14CS/IT 703

Hall Ticket Number:

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

January, 2021 **Common to CSE and IT**

Seventh Semester **Advanced Data Analytics**

Time: Three Hours **Maximum :** 60 Marks

Answer ALL Questions from PART-A. (1X12 = 12 Marks)
*Answer **ANY FOUR** questions from PART-B.* (4X12=48 Marks)

Part - A

1 Answer all questions (1X12=12 Marks)

a) What is Big Data?

b) What are the applications of big data?
Healthcare, Government Sector, Retail Industry, Ecommerce applications.

c) Define namenodes and datanodes in HDFS.
Namenode: Master node of HDFS, Responsible of allocation of Datanodes to store Data.
Datanode: Slave node of HDFS, Responsible for storing data upon request from namenode.

d) Why blocks are large in HDFS?
To save large data thus by avoiding fragmentation and segmentation
Or
If the blocks are small in size then we have more no of replications which leads to poor managing of data.

e) What is the default scheduler in Hadoop?
Capacity Scheduler

f) What is the role of Jobtracker and Tasktracker in Map Reduce?
Job Tracker: Responsible for scheduling jobs on task tracker, manages resources in cluster, starting the failed nodes, executing jobs upon request from client.
Task Tracker: Responsible for executing the jobs upon request from job tracker, sending heart beat signals about execution of jobs.

g) List the failures in classical Map Reduce.
Job tracker failure
Task Tracker failure
Task Failure
Child JVM Failure

h) List the built-in counters in Map Reduce.
Maapper counter
Reducer counter
No of bytes read
No bytes write
Record counter

i) How to display the records in a relation R using Pig Latin.
By using DUMP operator we can display records on console.
By using STORE operator we can save the records into a file and then DUMP the file.

j) Compare Managed table with external table in Hive.
Managed tables are Hive owned tables where the entire lifecycle of the tables' data are managed and controlled by Hive. External tables are tables where Hive has loose coupling with the data. Replication

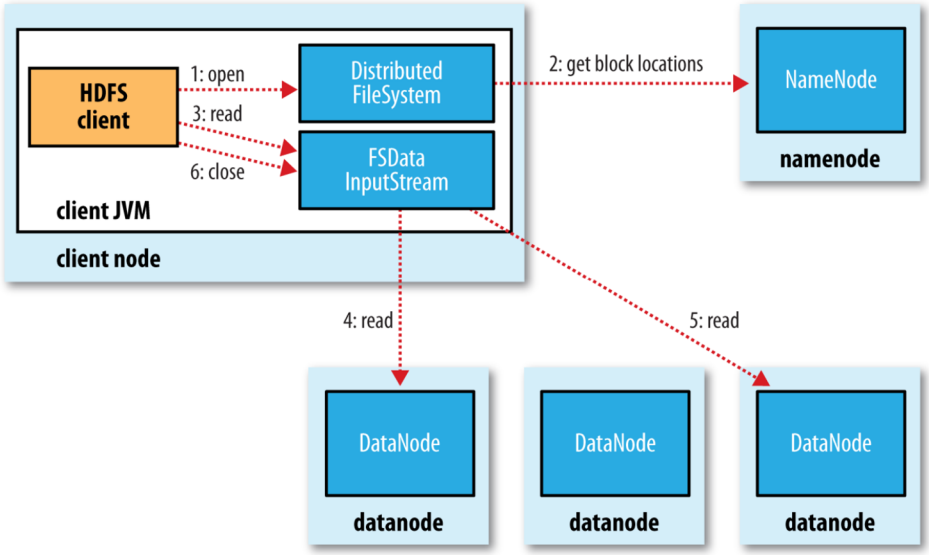
Manager replicates external tables successfully to a target cluster. The managed tables are converted to external tables after replication.

k) How to read and print a text file using spark?
textFile() method is used to **read** a **text file from** HDFS, S3 and any Hadoop supported **file** system, this method takes the path as an argument and optionally takes a number of partitions as the second argument.

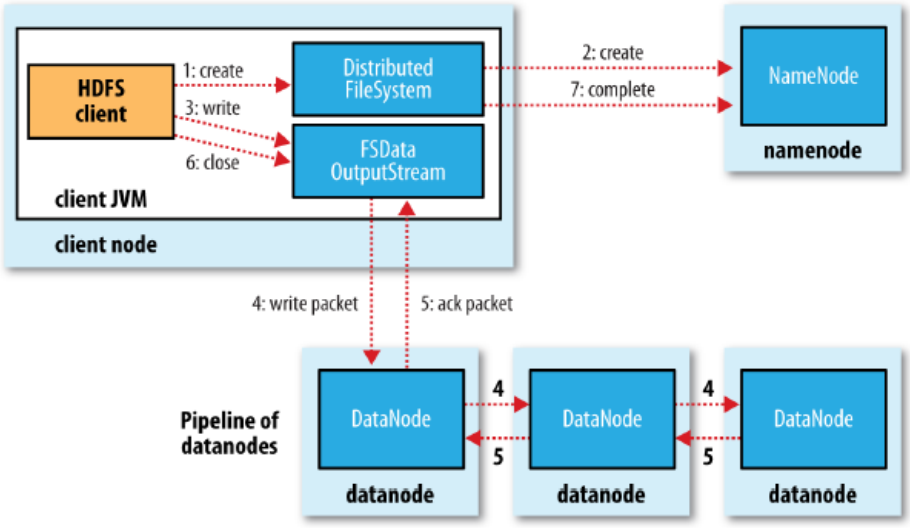
l) Write any one application of Apache Sqoop?
- Sqoop is a tool designed to transfer data between Hadoop and relational database servers.

Part - B

2 Show how a client read and write data in HDFS with suitable code. 12M



- The client opens the file it wishes to read by calling open() on the File System object, which for HDFS is an instance of DistributedFileSystem (step-1)
- DistributedFileSystem calls the name node, using remote procedure calls (RPCs), to determine the locations of the first few blocks in the file (step 2).
- The client then calls read() on the stream (step 3). DFSInputStream, which has stored the data node addresses for the first few blocks in the file, then connects to the first (closest) data node for the first block in the file.
- Data is streamed from the data node back to the client, which calls read() repeatedly on the stream (step 4).
- When the end of the block is reached, DFSInputStream will close the connection to the data node, then find the best data node for the next block (step 5).
- When the client has finished reading, it calls close() on the FSDataInputStream (step 6).



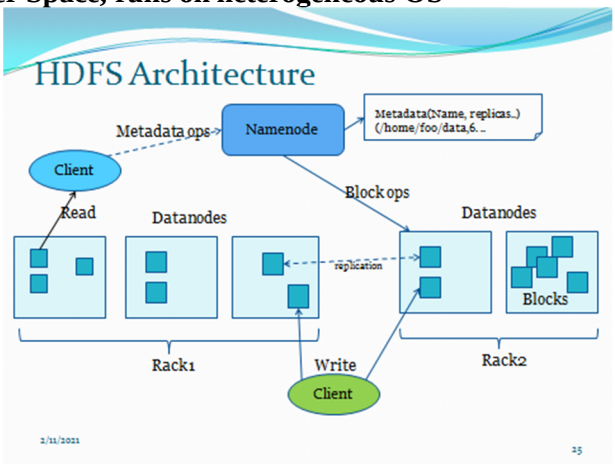
4

- 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).
- 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.
- Similarly, the second data node stores the packet and forwards it to the third (and last) data node in the pipeline (step 4).
- *A packet is removed from the ack* queue only when it has been acknowledged by all the data nodes in the pipeline (step 5).
- 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).

3 Discuss the steps involved in designing Hadoop Distributed File System and Give the design of HDFS. 12M

- 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).
- These blocks will be replicated three times and stored in local file systems as a separate file.
- Blocks are replicated across a multiple machines, known as DataNodes.
- DataNode is slave machine in Hadoop cluster running the data node daemon.
- Daemon-A process continuously running
- 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.
- **NamedNode** - Node that manages the Hadoop Distributed File System (HDFS).
- **DataNode** - Node where data is presented in advance before any processing takes place.
- **MasterNode** - Node where JobTracker runs and which accepts job requests from clients.
- **SlaveNode** - Node where Map and Reduce program runs.
- **Job Tracker** - Schedules jobs and tracks the assign jobs to Task tracker.
- **Task Tracker** - Tracks the task and reports status to Job Tracker.
- **Job** - A program is an execution of a Mapper and Reducer across a dataset.
- **Task** - An execution of a Mapper or a Reducer on a slice of data.

- **Very Large Distributed File System**
  - 10K nodes, 100 million files, 10 PB
- **Assumes Commodity Hardware**
  - Files are replicated to handle hardware failure
  - Detect failures and recovers from them
- **Optimized for Batch Processing**
  - Data locations exposed so that computations can move to where data resides
  - Provides very high aggregate bandwidth
- **User Space, runs on heterogeneous OS**



DataNode

HANBORQ

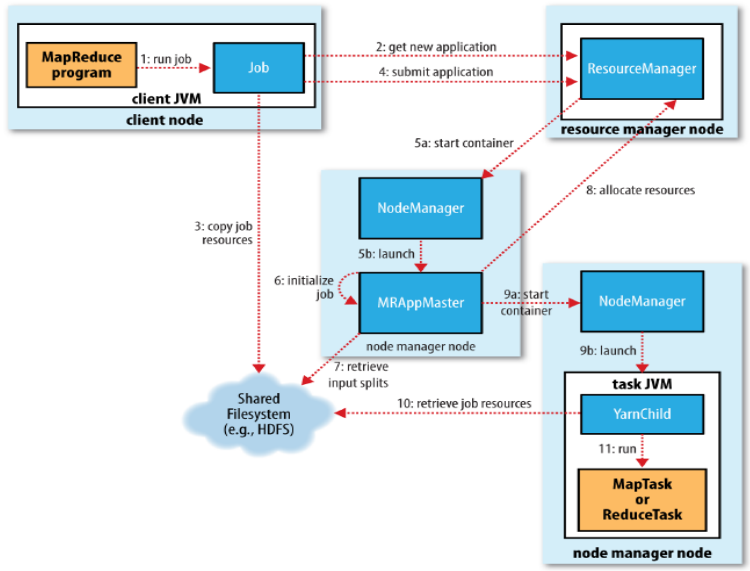
- Store data blocks
  - Have no knowledge about FSName
- Receive blocks from Client
- Receive blocks from DataNode peer
  - Replication
  - Pipeline writing
- Receive delete command from NameNode

NameNode FS Meta

HANBORQ

- FSImage
  - FSNames & FSName → Blocks
  - Saved replicas in multiple name directory
  - Recover on startup
- EditLog
  - Log every FS modification
- Block → Replicas (DataNodes)
  - Only in memory
  - rebuilt from block reports on startup

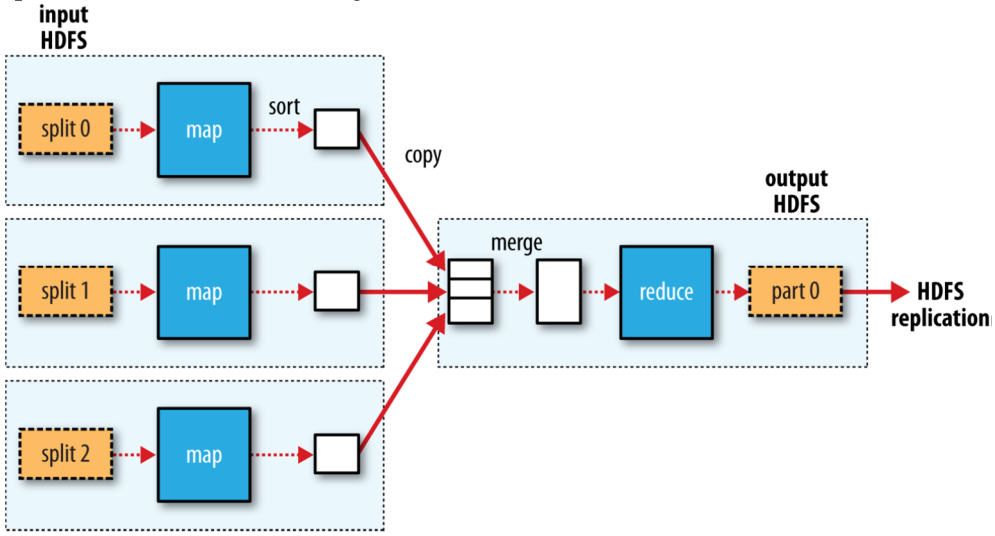
4 a) Analyze the data with Hadoop using Map and Reduce with an example. 6M



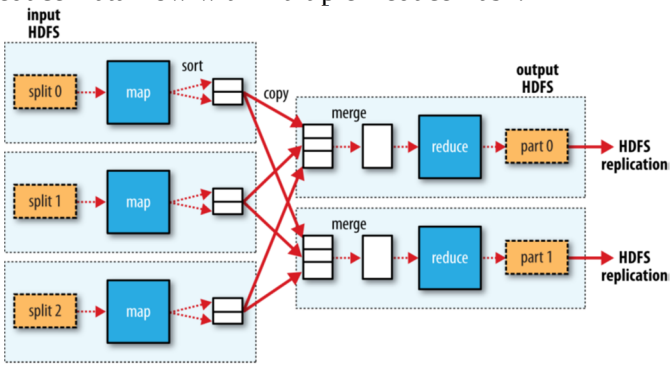
- Job Submission
- Job Initialization
- Task Assignment
- Task Execution
- Progress and Updates
- Job Completion

b) Explain in detail about MapReduce data flow with single and multiple reduce tasks. 6M

Map Reduce Data flow with single Reduce Task:



Map Reduce Data flow with Multiple Reduce Task:



5 a) How do you support the statement that, “YARN is better than MapReduce 1”? 6 M

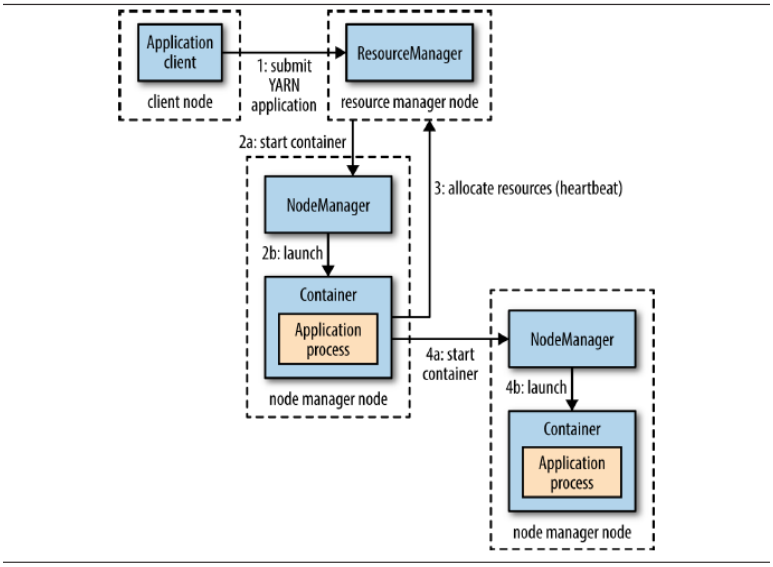
Yarn contains 10000 nodes, Mapreduce1 contains 4000 nodes

Yarn can run any application in its application master, Map reduce runs only mapreduce application

Yarn supports scheduling by using resource manager, Mapreduce supports scheduling by using Job tracker despite of its many functionalities which becomes bottleneck for job tracker

Yarn can reuse the contaiers, where as mapreduce cant reuse the child jvms

b) How YARN runs a MapReduce application? 6 M



- To run an application on YARN, a client contacts the resource manager and asks it to run an *application master process (step 1).*
- The resource manager then finds a node manager that can launch the application master in a container (steps 2a and 2b).
- It request more containers from the resource managers (step 3), and use them to run a distributed computation (steps 4a and 4b).

6 a) Explain about Pig Latin expressions and Pig Latin types. 6M

Pig Latin – Data types

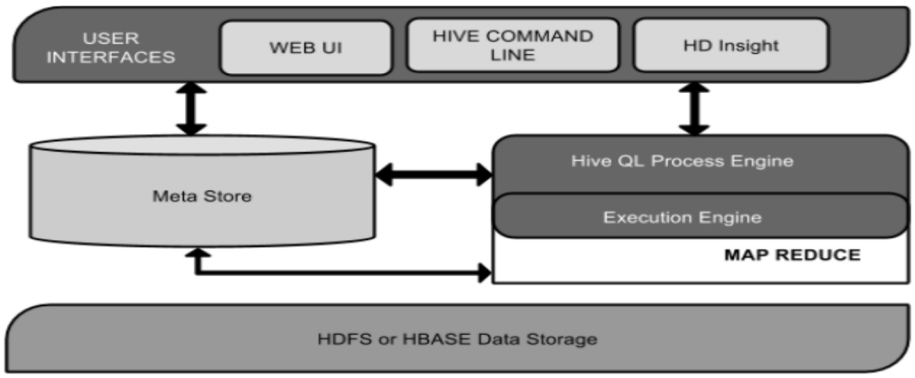
Given below table describes the Pig Latin data types.

Data Type	Description and Example
int	Represents a signed 32-bit integer. **Example:** 8
long	Represents a signed 64-bit integer. **Example:** 5L
float	Represents a signed 32-bit floating point. **Example:** 5.5F
double	Represents a 64-bit floating point. **Example:** 10.5
chararray	Represents a character array (string) in Unicode UTF-8 format. **Example:** 'tutorials point'
Bytearray	Represents a Byte array (blob).
Boolean	Represents a Boolean value. **Example:** true/ false.
Datetime	Represents a date-time. **Example:**1970-01-01T00:00:00.000+00:00
Biginteger	Represents a Java BigInteger. **Example:** 60708090709
Bigdecimal	Represents a Java BigDecimal **Example:** 185.98376256272893883
Complex Types	
Tuple	A tuple is an ordered set of fields. **Example:** (raja, 30)
Bag	A bag is a collection of tuples. **Example:** {(raju,30),(Mohhammad,45)}
Map	A Map is a set of key-value pairs. **Example:**['name'#'Raju', 'age'#30]

b) Explain about data processing operators in Pig. 6M

Statement	Description
Load	Read data from the file system
Store	Write data to the file system
Dump	Write output to stdout
Foreach	Apply expression to each record and generate one or more records
Filter	Apply predicate to each record and remove records where false
Group / Cogroup	Collect records with the same key from one or more inputs
Join	Join two or more inputs based on a key
Order	Sort records based on a Key
Distinct	Remove duplicate records
Union	Merge two datasets
Limit	Limit the number of records
Split	Split data into 2 or more sets, based on filter conditions

7 Write about the Hive architecture and how it works? 12M



- **User Interface :**

Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).

- **Meta Store :**

Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.

- **HiveQL Process Engine :**

HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.

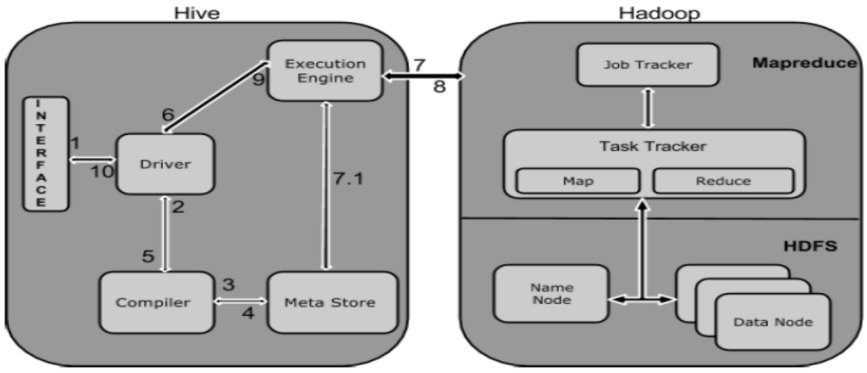
- **Execution Engine :**

The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.

- **HDFS or HBASE :**

Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

Working with HIVE



- **Step No 1 :**

Execute Query

The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.

- **Step No 2 :**

Get Plan

The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.

- **Step No 3:**

Get Metadata

The compiler sends metadata request to Metastore (any database).

- **Step No 4:**

Send Metadata

Metastore sends metadata as a response to the compiler.

- **Step No 5:**

Send Plan

The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.

- **Step No 6:**

Execute Plan

The driver sends the execute plan to the execution engine.

- **Step No 7:**

Execute Job

Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.

- **Step No 7.1:**

Metadata Ops

Meanwhile in execution, the execution engine can execute metadata operations with Metastore.

- **Step No 8:**

Fetch Result
The execution engine receives the results from Data nodes.

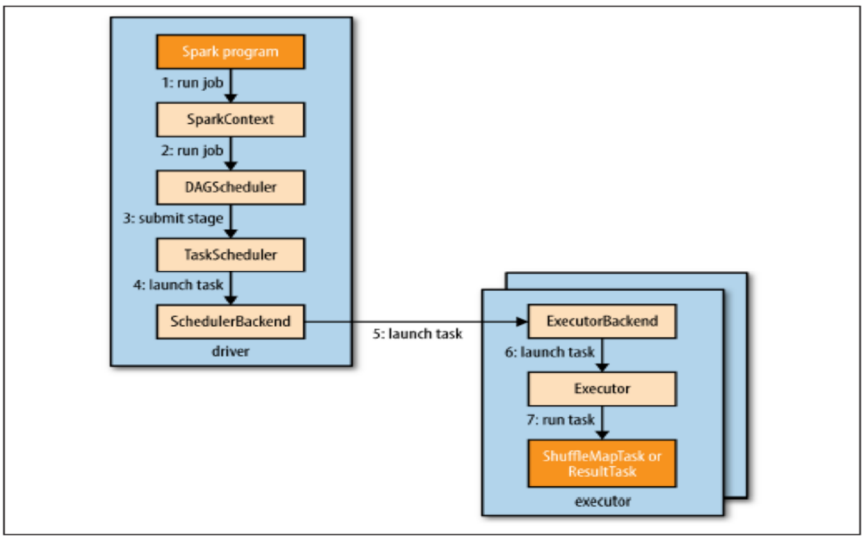
- **Step No 9:**

Send Results
The execution engine sends those resultant values to the driver.

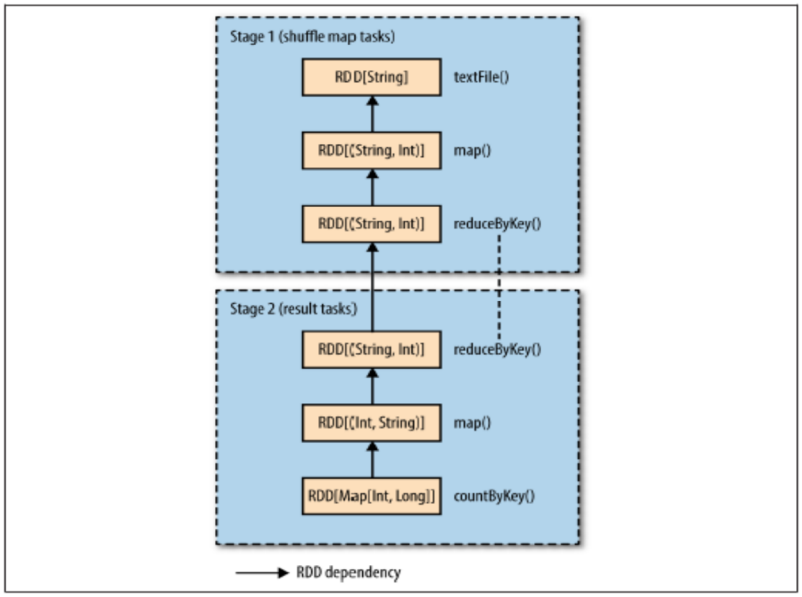
- **Step No 10:**

Send Results
The driver sends the results to Hive Interfaces.

8 a) Explain the Anatomy of spark job run. 6M



- **Job Submission :**
- Spark job is submitted automatically when an action (such as count()) is performed on an RDD. Internally, this causes runJob() to be called on the SparkContext(step 1), which passes the call on to the scheduler that runs as a part of the driver (step 2).
- The scheduler is made up of two parts: a DAG scheduler that breaks down the job into a DAG of stages, and a task scheduler that is responsible for submitting the tasks from each stage to the cluster.
- **DAG Construction :**
- To understand how a job is broken up into stages, we need to look at the type of tasks that can run in a stage.
- There are two types: *shuffle map tasks and result tasks.*
- *The name* of the task type indicates what Spark does with the task's output:
- ***Shuffle map tasks***
- As the name suggests, shuffle map tasks are like the map-side part of the shuffle in MapReduce. Each shuffle map task runs a computation on one RDD partition and, based on a partitioning function, writes its output to a new set of partitions, which are then fetched in a later stage (which could be composed of either shuffle map tasks or result tasks). Shuffle map tasks run in all stages except the final stage.
- ***Result tasks***
- Result tasks run in the final stage that returns the result to the user's program (such as the result of a count()). Each result task runs a computation on its RDD partition,then sends the result back to the driver, and the driver assembles the results from each partition into a final result (which may be Unit, in the case of actions like saveAsTextFile()).
- The simplest Spark job is one that does not need a shuffle and therefore has just a single stage composed of result tasks. This is like a map-only job in MapReduce.
- More complex jobs involve grouping operations and require one or more shuffle stages.
- For example, consider the following job for calculating a histogram of word counts for text files stored in inputPath (one word per line):



- The RDDs within each stage are also, in general, arranged in a DAG. The diagram shows the type of the RDD and the operation that created it.
- RDD[String] was created by textFile(), for instance.
- To simplify the diagram, some intermediate RDDs generated internally by Spark have been omitted.
- For example, the RDD returned by textFile() is actually a MappedRDD[String] whose parent is a HadoopRDD[LongWritable,Text].
- If an RDD has been persisted from a previous job in the same application (SparkContext), then the DAG scheduler will save work and not create stages for recomputing it(or the RDDs it was derived from).
- The DAG scheduler is responsible for splitting a stage into tasks for submission to the task scheduler.
- Each task is given a placement preference by the DAG scheduler to allow the task scheduler to take advantage of data locality. A task that processes a partition of an input RDD stored on HDFS, for example, will have a placement preference for the datanode hosting the partition's block (known as *node local), while a task that processes a partition* of an RDD that is cached in memory will prefer the executor storing the RDD partition(*process local).*
- once the DAG scheduler has constructed the complete DAG of stages, it submits each stage's set of tasks to the task scheduler (step 3).
- Child stages are only submitted once their parents have completed successfully.

b) Explain briefly about task scheduling and task execution in spark. 6M

Task Scheduling in Spark

When the task scheduler is sent a set of tasks, it uses its list of executors that are running for the application and constructs a mapping of tasks to executors that takes placement preferences into account. Next, the task scheduler assigns tasks to executors that have free cores (this may not be the complete set if another job in the same application is running), and it continues to assign more tasks as executors finish running tasks, until the task set is complete. Each task is allocated one core by default, although this can be changed by setting spark.task.cpus.

Note that for a given executor the scheduler will first assign process-local tasks, then node-local tasks, then rack-local tasks, before assigning an arbitrary (nonlocal) task, or a speculative task if there are no other candidates.

Assigned tasks are launched through a scheduler backend (step 4 in Figure 19-1), which sends a remote launch task message (step 5) to the executor backend to tell the executor to run the task.

Executors also send status update messages to the driver when a task has finished or if a task fails. In the latter case, the task scheduler will resubmit the task on another executor.

It will also launch speculative tasks for tasks that are running slowly, if this is enabled (it is not by default).

Task Execution

An executor runs a task as follows

1. It makes sure that the JAR and file dependencies for the task are up to date. The executor keeps a local cache of all the dependencies that previous tasks have used, so that it only downloads them when they have changed.
2. It de-serializes the task code (which includes the user's functions) from the serialized bytes that were sent as a part of the launch task message.
3. The task code is executed. Note that tasks are run in the same JVM as the executor, so there is no process overhead for task launch.

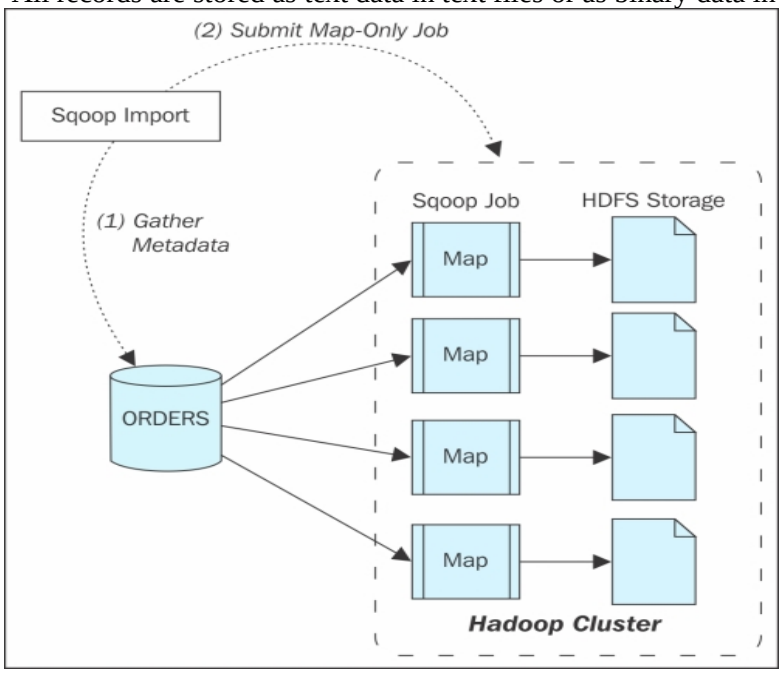
Tasks can return a result to the driver. The result is serialized and sent to the executor backend, and then back to the driver as a status update message.

A shuffle map task returns information that allows the next stage to retrieve the output partitions, while a result task returns the value of the result for the partition it ran on, which the driver assembles into a final result to return to the user's program.

9 a) Describe Sqoop import command. 6M

Sqoop Import

- 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 text files or as binary data in Avro and Sequence files



- 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:

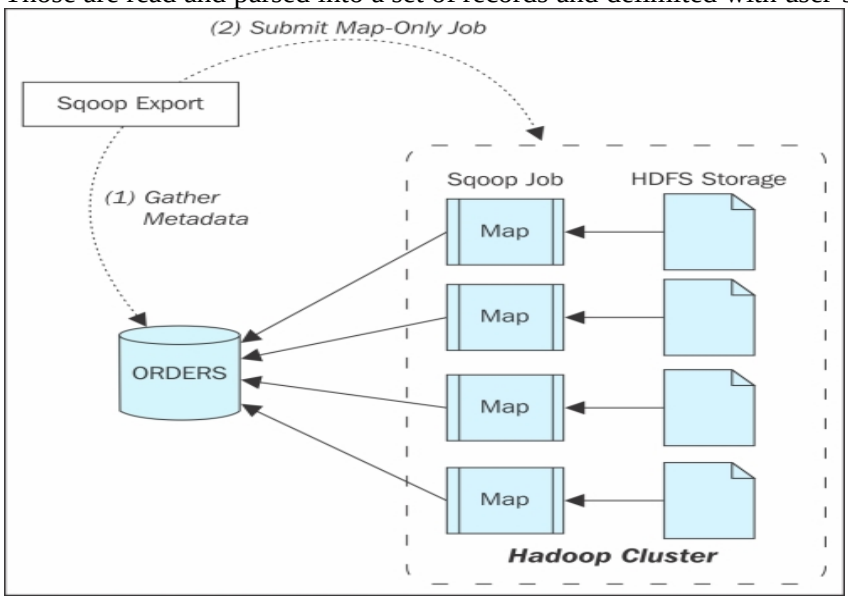
sqoop import \ --connect jdbc:mysql://mysql.example.com/sqoop \
--username sqoop \
--password sqoop \
--table cities \
--where "country = 'USA'"

- 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

b) How Sqoop export command works? 6M

Sqoop Export

- The export tool exports a set of files from HDFS back to an RDBMS.
- The files given as input to Sqoop contain records, which are called as rows in table.
- Those are read and parsed into a set of records and delimited with user-specified delimiter.



- The default operation is to insert all the record from the input files to the database table using the INSERT statement.
- In update mode, Sqoop generates the UPDATE statement that replaces the existing record into the database.
- The following is the syntax for the export command.
- **$ sqoop export (generic-args) (export-args)**
- **$ sqoop-export (generic-args) (export-args)**
- The following command is used to export the table data (which is in **emp_data file** on HDFS) to the employee table in db database of Mysql database server.
- **$ sqoop export **
- **--connect jdbc:mysql://localhost/db **
- **--username root **
- **--table employee **
- **--export-dir /emp/emp_data**