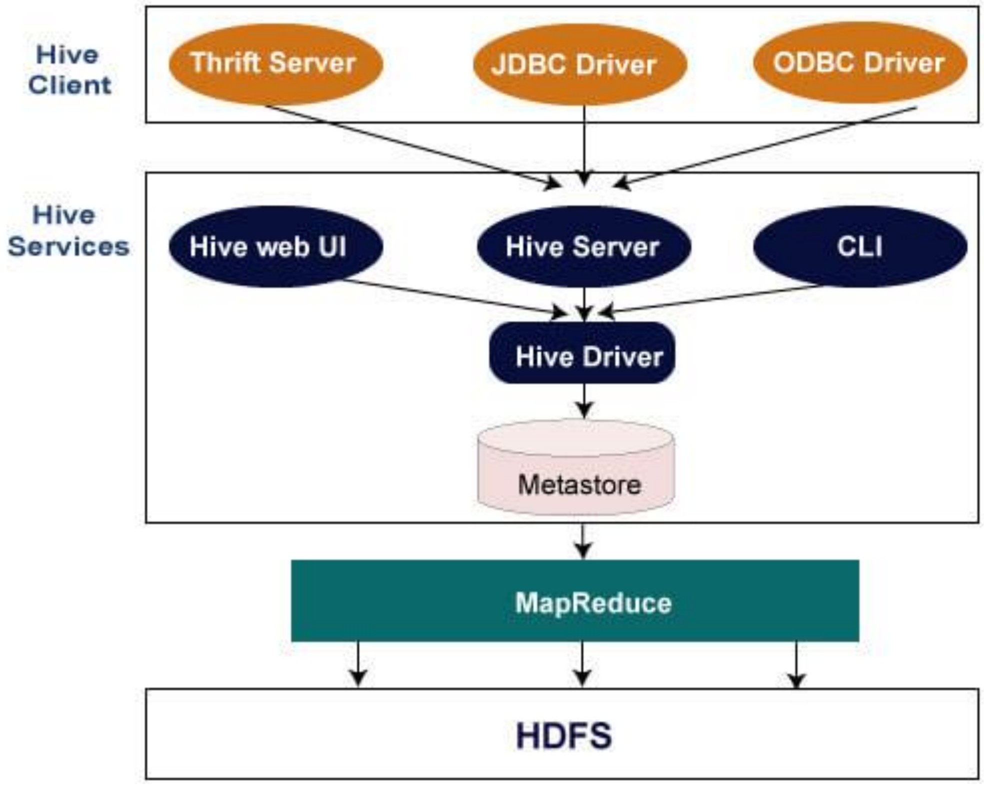
   We can monitor MRv1 applications running on a YARN cluster using the Resource Manager web interface.

Assignment V

Q1. Explain the architecture of Hive.

Ans: Hive Architecture

The following architecture explains the flow of submission of query into Hive.



Hive Client

Hive allows writing applications in various languages, including Java, Python, and C++. It supports different types of clients such as:-

- Thrift Server - It is a cross-language service provider platform that serves the request from all those programming languages that supports Thrift.
- JDBC Driver - It is used to establish a connection between hive and Java applications. The JDBC Driver is present in the class org.apache.hadoop.hive.jdbc.HiveDriver.
- ODBC Driver - It allows the applications that support the ODBC protocol to connect to Hive.

Hive Services

The following are the services provided by Hive:-

- Hive CLI - The Hive CLI (Command Line Interface) is a shell where we can execute Hive queries and commands.

- Hive Web User Interface - The Hive Web UI is just an alternative of Hive CLI. It provides a web-based GUI for executing Hive queries and commands.
- Hive MetaStore - It is a central repository that stores all the structure information of various tables and partitions in the warehouse. It also includes metadata of column and its type information, the serializers and deserializers which is used to read and write data and the corresponding HDFS files where the data is stored.
- Hive Server - It is referred to as Apache Thrift Server. It accepts the request from different clients and provides it to Hive Driver.
- Hive Driver - It receives queries from different sources like web UI, CLI, Thrift, and JDBC/ODBC driver. It transfers the queries to the compiler.
- Hive Compiler - The purpose of the compiler is to parse the query and perform semantic analysis on the different query blocks and expressions. It converts HiveQL statements into MapReduce jobs.
- Hive Execution Engine - Optimizer generates the logical plan in the form of DAG of map-reduce tasks and HDFS tasks. In the end, the execution engine executes the incoming tasks in the order of their dependencies.

Q2. Write the necessary steps for the installation of Hive.

Ans: To install Hive, you can follow these steps:

Hardware requirements

- At least 8 GB of RAM
- A quad-core CPU with at least 1.80 GHz
- JRE 1.8
- Java Development Kit 1.8
- A software for unzipping, like 7Zip or Win Rar

1. **Install Java**: Hive is a Java-based tool, so you need to install Java on your server. You can check the Hive documentation for the most compatible Java version.
2. **Download and untar Hive**: You can download Hive from GitHub at https://github.com/apache/hive.
3. **Configure Hive environment variables**: You can configure Hive environment variables in .bashrc.
4. **Edit core-site.xml file**: You can edit the core-site.xml file.
5. **Create Hive directories in HDFS**: You can create the /tmp directory and the /user/hive/warehouse directory.
6. **Configure hive-site.xml file**: You can configure the hive-site.xml file, but this is optional.
7. **Initiate Derby database**: You can initiate the Derby database.

8. **Launch Hive Client Shell**: You can launch the Hive Client Shell.

Q3. Write the steps for installing ZooKeeper.

Ans: **Install and Set Up ZooKeeper**

1. Download the latest ZooKeeper release from the Apache ZooKeeper website.
2. Extract the downloaded file and configure it by editing the zoo.cfg file in the conf directory. Define parameters like dataDir, clientPort, and tickTime.
3. Start ZooKeeper:

   bash

   Copy code

   ```
   bin/zkServer.sh start
   ```
4. Use zkCli.sh to interact with the ZooKeeper server and test your setup.
5. Set Up ZooKeeper Client Library in our Application
6. Add the ZooKeeper client library to our application:
7. Java: Add the ZooKeeper dependency to our pom.xml if using Maven:

   xml

   Copy code

   ```
   <dependency>
      <groupId>org.apache.zookeeper</groupId>
      <artifactId>zookeeper</artifactId>
      <version>3.7.0</version>
   </dependency>
   ```
8. Connect to ZooKeeper in our Application

   - Java Example:

     java

     Copy code

     ```
     import org.apache.zookeeper.ZooKeeper;

     ZooKeeper zooKeeper = new ZooKeeper("localhost:2181", 3000, event ->
     {
        System.out.println("Event received: " + event);
     });
     ```

9. Create, Read, Update, and Delete znodes
10. Use the ZooKeeper API to create, read, update, and delete znodes for storing shared configuration and synchronization information.

    Java Example:

    java

    Copy code

    // Create a new znode

    String path = "/myapp";

    byte[] data = "mydata".getBytes();

    zooKeeper.create(path, data, ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);

    // Read data from znode

    byte [] retrievedData = zooKeeper.getData(path, false, null);

11. Test and monitor our application

- Use the Zookeeper command-line interface (zkCli.sh) to check the status of our znodes and our application's interactions.
- Zookeeper provides logging and metrics that can be used to monitor performance and identify potential issues.

12. . Handle Session Timeouts and Failover

- Zookeeper clients may experience session timeouts, leading to issues if the application isn't designed for recovery. Make sure your application can handle KeeperException.SessionExpiredException.
- Use a retry mechanism or reconnect logic in your application to handle temporary network or Zookeeper node failures.

13. Deploy Zookeeper in Production

- For production, configure a Zookeeper ensemble (cluster) for high availability. Set up at least three nodes to maintain quorum.
- Regularly back up data stored in Zookeeper and monitor logs and metrics for system health.

Q4. Explain script joins in subquery with HBase query.

Ans: In HBase, we often deal with NoSQL data models that lack the join capabilities typical in relational databases. To perform "joins" in HBase, we generally rely on workarounds, such as using client-side code or leveraging additional tools. One common method is to use subqueries or scripting with joins in frameworks like Apache Hive or Apache Spark to make HBase queries that resemble SQL-style joins. Here's a breakdown:

1. Understanding HBase's Data Model

HBase stores data in a wide-column store format, organized by tables, rows, column families, and columns. Unlike traditional relational databases, it doesn't have built-in joins, so you need alternative approaches to combine data from multiple tables.

2. Using Hive with HBase Tables

Apache Hive provides an SQL-like interface on top of HBase tables. By defining HBase tables in Hive, you can use SQL queries, including subqueries and joins, on the HBase data.

For example, let's say we have two HBase tables:

- **users**: Stores user information with columns like user_id, name, and location.
- **orders**: Stores order information with columns like order_id, user_id, and amount.

We want to perform a join on user_id between the users and orders tables to find all orders made by each user.

3. Example Query Using Hive on HBase

Define HBase tables in Hive using CREATE EXTERNAL TABLE. You'll specify the HBase configuration (such as table name and column mappings).

sql

Copy code

```
CREATE EXTERNAL TABLE users (
  user_id STRING,
  name STRING,
  location STRING
) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,info:name,info:location")
TBLPROPERTIES("hbase.table.name" = "users");
CREATE EXTERNAL TABLE orders (
  order_id STRING,
  user_id STRING,
  amount DOUBLE
) STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,info:user_id,info:amount")
```

```
TBLPROPERTIES("hbase.table.name" = "orders");
```

Once these HBase tables are defined in Hive, you can use SQL queries, including joins:

sql

Copy code

```sql
SELECT u.user_id, u.name, o.order_id, o.amount
FROM users u
JOIN orders o
ON u.user_id = o.user_id
WHERE o.amount > 100;
```

4. Script-Based Joins with Subqueries

If you need more complex joins or subqueries not supported by HBase or Hive, you could use Apache Spark or a custom script to handle these cases. Spark provides a robust framework for querying and transforming large datasets, including HBase tables, with powerful join capabilities.

In Spark (using PySpark as an example):

python

Copy code

```python
from pyspark.sql import SparkSession
# Initialize Spark session with HBase configurations
spark = SparkSession.builder \
   .appName("HBase Joins") \
   .config("spark.hbase.host", "your-hbase-server") \
   .getOrCreate()
# Load HBase tables into DataFrames
users_df = spark.read.format("org.apache.hadoop.hbase.spark") \
   .option("hbase.table", "users") \
   .option("hbase.columns.mapping", "user_id STRING :key, name STRING info:name, location STRING info:location") \
   .load()
orders_df = spark.read.format("org.apache.hadoop.hbase.spark") \
   .option("hbase.table", "orders") \
    .option("hbase.columns.mapping", "order_id STRING :key, user_id STRING info:user_id, amount DOUBLE info:amount") \
    .load()

# Perform a join
joined_df = users_df.join(orders_df, users_df.user_id == orders_df.user_id, "inner") \
    .select(users_df.user_id, users_df.name, orders_df.order_id, orders_df.amount)

# Display results
joined_df.show()
```

Summary

- **Hive**: Use Hive's SQL interface to define HBase tables, then query them with SQL join syntax.
- **Spark**: Use Spark to load HBase tables as DataFrames, then use Spark's DataFrame joins for more flexibility and power.

Both methods offer ways to mimic SQL joins with HBase tables, making it easier to analyze and query data across multiple tables.

Q4. What is the difference between GFS and HDFS?