18CSD42

Hall Ticket Number:

IV/IV B.Tech (Regular/Supplementary) DEGREE EXAMINATION

November, 2022 — **Computer Science and Engineering**

Seventh Semester — **Big Data Analytics**

Time: Three Hours — **Maximum:** 50 Marks

Answer Question No.1 compulsorily. (10X1 = 10 Marks)

Answer ONE question from each unit. (4X10=40 Marks)

1.	a)	What is Big Data?	CO1	L2	
	b)	Write the characteristics of big data.	CO1	L1	
	c)	Name hadoop configuration files.	CO1	L3	
	d)	What is the role of Jobtracker and Tasktracker in Map Reduce?	CO2	L1	
	e)	List the features of Map Reduce.	CO2	L2	
	f)	List the failures in classical Map Reduce.	CO2	L4	
	g)	Define Hive.	CO3	L1	
	h)	What is Pig?	CO4	L2	
	i)	What is Sqoop?	CO4	L2	
	j)	How many ways to run a Pig scritps?	CO4	L3	
		Unit -I			
2.	a)	Identify the data storage problems and data analysis tasks problems prior to hadoop's file system.	CO1	L2	2M
	b)	Explain the characteristics and applications of Big Data.	CO1	L1	8M
		(OR)			
3.	a)	Explain in detail about Hadoop along with its architecture.	CO1	L4	5M
	b)	Explain in details about Hadoop Ecosystem.	CO1	L3	5M
		Unit -II			
4.	a)	Discuss the steps involved in designing Hadoop Distributed File System and Give the design of HDFS.	CO2	L1	5M
	b)	Write in detial about reading data from HDFS.	CO2	L1	5M
		(OR)			
5.	a)	Demonstrate the HDFS architecture along with its components indetail with neat sketch.	CO2	L3	5M
	b)	Explain how HDFS write operation is performed with neat sketch	CO2	L2	5M
		Unit -III			
6.		Explain in details about anatomy of Map Reduce job run.	CO3	L2	10M
		(OR)			
7.	a)	Explain briefly about shuffle and sort in Map Reduce .	CO3	L2	5M
	b)	Explain in detail about scheduling in YARN.	CO3	L4	5M
		Unit -IV			
8.	a)	Find Explain in detail about loading and storing the data in Pig with examples	CO4	L1	5M
	b)	Expl Explain Sqoop's import process with a neat sketch.	CO4	L3	5M
		(OR)			
9.	a)	Demonstrate briefly about Hive services.	CO4	L2	5M
	b)	Write the difference between the Traditional Databases and Hive.	CO4	L3	5M

a) What is Big Data?

"Big data" is high-volume, velocity, and variety information assets that demand cost-effective, innovative forms of information processing for enhanced insight and decision making."

b) Write the characteristics of big data.

1) volume
2) variety
3) velocity
4) veracity
5) value

c) Name hadoop configuration files.

- mapred-site.xml
- core-site.xml
- hdfs-site.xml
- hadoop-env.sh

d) What is the role of Job tracker and Task tracker in Map Reduce?

- **The job tracker**, which coordinates the job run. The job tracker is a Java application whose main class is Job Tracker.
- **The task trackers**, which run the tasks that the job has been split into. Task trackers are Java applications whose main class is Task Tracker.

e) List the features of Map Reduce.

- Highly scalable
- Versatile
- Affordability
- Secure
- Reliable

f) List the failures in classical Map Reduce.

- Task Failure
- Resource Manager Failure
- Application Master Failure
- Node Manager failure

g) Define Hive.

- Hive is a **data warehouse infrastructure** tool to process structured data in Hadoop.
- It resides on top of Hadoop to **summarize Big Data, and makes querying and analyzing easy**.

h) What is Pig?

- Apache Pig is an abstraction over MapReduce.
- It is a tool/platform which is used to analyse larger sets of data representing them as data flows.

i) What is Sqoop?

- Sqoop occupies a place in the Hadoop ecosystem to provide feasible interaction between relational database server and Hadoop's HDFS.

j) How many ways to run a Pig scripts?

- Local mode
- Map Reduce Mode

Unit-1

2. a) Identify the data storage problems and data analysis tasks problems prior to hadoop's file system.

Issue with Small Files. Hadoop does not suit for small data. ...

Slow Processing Speed. ...

Support for Batch Processing only. ...

No Real-time Data Processing. ...

No Delta Iteration. ...

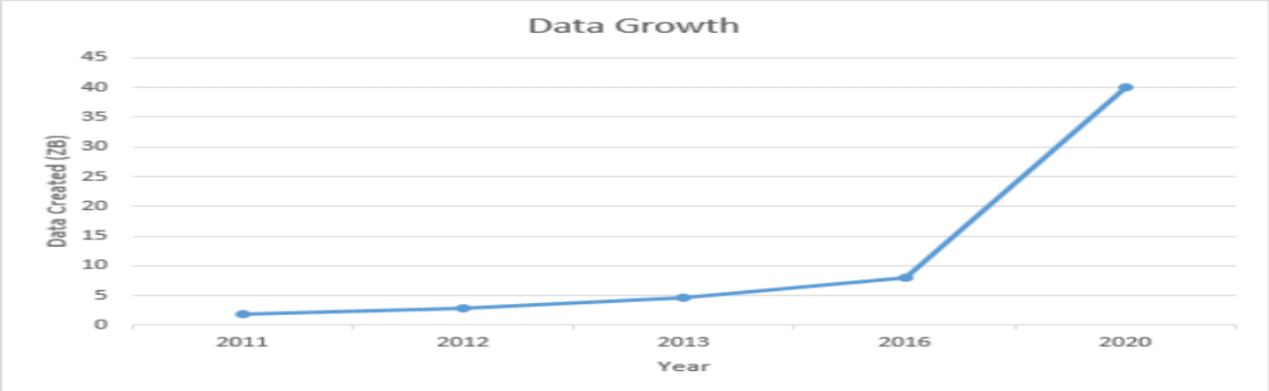
Latency. ...

Not Easy to Use. ...

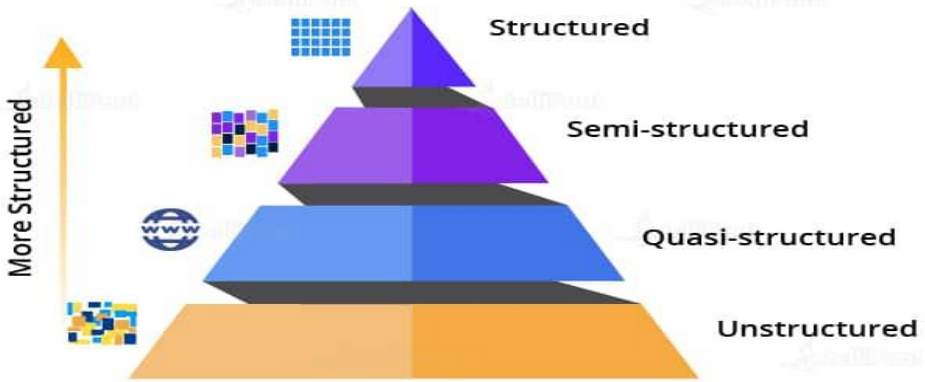
Security.

2. b) Explain the characteristics and applications of Big Data.

1**. Volume:** This refers to the data that is tremendously large. As you can see from the image, the volume of data is rising exponentially. In 2016, the data created was only 8 ZB and it is expected that, by 2020, the data would rise up to 40 ZB, which is extremely large.



2. Variety: A reason for this rapid growth of data volume is that the data is coming from different Sources in various formats.
The data is categorized as follows:



a) **Structured Data:** Here, data is present in a structured schema, along with all the required columns. It is in a structured or tabular format.

Emp ID	Emp Name	Gender	Department	Salary
2383	ABC	Male	Finance	6,50,000
4623	XYZ	Male	Admin	50,00,000

b) **Semi-structured Data:** In this form of data, the schema is not properly defined, i.e., both forms of data is present. So, basically semi-structured data has a structured form but it isn't defined, e.g., JSON, XML, CSV, TSV, and email.

c) **Unstructured Data:** In this data format, all the unstructured files such as video files, log files, audio files, and image files are included. Any data which has an unfamiliar model or structure is categorized as unstructured data. Since the size is large, unstructured data possesses various challenges in terms of processing for deriving value out of it.

d) **Quasi-structured Data:** This data format consists of textual data with inconsistent data formats that can be formatted with effort and time, and with the help of several tools. For example, web server logs, i.e., a log file that is automatically created and maintained by some server which contains a list of activities.

3. Velocity:

- The speed of data accumulation also plays a role in determining whether the data is categorized into big data or normal data.
  As can be seen from the image below, at first, mainframes were used wherein fewer people used computers.
- Then came the client/server model and more and more computers were evolved. After this, the web applications came into the picture and started increasing over the Internet.
- Then, everyone began using these applications.
- These applications were then used by more and more devices such as mobiles as they were very easy to access. Hence, a lot of data!

4. Value:

- How will the extraction of data work? Here, our fourth V comes in, which deals with a mechanism to bring out the correct meaning out of data.

- First of all, you need to mine the data, i.e., a process to turn raw data into useful data. Then, an analysis is done on the data that you have cleaned or retrieved out of the raw data.
- Then, you need to make sure whatever analysis you have done benefits your business such as in finding out insights, results, etc. which were not possible earlier.
- You need to make sure that whatever raw data you are given, you have cleaned it to be used for deriving business insights.

- After you have cleaned the data, a challenge pops up, i.e., during the process of dumping a huge amount of data, some packages might have lost.
  So for resolving this issue, our next V comes into the picture.

5. Veracity:

- Since the packages get lost during the execution, we need to start again from the stage of mining raw data in order to convert them into valuable data. And this process goes on.
- Also, there will be uncertainties and inconsistencies in the data.
- To overcome this, our last V comes into place, i.e., Veracity.
- Veracity means the trustworthiness and quality of data.
- It is necessary that the veracity of the data is maintained. For example, think about Facebook posts, with hashtags, abbreviations, images, videos, etc., which make them unreliable and hamper the quality of their content.
- Collecting loads and loads of data is of no use if the quality and trustworthiness of the data is not up to the mark.

OR

3. a) Explain in detail about Hadoop along with its architecture.

Hadoop

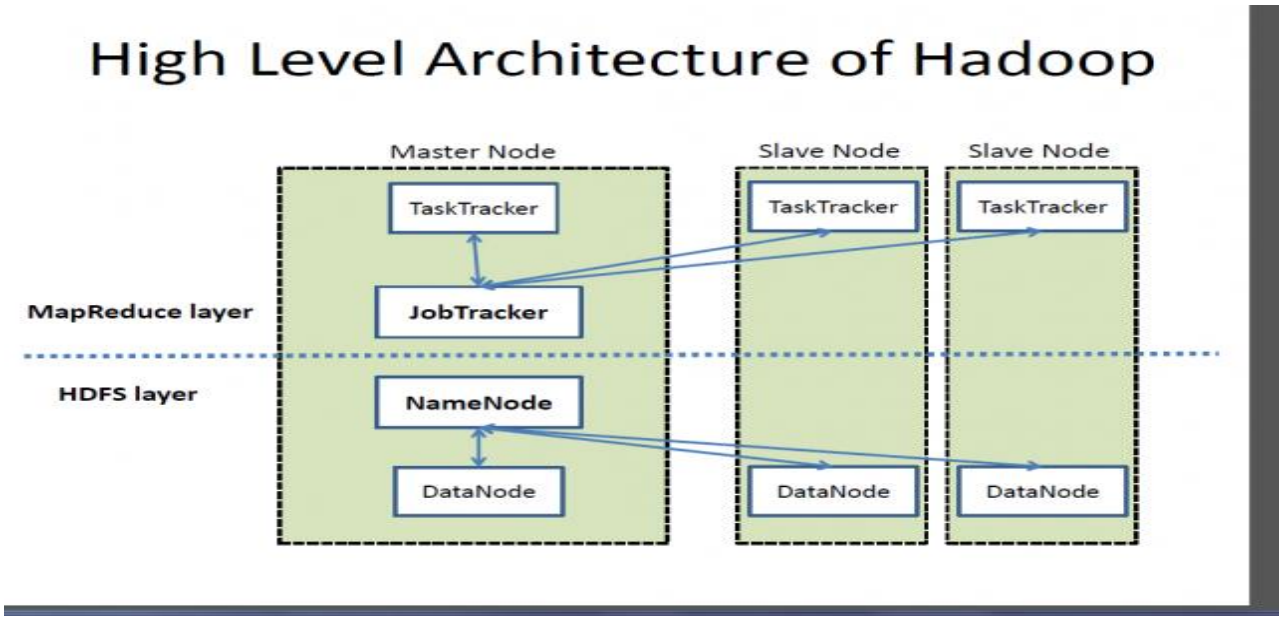
- Apache Hadoop is an open source software framework which was developed in JAVA for storage and large scale processing of data-sets on clusters of commodity hardware.
- Hadoop is an Apache top-level project being built and used by a global community of contributors and users.
- It is licensed under the Apache License 2.0.
- Yahoo created Hadoop in the year 2006, and it started using this technology by 2007. It was given to the Apache Software Foundation in 2008.
- In 2003, Doug Cutting had launched a project called Nutch. Nutch was created to handle the indexing of numerous web pages and billions of online search.

The Apache Hadoop framework is composed of the following modules or components

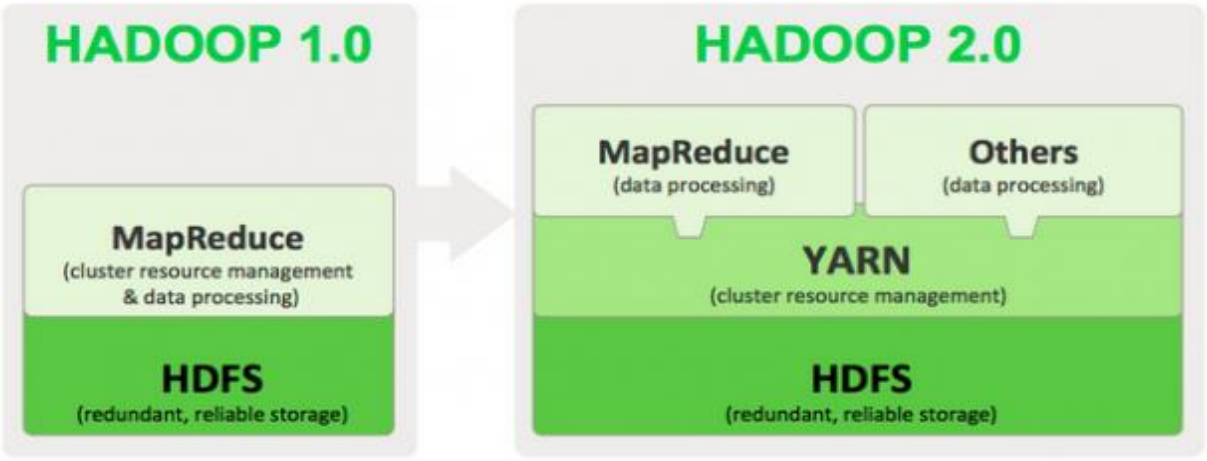
- Hadoop Common: contains libraries and utilities needed by other Hadoop modules
- Hadoop Distributed File System (HDFS): a distributed file-system that stores data on the commodity machines, providing very high aggregate bandwidth across the cluster
- Hadoop YARN: a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications.
- Hadoop MapReduce: a programming model for large scale data processing.

HADOOP ARCHITECTURE

- There are two primary components at the core of Apache Hadoop 1.x: Hadoop Distributed File System (HDFS) and the MapReduce parallel processing framework.
- These are both open source projects, inspired by technologies created inside Google.



- As part of Hadoop 2.0, YARN takes the resource management capabilities that were in MapReduce and packages them so they can be used by new engines. This also streamlines MapReduce to do what it does best, process data.
- With YARN, you can now run multiple applications in Hadoop, all sharing a common resource management. Many organizations are already building applications on YARN in order to bring them IN to Hadoop.

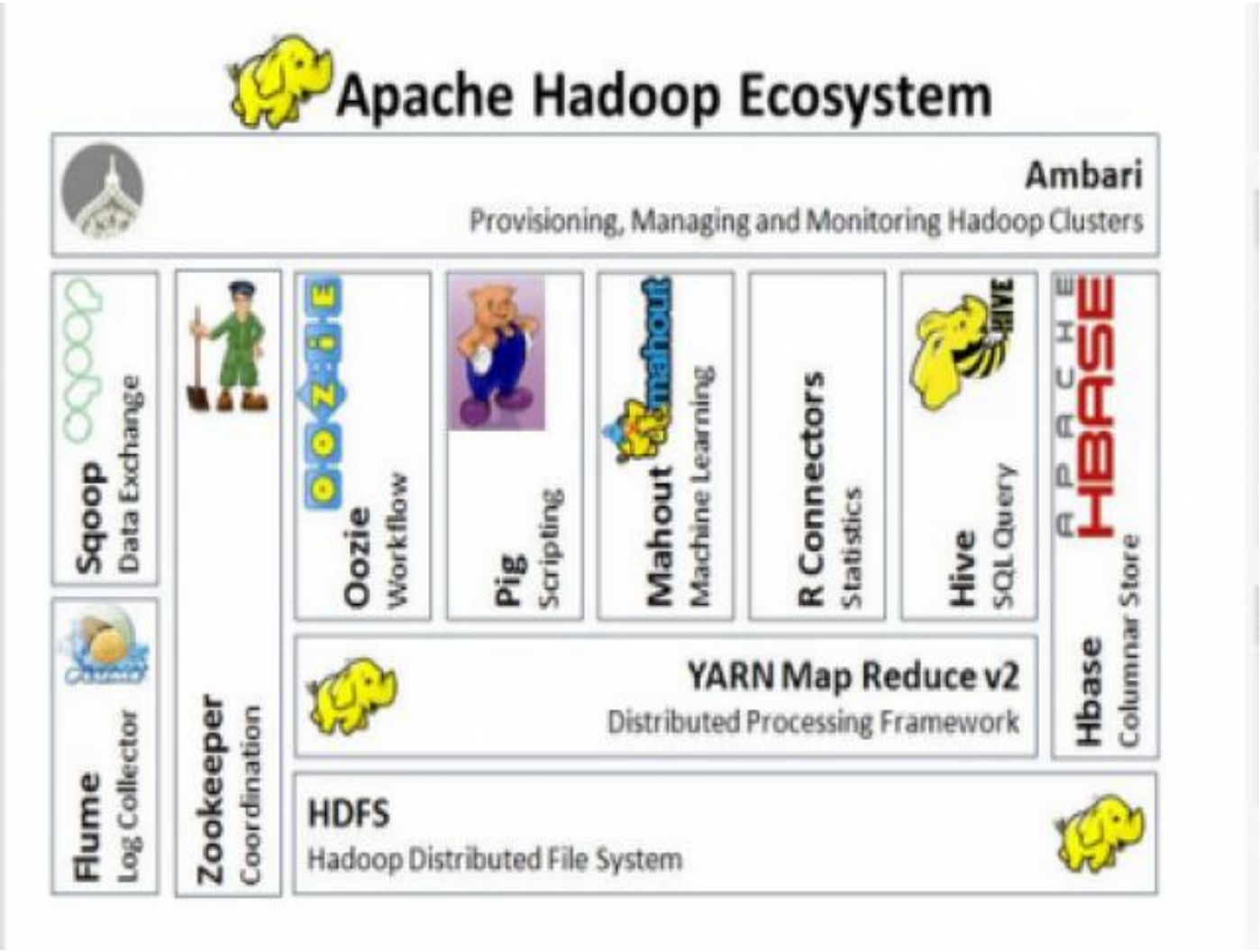


3. b) Explain in details about Hadoop Ecosystem.

Hadoop Ecosystem has

- Pig
- Hive
- Flume
- Sqoop
- Oozie

And so on..



- A set of machines running HDFS and MapReduce is known as **Hadoop Cluster** and individual machines are known as **nodes**.
- A cluster can have as few as one node or as many as several thousands of nodes.
- As the number of nodes are increased performance will be increased.
- The other languages excepts java(C++, RubyOnRails, Python, Perl)that are supported by Hadoop are called as Hadoop streaming.

Hadoop H/W & S/W requirements

- Hadoop usually runs on open source OS'(like Linux, Ubuntu etc)
  - Community Enterprise Operating System(CENTOS)
  - Red Hat Enterprise Linux Operating System(RHEL)
- If we have Windows it require virtualization s/w for running other OS on windows
  - VM player/VM workstation/Virtual Box
- Java is prerequisite for Hadoop installation.

UNIT-2

4. a) Discuss the steps involved in designing Hadoop Distributed File System and Give the design of HDFS.

HDFS::Introduction..

- A Master Node(high end configurations like dual power supply, dual n/w cards. Etc) called the Name Node(a node which is running name node daemon) keeps track of which blocks make up a file, and where those blocks are located , known as Meta Data.
- The Name Node keep track of the file metadata –which files are running in the system and how each file is broken down into the Name Node to keep the metadata current**.**

HDFS is a Optimizer for

- Large files
    - 100MB or more
- Streaming Data Access
    - Once we are writing data into HDFS, we can read the data for any number of time but we should not change the content of the file…
- Commodity Hardware

Five Daemons in HDFS

- **NamedNode -** Node that manages the Hadoop Distributed File System (HDFS).
- **DataNode** - Node where data is presented in advance before any processing takes place.
- **Job Tracker -** Schedules jobs and tracks the assign jobs to Task tracker.
- **Task Tracker -** Tracks the task and reports status to Job Tracker**.**
- **MasterNode** - Node where JobTracker runs and which accepts job requests from clients.

HDFS DESIGN:

- Apache HDFS or Hadoop Distributed File System is a block-structured file system where each file is divided into blocks of a pre-determined size.
- These blocks are stored across a cluster of one or several machines. Apache Hadoop HDFS Architecture follows a *Master/Slave Architecture*, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes).
- Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines**.**

NameNode:

- NameNode is the master node in the Apache Hadoop HDFS Architecture that maintains and manages the blocks present on the DataNodes (slave nodes).
- NameNode is a very highly available server that manages the File System Namespace and controls access to files by clients.
- The HDFS architecture is built in such a way that the user data never resides on the NameNode. The data resides on DataNodes only.

Functions of NameNode:

- It is the master daemon that maintains and manages the DataNodes (slave nodes)
- It records the metadata of all the files stored in the cluster, e.g. The **location of blocks** stored, the **size of the files**, permissions, hierarchy, etc. There are **two files associated with the metadata**:
    - **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode.
    - **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage.

DataNode:

- DataNodes are the slave nodes in HDFS.
- Unlike NameNode, DataNode is a commodity hardware, that is, a non-expensive system which is not of high quality or high-availability.

Functions of DataNode:

- These are slave daemons or process which runs on each slave machine.
- The actual data is stored on DataNodes.
- The DataNodes perform the low-level read and write requests from the file system's clients.
- They send heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds.

Secondary NameNode:

- Apart from these two daemons, there is a third daemon or a process called Secondary NameNode.
- The Secondary NameNode works concurrently with the primary NameNode as a helper daemon. And don't be confused about the Secondary NameNode being a backup NameNode because it is not.

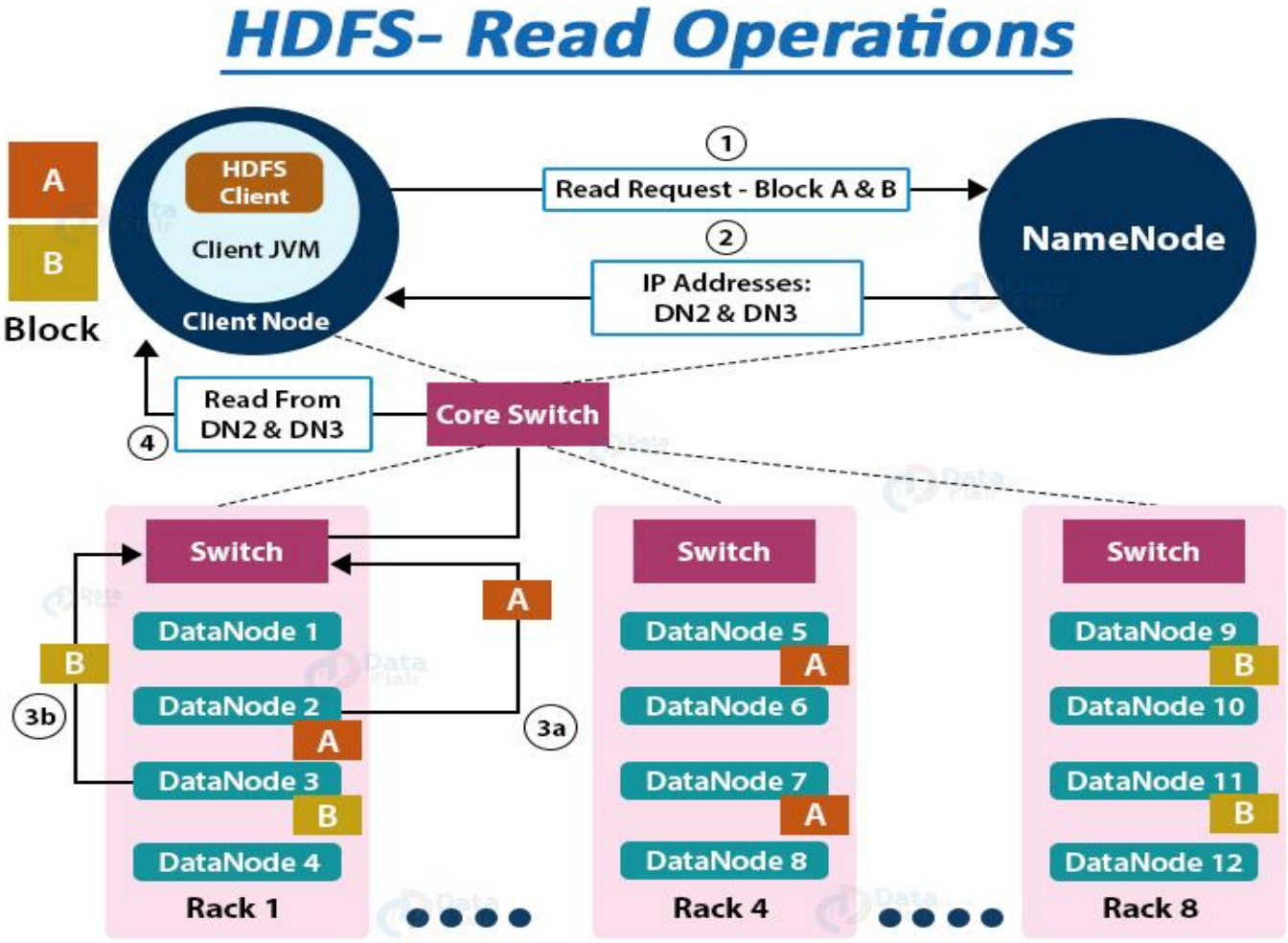
Functions of Secondary NameNode:

- The Secondary NameNode is one which constantly reads all the file systems and metadata

from the RAM of the NameNode and writes it into the hard disk or the file system.

- It is responsible for combining the EditLogs with FsImage from the NameNode.
- It downloads the EditLogs from the NameNode at regular intervals and applies to FsImage. The new FsImage is copied back to the NameNode, which is used whenever the NameNode is started the next time.
- Hence, Secondary NameNode performs regular checkpoints in HDFS. Therefore, it is also called CheckpointNode.

b) Write in detial about reading data from HDFS.

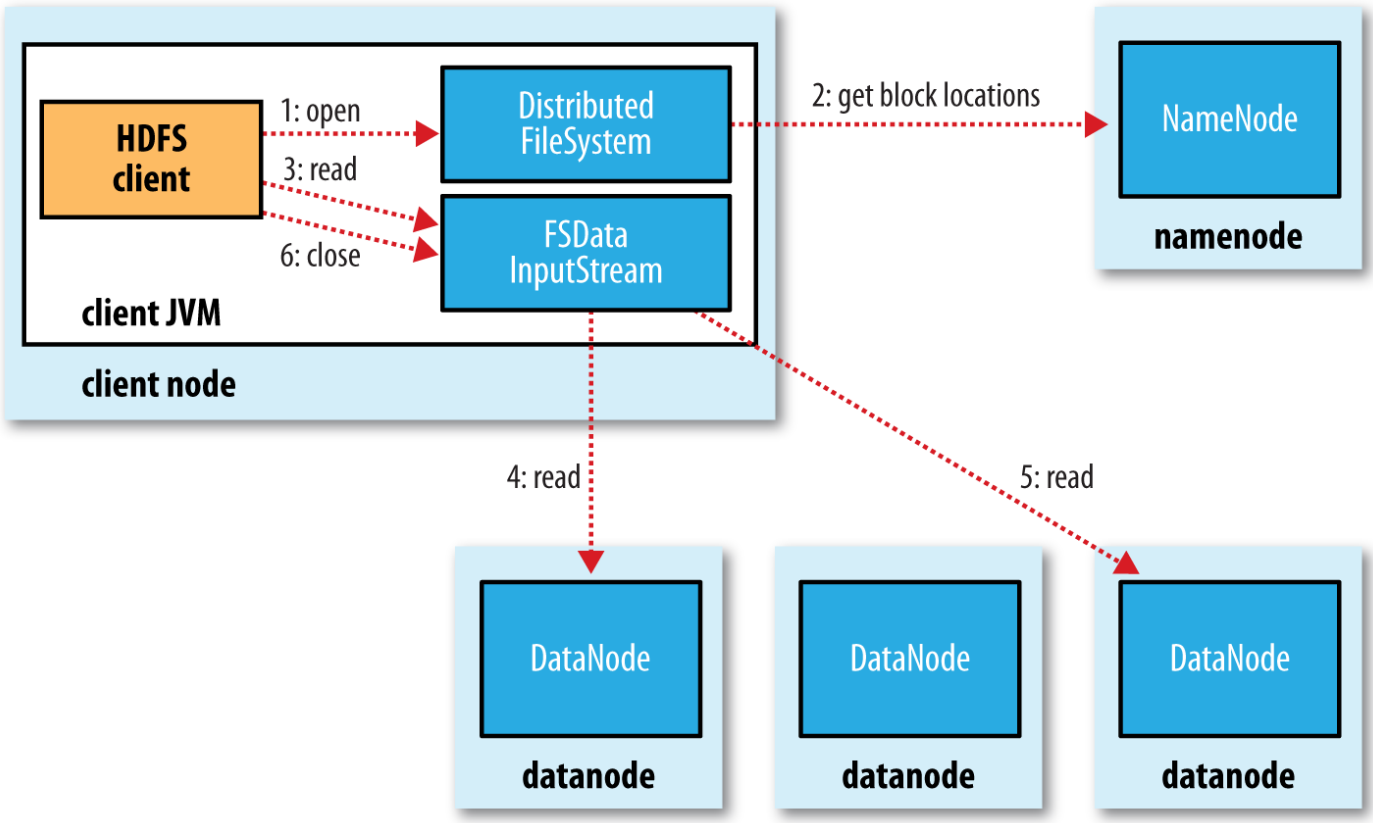


The Client interacts with HDFS NameNode

- As the NameNode stores the block's metadata for the file "File.txt', the client will reach out to NameNode asking locations of DataNodes containing data blocks.
- The NameNode first checks for required privileges, and if the client has sufficient privileges, the NameNode sends the locations of DataNodes containing blocks (A and B). NameNode also gives a security token to the client, which they need to show to the DataNodes for authentication. Let the NameNode provide the following list of IPs for block A and B – for block A, location of DataNodes D2, D5, D7, and for block B, location of DataNodes D3, D9, D11.
- To perform various HDFS operations (read, write, copy, move, change permission, etc.)

The client interacts with HDFS DataNode

- After receiving the addresses of the DataNodes, the client directly interacts with the DataNodes. The client will send a request to the closest DataNodes (D2 for block A and D3 for block B) through the FSDataInputstream object. The DFSInputstream manages the interaction between client and DataNode.
- The client will show the security tokens provided by NameNode to the DataNodes and start reading data from the DataNode. The data will flow directly from the DataNode to the client.
- After reading all the required file blocks, the client calls close() method on the FSDataInputStream object.



- In order to open the required file, the client calls the open() method on the FileSystem object, which for HDFS is an instance of DistributedFilesystem.
- DistributedFileSystem then calls the NameNode using RPC to get the locations of the first few blocks of a file.
- For each data block, NameNode returns the addresses of Datanodes that contain a copy of that block. Furthermore, the DataNodes are sorted based on their proximity to the client.
- The DistributedFileSystem returns an FSDataInputStream to the client from where the client can read the data. FSDataInputStream in succession wraps a DFSInputStream. DFSInputStream manages the I/O of DataNode and NameNode.
- Then the client calls the read() method on the FSDataInputStream object.
- The DFSInputStream, which contains the addresses for the first few blocks in the file, connects to the closest DataNode to read the first block in the file. Then, the data flows from DataNode to the client, which calls read() repeatedly on the FSDataInputStream.
- Upon reaching the end of the file, DFSInputStream closes the connection with that DataNode and finds the best suited DataNode for the next block.
- If the DFSInputStream during reading, faces an error while communicating with a DataNode, it will try the other closest DataNode for that block.
- Also, the DFSInputStream verifies checksums for the data transferred to it from the DataNode.
- If it finds any corrupt block, it reports this to the NameNode and reads a copy of the block from another DataNode.
- When the client has finished reading the data, it calls close() on the FSDataInputStream.

OR

5) a) Demonstrate the HDFS architecture along with its components indetail with neat sketch.

HDFS ARCHITECTURE:

- Apache HDFS or Hadoop Distributed File System is a block-structured file system where each file is divided into blocks of a pre-determined size.
- These blocks are stored across a cluster of one or several machines. Apache Hadoop HDFS Architecture follows a *Master/Slave Architecture*, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes).
- Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines**.**

NameNode:

- NameNode is the master node in the Apache Hadoop HDFS Architecture that maintains and manages the blocks present on the DataNodes (slave nodes).
- NameNode is a very highly available server that manages the File System Namespace and controls access to files by clients.
- The HDFS architecture is built in such a way that the user data never resides on the NameNode. The data resides on DataNodes only.

Functions of NameNode:

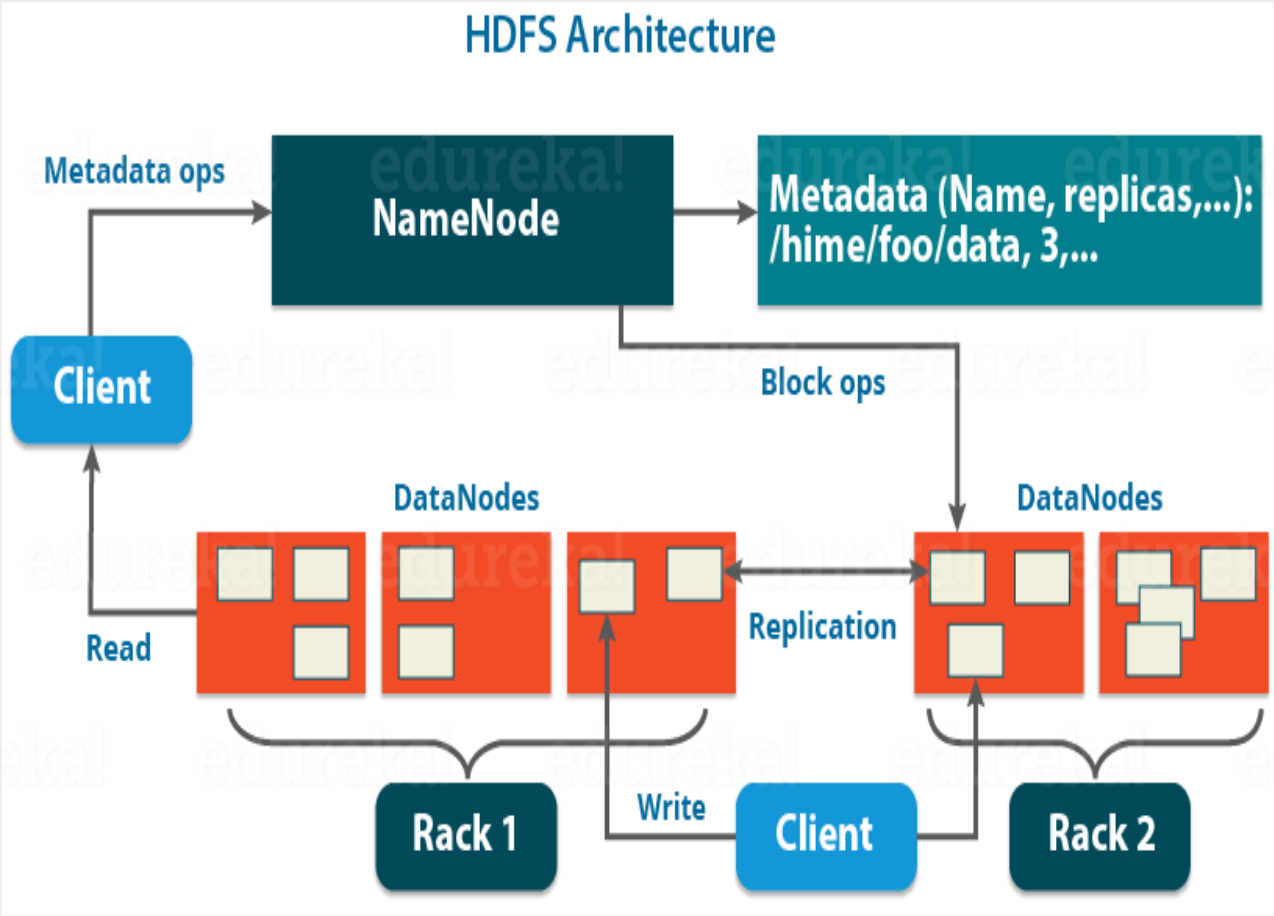
- It is the master daemon that maintains and manages the DataNodes (slave nodes)
- It records the metadata of all the files stored in the cluster, e.g. The **location of blocks** stored, the **size of the files**, permissions, hierarchy, etc. There are **two files associated with the metadata**:
  - **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode.
  - **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage.
- It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
- It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
- It keeps a record of all the blocks in HDFS and in which nodes these blocks are located.
- The NameNode is also responsible to take care of the **replication factor** of all the blocks which In **case of the DataNode failure**, the NameNode chooses new DataNodes for new replicas,balance disk usage and manages the communication traffic to the DataNodes.

DataNode:

- DataNodes are the slave nodes in HDFS.
- Unlike NameNode, DataNode is a commodity hardware, that is, a non-expensive system which is not of high quality or high-availability.

Functions of DataNode:

- These are slave daemons or process which runs on each slave machine.
- The actual data is stored on DataNodes.
- The DataNodes perform the low-level read and write requests from the file system’s clients.
- They send heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds.



Secondary NameNode:

- Apart from these two daemons, there is a third daemon or a process called Secondary NameNode.
- The Secondary NameNode works concurrently with the primary NameNode as a helper daemon. And don't be confused about the Secondary NameNode being a backup NameNode because it is not.

Functions of Secondary NameNode:

- The Secondary NameNode is one which constantly reads all the file systems and metadata from the RAM of the NameNode and writes it into the hard disk or the file system.
- It is responsible for combining the EditLogs with FsImage from the NameNode.
- It downloads the EditLogs from the NameNode at regular intervals and applies to FsImage. The new FsImage is copied back to the NameNode, which is used whenever the NameNode is started the next time.
- Hence, Secondary NameNode performs regular checkpoints in HDFS. Therefore, it is also called CheckpointNode.

5) b) Explain how HDFS write operation is performed with neat sketch

- To write data in HDFS, the client first interacts with the **NameNode** to get permission to write data and to get IPs of **DataNodes** where the client writes the data.
- The client then directly interacts with the DataNodes for writing data.
- The DataNode then creates a replica of the data block to other DataNodes in the pipeline based on the replication factor.
- To write a file inside the HDFS, the client first interacts with the NameNode.

The client interacts with HDFS NameNode

- NameNode first checks for the client privileges to write a file. If the client has sufficient privilege and there is no file existing with the same name, NameNode then creates a record of a new file.
- NameNode then provides the address of all DataNodes, where the client can write its data.
- It also provides a **security token to the client**, which they need to present to the DataNodes before writing the block.

- If the file already exists in the HDFS, then file creation fails, and the client receives an **IO Exception**.

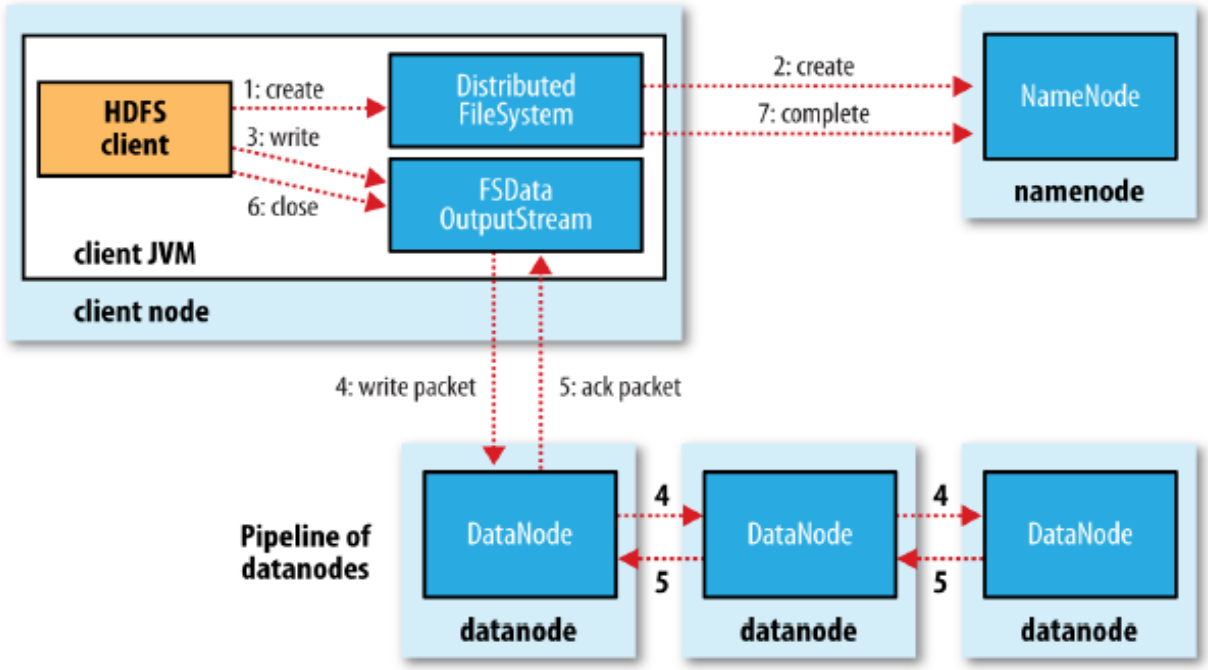
The client interacts with HDFS DataNode

- After receiving the list of the DataNodes and file write permission, the client starts writing data directly to the first DataNode in the list. As the client finishes writing data to the first DataNode, the DataNode starts making replicas of a block to other DataNodes depending on the replication factor.
- If the replication factor is 3, then there will be a minimum of 3 copies of blocks created in different DataNodes, and after creating required replicas, it sends an acknowledgment to the client.
- Thus it leads to the **creation of a pipeline**, and data replication to the desired value, in the cluster.

- The client creates the file by calling create() on **DistributedFileSystem**(step 1).
- DistributedFileSystem makes an RPC call to the name node to create a new file in the file system's namespace, with no blocks associated with it (step 2).
- The name node performs various checks to make sure the file doesn't already exist and that the client has the right permissions to create the file.
- If these **checks pass**, the name node makes a record of the new file; otherwise, file creation fails and the client is thrown an **IOException**.
- The DistributedFileSystem returns an **FSDataOutputStream** for the client to start writing data to.
- Just as in the read case, **FSDataOutputStream** wraps a DFSOutputStream, which handles communication with the data nodes and name node.
- As the client writes data (step 3), the **DFSOutputStream** splits it into packets, which it writes to an internal queue called the ***data queue***.
- *The data queue is consumed by the* **Data Streamer**, which is responsible for asking the name node to allocate **new blocks by picking a list of suitable data nodes to store the replicas.**
- The list of data nodes forms a **pipeline**, and here we'll assume the replication level is **three**, so there are **three nodes in the pipeline**.
- **The Data Streamer streams the packets to the first data node in the pipeline, which stores each packet and forwards it to the second data node in the pipeline.**

- Similarly, the second data node stores the packet and forwards it to the third (and last) data node in the pipeline (step 4).
- The DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowledged by data nodes, called the *ack queue.*
- *A packet is removed from the ack* queue only when it has been acknowledged by all the data nodes in the pipeline (step 5).
- If any data node fails while data is being written to it, then the following actions are taken, which are transparent to the client writing the data.
- First, the pipeline is closed, and any packets in the ack queue are added to the front of the data queue so that data nodes that are downstream from the failed node will not miss any packets.
- The failed data node is removed from the pipeline, and a new pipeline is constructed from the two good data nodes.
- The remainder of the block's data is written to the good data nodes in the pipeline.
- The name node notices that the block is under-replicated, and it arranges for a further replica to be created on another node.
- Subsequent blocks are then treated as normal.
- It's possible, but unlikely, for multiple data nodes to fail while a block is being written.

- As long as dfs.namenode.replication.min replicas (which defaults to 1) are written , the write will succeed, and the block will be asynchronously replicated across the cluster until its target replication factor is reached (dfs.replication, which defaults to 3).
- When the client has finished writing data, it calls close() on the stream (step 6).
- This action flushes all the remaining packets to the datanode pipeline and waits for acknowledgments before contacting the namenode to signal that the file is complete (step 7).
- The namenode already knows which blocks the file is made up of (because Data Streamer asks for block allocations), so it only has to wait for blocks to be minimally replicated before returning



successfully.

UNIT-3

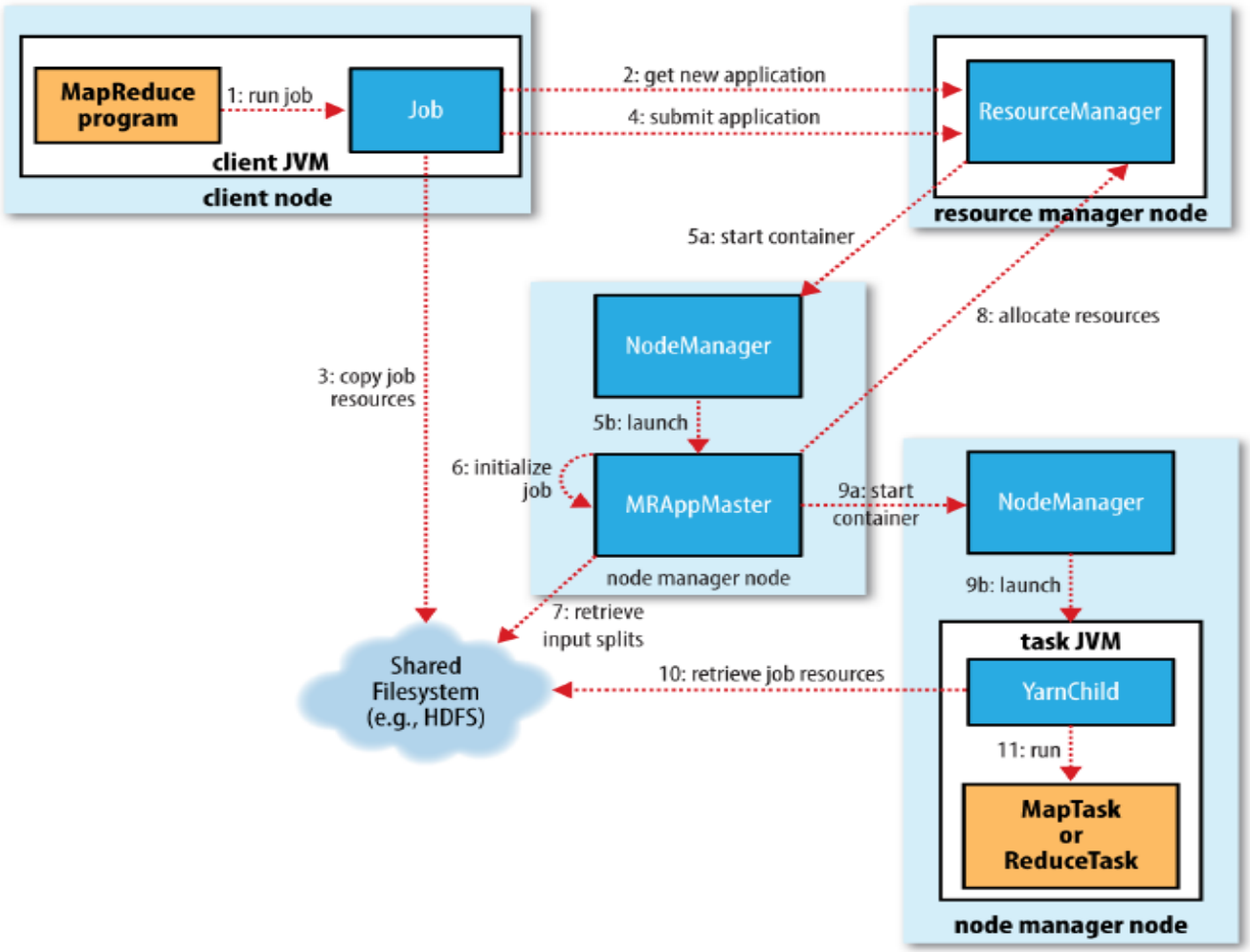
6) Explain in details about anatomy of Map Reduce job run.

ANATOMY OF MAPREDUCE RUN

- You can run a MapReduce job with a single method call: submit() on a Job object.
- You can also call waitForCompletion(), which submits the job if it hasn't been submitted already, then waits for it to finish.

At the highest level, there are **Five Independent Entities**:

- The **client**, which submits the MapReduce job.
- The **YARN resource manager**, which coordinates the allocation of compute resources on the cluster.
- The **YARN node managers**, which launch and monitor the compute containers on machines in the cluster.
- The **MapReduce application master**, which coordinates the tasks running the Map-Reduce job. The application master and the MapReduce tasks run in containers that are scheduled by the resource manager and managed by the node managers.
- The **distributed file system (normally HDFS),** which is used for sharing job files between the other entities.



How Hadoop runs a MapReduce job

- Job Submission
- Job Initialization
- Task Assignment
- Task Execution
- Progress and Updates
- Job Completion
- Failures

Job Submission

- The submit() method on Job creates an internal Job Submitter instance and calls **submit Job Internal()** on it.
- Having submitted the job, **waitForCompletion()** polls the job's progress once per second and reports the progress to the console if it has changed since the last report.
- When the job completes successfully, the **job counters are displayed**.
- Otherwise, the error that caused the job to fail is logged to the console.
- The job submission process implemented by **JobSubmitter** does the following:

1. Asks the **resource manager** for a **new application ID**, used for the **MapReduce jobID** (step 2).
2. Checks the **output specification** of the job. For example, if the output directory has not been specified or it already exists, the job is not submitted and an error is thrown to the MapReduce program.
3. Computes the **input splits for the job**. If the splits cannot be computed (because the input paths don't exist, for example), the job is not submitted and an error is thrown to the MapReduce program.
4. Copies the resources needed to run the job, including the **job JAR file, the configuration file, and the computed input splits**, to the shared filesystem in a directory named after the job ID (step 3).
5. The job JAR is copied with a high replication factor ,so that there are lots of copies across the cluster for the node managers to access when they run tasks for the job.
6. Submits the job by calling submitApplication() on the resource manager(step 4).

Job Initialization

- The application master must decide how to run the tasks that make up the MapReduce job. If the job is small, the application master may choose to run the tasks in the same JVM as itself.
- Such a job is said to be *uberized, or* run as an *uber task.*
- What qualifies as a small job? By default, a small job is one that has less than 10 mappers, only one reducer, and an input size that is less than the size of one HDFS block.
- Finally, before any tasks can be run, the application master calls the setupJob() method on the OutputCommitter.

Task Assignment

- If the job does not qualify for running as an uber task, then the application master requests containers for all the map and reduce tasks in the job from the resource manager(step 8).
- Requests for map tasks are made first and with a higher priority than those for reduce tasks, since all the map tasks must complete before the sort phase of the reduce can start.
- Requests for reduce tasks are not made until 5% of map tasks have completed.
- Reduce tasks can run anywhere in the cluster, but requests for map tasks have data locality constraints that the scheduler tries to honor.
- Requests also specify memory requirements and CPUs for tasks.
- By default, each map and reduce task is allocated 1,024 MB of memory and one virtual core.
-

TASK EXECUTION

- Once a task has been assigned,resources for a container on a particular node by the resource manager's scheduler, the application master starts the container by contacting the node manager (steps 9a and 9b).
- The task is executed by a Java application whose main class is YarnChild.
- Before it can run the task, it localizes the resources that the task needs, including the job configuration and JAR file, and any files from the distributed cache .
- Finally, it runs the map or reduce task (step 11).
- The YarnChild runs in a dedicated JVM, so that any bugs in the user-defined map and reduce functions don't affect the node manager—by causing it to crash or hang.
- Streaming
- Streaming runs special map and reduce tasks for the purpose of launching the user supplied executable and communicating with it.
-

Progress and Status Updates

- MapReduce jobs are long-running batch jobs, taking anything from tens of seconds to hours to run.
- A job and each of its tasks have a *status,* which includes such things as the state of the job or task ,the progress of maps and reduces, the values of the job's counters, and a status message or description.
- When a task is running, it keeps track of its *progress.*
- For map tasks, this is the proportion of the input that has been processed.
- For reduce tasks, it's a little more complex.

Job Completion

- When the application master receives a notification that the last task for a job is complete, it changes the status for the job to "successful."
- when the Job polls for status, it learns that the job has completed successfully, so it prints a message to tell the user and then returns from the waitForCompletion() method.
- Job statistics and counters are printed to the console at this point.
- Finally, on job completion, the application master and the task containers clean up their working state (so intermediate output is deleted).

Failures

- In the real world, user code is buggy, processes crash, and machines fail.
- One of the major benefits of using Hadoop is its ability to handle such failures and allow your job to complete successfully.

We need to consider the failure of any of the following entities:

- The task
- The application master
- The node manager
- The resource manager.

OR

7. a) Explain briefly about shuffle and sort in Map Reduce.

In Hadoop, the process by which the intermediate output from mappers is transferred to the reducer is called Shuffling. Reducer gets 1 or more keys and associated values on the basis of reducers. Intermediated key-value generated by mapper is sorted automatically by key. In this blog, we will discuss in detail about shuffling and Sorting in Hadoop MapReduce.

Here we will learn what is sorting in Hadoop, what is shuffling in Hadoop, what is the purpose of Shuffling and sorting phase in MapReduce, how MapReduce shuffle works and how MapReduce sort works.

Shuffling in MapReduce

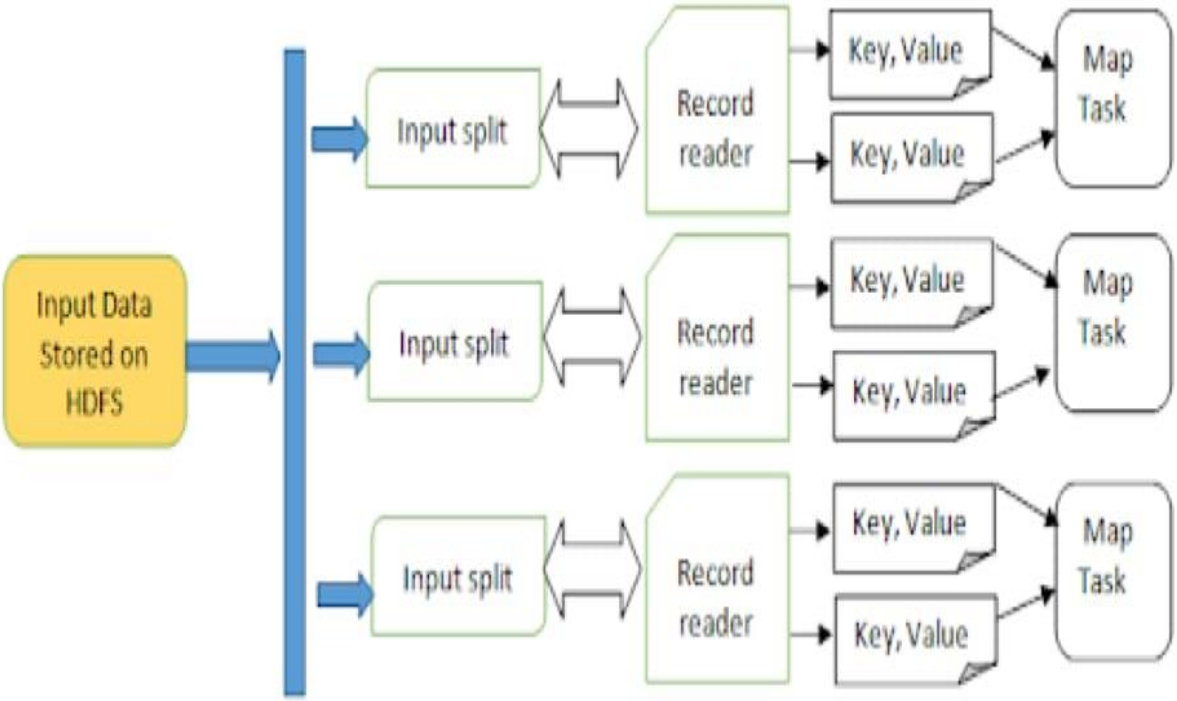
The process of transferring data from the mappers to reducers is known as shuffling i.e. the process by which the system performs the sort and transfers the map output to the reducer as input. So, MapReduce shuffle phase is necessary for the reducers, otherwise, they would not have any input (or input from every mapper). As shuffling can start even before the map phase has finished so this saves some time and completes the tasks in lesser time.

Sorting in MapReduce

The keys generated by the mapper are automatically sorted by MapReduce Framework, i.e. Before starting of reducer, all intermediate key-value pairs in MapReduce that are generated by mapper get sorted by key and not by value. Values passed to each reducer are not sorted; they can be in any order.

Sorting in Hadoop helps reducer to easily distinguish when a new reduce task should start. This saves time for the reducer. Reducer starts a new reduce task when the next key in the sorted input data is different than the previous. Each reduce task takes key-value pairs as input and generates key-value pair as output.

Note that shuffling and sorting in Hadoop MapReduce is not performed at all if you specify zero reducers (setNumReduceTasks(0)). Then, the MapReduce job stops at the map phase, and the map phase does not include any kind of sorting (so even the map phase is faster).



b) Explain in detail about scheduling in YARN.

Scheduling in YARN

The ResourceManager (RM) tracks resources on a cluster, and assigns them to applications that need them. The scheduler is that part of the RM that does this matching honoring organizational policies on sharing resources. Please note that:

YARN uses queues to share resources among multiple tenants. This will be covered in more detail in "Introducing Queues" below.

The ApplicationMaster (AM) tracks each task's resource requirements and coordinates container requests. This approach allows better scaling since the RM/scheduler doesn't need to track all containers running on the cluster.

Fair Scheduler

Fair Hadoop scheduler

FairScheduler allows YARN applications to fairly share resources in large Hadoop clusters. With FairScheduler, there is no need for reserving a set amount of capacity because it will dynamically balance resources between all running applications.

It assigns resources to applications in such a way that all applications get, on average, an equal amount of resources over time.

The FairScheduler, by default, takes scheduling fairness decisions only on the basis of memory. We can configure it to schedule with both memory and CPU.

When the single application is running, then that app uses the entire cluster resources. When other applications are submitted, the free up resources are assigned to the new apps so that every app eventually gets roughly the same amount of resources. FairScheduler enables short apps to finish in a reasonable time without starving the long-lived apps.

Similar to CapacityScheduler, the FairScheduler supports hierarchical queue to reflect the structure of the long shared cluster.

Apart from fair scheduling, the FairScheduler allows for assigning minimum shares to queues for ensuring that certain users, production, or group applications always get sufficient resources. When an app is present in the queue, then the app gets its minimum share, but when the queue doesn't need its full guaranteed share, then the excess share is split between other running applications.

UNIT-4

8. a) Find Explain in detail about loading and storing the data in Pig with examples.

LOAD

To load the data from the file system in to a relation.

- Basically, Load Operator loads data from the file system.
- LOAD 'data' [USING **function**] [AS schema];
- 1 2 3
  4 2 1
  8 3 4
- A = LOAD 'file1.txt';
- A = LOAD 'file1.txt' USING PigStorage('\t');
- DUMP A;
- (1,2,3)
  (4,2,1)
  (8,3,4)
- A = LOAD 'file1.txt' AS (f1:int, f2:int, f3:int); A = LOAD 'file1.txt' USING PigStorage('\t') AS (f1:int, f2:int, f3:int);
- DESCRIBE A;
- a: {f1: int,f2: int,f3: int}
- ILLUSTRATE A;

a	f1: bytearray	f2: bytearray	f3: bytearray
	4	2	1

STORE

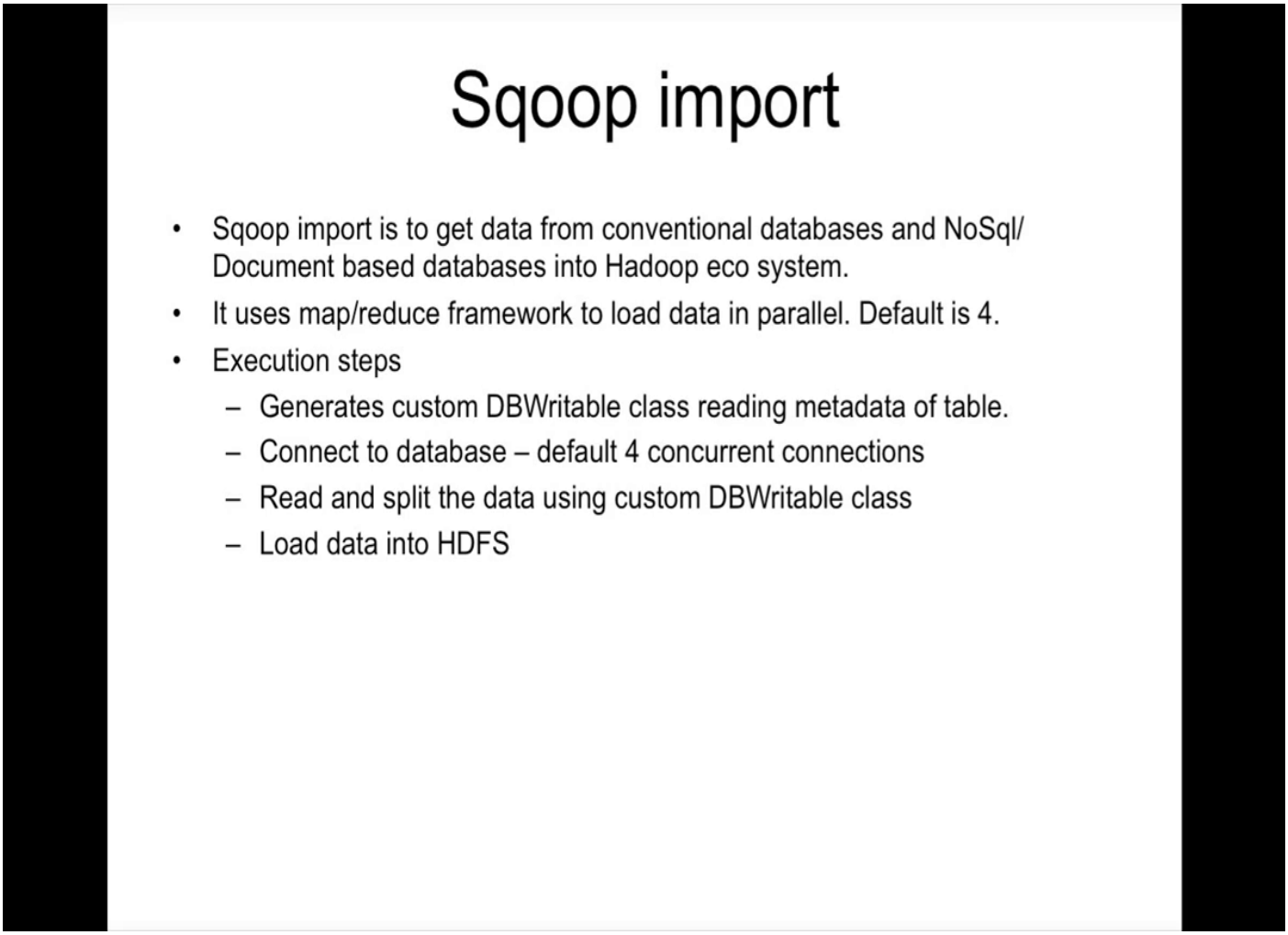
To save a relation to the file system.

- Basically, STORE operator, Stores or saves results to the file system.
- STORE alias INTO 'directory' [USING **function**];
- A = LOAD 'data' AS (a1:int,a2:int,a3:int);
- DUMP A;
- (1,2,3)
  (4,2,1)
  (8,3,4)
  (4,3,3)
  (7,2,5)
  (8,4,3)
- STORE A INTO 'output1' USING PigStorage ('*');
- CAT output1;
- 1*2*3
  4*2*1
  8*3*4
  4*3*3
  7*2*5
  8*4*3

b) Explain Sqoop's import process with a neat sketch.

- The 'Import tool' imports individual tables from RDBMS to HDFS. Each row in a table is treated as a record in HDFS. All records are stored as text data in the text files or as binary data in Avro and Sequence files.
- Syntax
- The following syntax is used to import data into HDFS.
- $ sqoop import (generic-args) (import-args)

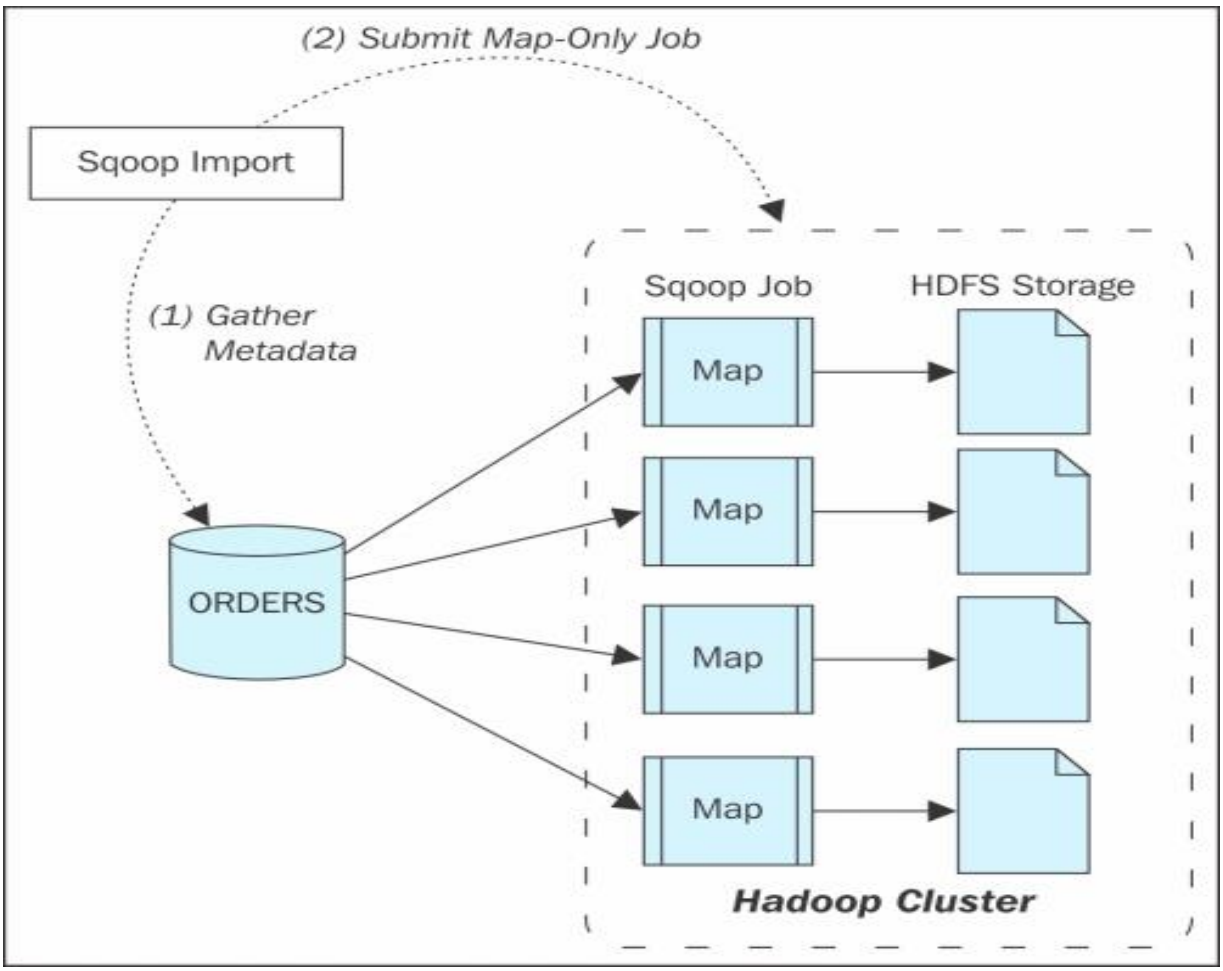
- Importing a Table
- Sqoop tool ‘import’ is used to import table data from the table to the Hadoop file system as a text file or a binary file.



- **Import an entire table:**

```
sqoop import \
 --connect
jdbc:mysql://mysql.example.com/sqoop \
--username sqoop \
--password sqoop \
--table cities
```

- Import a subset of data:
  sqoop import \ --connect jdbc:mysql://mysql.example.com/sqoop \
  --username sqoop \
  --password sqoop \
  --table cities \
  --where "country = 'USA'"
- IMPORT-ALL-TABLES
- The following syntax is used to import all tables.
  $ sqoop import-all-tables (generic-args) (import-args)
  $ sqoop-import-all-tables (generic-args) (import-args)
- The following command is used to import all the tables from the **userdb database.**
  **$ sqoop import **
  **--connect jdbc:mysql://localhost/userdb **
  --username root
- The following command is used to verify all the table data to the userdb database in HDFS.
  $ $HADOOP_HOME/bin/hadoop fs –ls



OR

9. a) Demonstrate briefly about Hive services.

Hive provides various services like the Hive server2, Beeline, etc. The various services offered by Hive are:

1. Beeline
The Beeline is a command shell supported by HiveServer2, where the user can submit its queries and command to the system. It is a JDBC client that is based on SQLLINE CLI (pure Java-console-based utility for connecting with relational databases and executing SQL queries).

2. Hive Server 2
HiveServer2 is the successor of HiveServer1. HiveServer2 enables clients to execute queries against the Hive. It allows multiple clients to submit requests to Hive and retrieve the final results. It is basically designed to provide the best support for open API clients like JDBC and ODBC.

3. Hive Driver
The Hive driver receives the HiveQL statements submitted by the user through the command shell. It creates the session handles for the query and sends the query to the compiler.

4. Hive Compiler
Hive compiler parses the query. It performs semantic analysis and type-checking on the different query blocks and query expressions by using the metadata stored in metastore and generates an execution plan. The execution plan created by the compiler is the DAG(Directed Acyclic Graph), where each stage is a map/reduce job, operation on HDFS, a metadata operation.

5. Optimizer
Optimizer performs the transformation operations on the execution plan and splits the task to improve efficiency and scalability.

6. Execution Engine
Execution engine, after the compilation and optimization steps, executes the execution plan created by the compiler in order of their dependencies using Hadoop.

7. Metastore
Metastore is a central repository that stores the metadata information about the structure of tables and partitions, including column and column type information.

b) Write the difference between the Traditional Databases and Hive.

RDBMS	Hive
It is used to maintain database.	It is used to maintain data warehouse.
It uses SQL (Structured Query Language).	It uses HQL (Hive Query Language).
Schema is fixed in RDBMS.	Schema varies in it.
Normalized data is stored.	Normalized and de-normalized both type of data is stored.
Tables in rdms are sparse.	Table in hive are dense.
It doesn't support partitioning.	It supports automation partition.
No partition method is used.	Sharding method is used for partition.