Scheme of Valuation

ADVANCED DATA ANALYTICS

CS/IT422

B.Tech.,(Semester- VIII)

Date	:	March, 2017	Semester End Examination	:	Regular
Duration	:	3 hrs.	Maximum Marks	:	60

I **(1X12 = 12 Marks)**

(a) **What is the role of namenode in Hadoop Cluster?**
Answer:
The namenode holds in it's memory the metadata for all the files and blocks available in the Hadoop cluster.

(b) **What is the default size of HDFS block?**
Answer:
The HDFS block size is 128 MB by default.

(c) **Write the URL to access namenode service through web interface in a single node Hadoop cluster.**
Answer:
The URL to access namenode service through web interface in a single node Hadoop cluster is `http://localhost:50070`.

(d) **How to run Pig in local execution environment?**
Answer:
The command to run Pig in local execution environment is pig -x local.

(e) **Write Pig Latin script to find number of records in a relation (table) R.**
Answer:

```
TR = GROUP R ALL;
grouped_TR = FOREACH TR GENERATE group, COUNT(R);
DUMP grouped_TR;
```

(f) **How to display the records in a relation R using Pig Latin statement.**
Answer:

```
DUMP R;
```

(g) **Write briefly about map data type in Hive query language.**
Answer: A column by name "contacts" in a relation can be declared as map type as follows `contacts MAP<STRING, BIGINT>`. The data in the column can be as follows `personal:9876543210,official:9123456789`, if ":" and "," are used as the delimiters for map keys and collection items respectively. The personal number of the contacts column can be selected using the syntax `contacts['personal']`.

(h) **Compare Managed table with External table in Hive.**
Answer:
Data loaded into managed table is moved to the warehouse directory of Hive, where as it is kept in the original location when data is loaded into external table. If a managed table is dropped both data and metadata are deleted where as only metadata is deleted by Hive in case of External table.

(i) **What is schema on read in the context of Hive?**
Answer: Hive doesn't verify the data, against the schema of the table, when it is loaded into the table. Only when a query is invoked on the table Hive verifies the data. This is called schema on read.

(j) **What is a Resilient Distributed Dataset (RDD)?**
Answer:
A read-only collection, of objects, that is partitioned across multiple machines in a cluster is called Resilient Distributed Dataset (RDD).

(k) **Write any one application of Apache Sqoop?**
Answer:
Apache Sqoop is an open source tool that allows users to extract data from a structured data store into Hadoop cluster for further processing.

(l) **How to read and print a text file using Spark framework?**
Answer:

```
val lines = sc.textFile("input.txt")
lines.foreach(println(_))
```

where sc is an instance of SparkContext class.

UNIT - I

II (a) **Describe a typical topology for a fully distributed Hadoop cluster.**

(3 Marks)

Answer:

< 3 Marks >

A common Hadoop cluster architecture consists of a two-level network topology, as illustrated in Fig.1. Typically there are 30 to 40 servers per rack (only 3 are shown in the diagram), with a 10 Gb switch for the rack and an uplink to a core switch or router (at least 10 Gb or better). The salient point is that the aggregate bandwidth between nodes on the same rack is much greater than that between nodes on different racks.

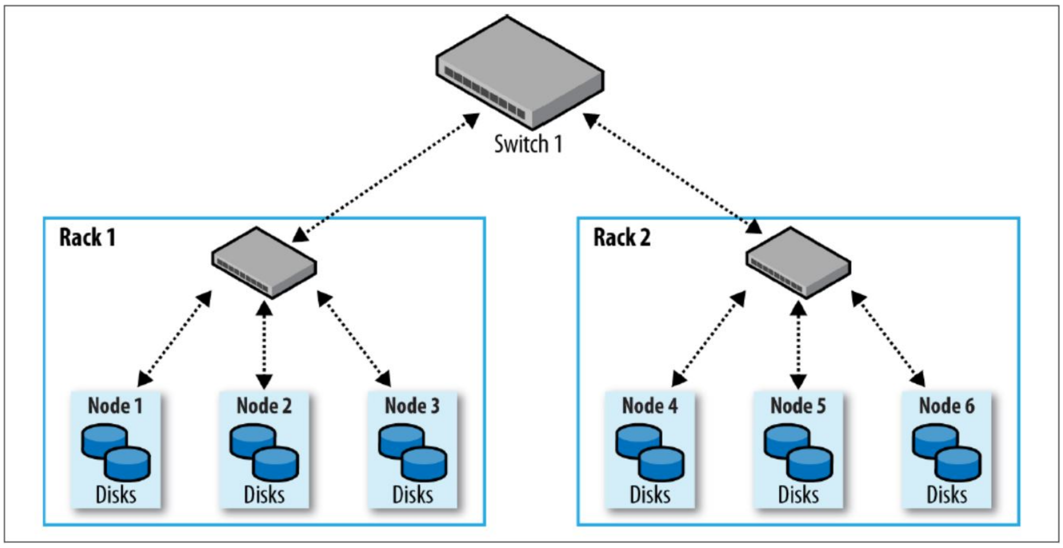


Figure 1: Typical two-level network architecture for a Hadoop cluster

(b) **Write a mapreduce program in Java to find word frequencies in a given set of text files.** **(9 Marks)**

Answer:

< 3 Marks >

Listing 1: Mapper for word count

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable>{
  private String tokens = "[_|$#<>\\^=\\[\\]\\*/\\\\,;,.\\-:()?!\"']";
  public void map(LongWritable recadd, Text rec, Context con) throws IOException,
      InterruptedException {
    String cleanline = rec.toString().toLowerCase().replaceAll(tokens, " ");
    String[] words = cleanline.split(" ");
    for(String kw : words) {
      con.write(new Text(kw.trim()), new IntWritable(1));
    }

  }
}
```

< 2 Marks >

Listing 2: Reducer for word count

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer extends
    Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context con)
      throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable el : values) {
      sum = sum + el.get();
    }
    con.write(key, new IntWritable(sum));
  }
}
```

< 4 Marks >

Listing 3: Driver for word count

```
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCountJob extends Configured implements Tool {
public static void main(String[] cla) throws Exception {
  int exitstatus = ToolRunner.run(new WordCountJob(), cla);
  System.exit(exitstatus);
}

@Override
public int run(String[] args) throws Exception {
  Job jb = Job.getInstance(getConf());

  jb.setJobName("Word Count");

  jb.setMapperClass(WordCountMapper.class);
  jb.setReducerClass(WordCountReducer.class);

  jb.setMapOutputKeyClass(Text.class);
  jb.setMapOutputValueClass(IntWritable.class);

  jb.setOutputKeyClass(Text.class);
  jb.setOutputValueClass(IntWritable.class);

  jb.setJarByClass(WordCountJob.class);

  FileInputFormat.setInputPaths(jb, new Path(args[0]));
  FileOutputFormat.setOutputPath(jb, new Path(args[1]));

  return jb.waitForCompletion(true) ? 0 : 1;
}
}
```

OR

III (a) **Describe the important daemon properties used to configure HDFS service.**

(6 Marks)

Answer:

< 4 properties each 1.5 Marks >

Property name **Type** **Default value**	**Description**
fs.defaultFS URI file:///	The default filesystem. The URI defines the hostname and port that the namenode's RPC server runs on. The default port is 8020. This property is set in core-site.xml.
dfs.namenode.name.dir Comma-separated directory names file://${hadoop.tmp.dir}/dfs/name	The list of directories where the namenode stores its persistent metadata. The namenode stores a copy of the metadata in each directory in the list.
dfs.datanode.data.dir Comma-separated directory names file://${hadoop.tmp.dir}/dfs/data	A list of directories where the datanode stores blocks. Each block is stored in only one of these directories.
dfs.namenode.checkpoint.dir Comma-separated directory names file://${hadoop.tmp.dir}/dfs/namesecondary	A list of directories where the secondary name node stores checkpoints. It stores a copy of the checkpoint in each directory in the list.

(b) **How to configure YARN service?**

(6 Marks)

Answer:

< 4 properties each 1.5 Marks >

Property name **Type** **Default value**	**Description**
yarn.resourcemanager.hostname Hostname 0.0.0.0	The hostname of the machine the resource manager runs on. Abbreviated $y.rm.hostname below.
yarn.resourcemanager.address Hostname and port $y.rm.hostname:8032	The hostname and port that the resource manager's RPC server runs on.
yarn.nodemanager.local-dirs Comma-separated directory names $hadoop.tmp.dir/nm-local-dir	A list of directories where node managers allow containers to store intermediate data. The data is cleared out when the application ends.
yarn.nodemanager.aux-services Comma-separated service names	A list of auxiliary services run by the node manager. A service is implemented by the class defined by the property yarn.nodemanager.auxservices. servicename.class. By default, no auxiliary services are specified.
yarn.nodemanager.resource.memory-mb int 8192	The amount of physical memory (in MB) that may be allocated to containers being run by the node manager.
yarn.nodemanager.vmem-pmem-ratio float 2.1	The ratio of virtual to physical memory for containers. Virtual memory usage may exceed the allocation by this amount.
yarn.nodemanager.resource.cpu-vcores int 8	The number of CPU cores that may be allocated to containers being run by the node manager.

UNIT - II

IV (a) **Write a Pig Latin script to find maximum temperature in a given year given the weather data with three fields Year, Temperature and Quality Parameter (QP) per record as depicted in Table 1. Discard the records with temperature value 9999 or QP value in 2, 3, 6, 7, 8. The range of QP is 0 to 9.**

(6 Marks)

Answer:

< First 4 lines each 1.5 Marks >

Listing 4: Pig Latin code to find maximum temperature in an year

```
records = LOAD 'input/wdata.txt' AS (Year:chararray, Temperature, QP:int);

filtered_records = FILTER records BY Temperature != 9999 AND QP IN (0, 1, 4, 5, 9);

grouped_records = GROUP filtered_records BY Year;

max_temp = FOREACH grouped_records GENERATE group, MAX(filtered_records.Temperature);

DUMP max_temp;
```

(b) **Implement an EvalFunc UDF in Pig Latin to trim leading and trailing whitespace from char array values and explain the procedure to call the UDF in a Pig Latin script.**

(6 Marks)

Answer:

< 3 Marks >

Any user defined function (UDF) in Pig has to be implemented in Java and is to be packaged as a JAR file.

```
package util.pig.bec;

import org.apache.pig.PrimitiveEvalFunc;

public class Trim extends PrimitiveEvalFunc<String, String> {

   @Override
   public String exec(String input) {

      return input.trim();

   }

}
```

Assume the name of the JAR file created from the above program is "mypigutil.jar" and it is located in the directory whose path is "/home/udf/pig". The JAR file has to be registered with the Pig framework by using the following command.

```
REGISTER /home/udf/pig/mypigutil.jar;
```

< 3 Marks >

If A is a relation with schema `(fruit : chararray)` and contains the following tuples

```
(   pomegranate)
(banana  )
(apple)
(   lychee  )
```

The Trim function can be invoked on the tuples in relation A as follows

```
B = FOREACH A GENERATE util.pig.bec.Trim(fruit);
```

The relation B contains the following tuples.

```
(pomegranate)
(banana)
(apple)
(lychee)
```

OR

V (a) **How Pig supports join operations between relations?**

(6 Marks)

Answer:

< 2 Marks >

Joining datasets in MapReduce takes some work on the part of the programmer, whereas Pig has very good built-in support for join operations, making it much more approachable. Consider the relations A and B:

```
grunt> DUMP A;

(2,Tie)
(4,Coat)
(3,Hat)
(1,Scarf)

grunt> DUMP B;

(Joe,2)
(Hank,4)
(Ali,0)
(Eve,3)
(Hank,2)
```

We can perform INNERJOIN between the two relations on the numerical (identity) field in each as follows:

```
grunt> C = JOIN A BY $0, B BY $1;

grunt> DUMP C;

(2,Tie,Hank,2)
(2,Tie,Joe,2)
(3,Hat,Eve,3)
(4,Coat,Hank,4)
```

< 2 Marks >
Fragment replicate join:If one of the relations is small enough to fit in memory, a special type of join called a fragment replicate join, which is implemented by distributing the small input to all the mappers and performing a map-side join using an in-memory lookup table against the (fragmented) larger relation. The syntax to perform this join is

```
grunt> C = JOIN A BY $0, B BY $1 USING 'replicated';
```

The first relation must be the large one, followed by one or more small ones (all of which must fit in memory).

< 2 Marks >
Pig also supports outer joins using a syntax that is similar to SQL's. For example:

```
grunt> C = JOIN A BY $0 LEFT OUTER, B BY $1;
grunt> DUMP C;

(1,Scarf,,)
(2,Tie,Hank,2)
(2,Tie,Joe,2)
(3,Hat,Eve,3)
(4,Coat,Hank,4)
```

(b) **Write about COGROUP statement in Pig Latin.**

(6 Marks)

Answer:

< 2 Marks >
The COGROUP statement is similar to JOIN, but instead of creating a flat structure COGROUP creates a nested set of output tuples.

```
grunt> D = COGROUP A BY $0, B BY $1;
grunt> DUMP D;

(0,{},{(Ali,0)})
(1,{(1,Scarf)},{})
(2,{(2,Tie)},{(Hank,2),(Joe,2)})
(3,{(3,Hat)},{(Eve,3)})
(4,{(4,Coat)},{(Hank,4)})
```

COGROUP generates a tuple for each unique grouping key. The first field of each tuple is the key, and the remaining fields are bags of tuples from the relations with a matching key. The first bag contains the matching tuples from relation A with the same key. Similarly, the second bag contains the matching tuples from relation B with the same key. This is an example of an outer join, which is the default type for COGROUP. It can be made explicit using the OUTER keyword, making this COGROUP statement the same as the previous one:

```
D = COGROUP A BY $0 OUTER, B BY $1 OUTER;
```

< 2 Marks >
Rows with empty bags can be supressed by using the INNER keyword, which gives the COGROUP inner join semantics. The INNER keyword is applied per relation, so the following suppresses rows only when relation A has no match (dropping the unknown product 0 here):

```
grunt> E = COGROUP A BY $0 INNER, B BY $1;
grunt> DUMP E;

(1,{(1,Scarf)},{})
(2,{(2,Tie)},{(Hank,2),(Joe,2)})
(3,{(3,Hat)},{(Eve,3)})
(4,{(4,Coat)},{(Hank,4)})
```

< 2 Marks >
Using a combination of COGROUP, INNER, and FLATTEN (which removes nesting) it's possible to simulate an (inner) JOIN:

```
grunt> G = COGROUP A BY $0 INNER, B BY $1 INNER;
grunt> H = FOREACH G GENERATE FLATTEN($1), FLATTEN($2);
grunt> DUMP H;

(2,Tie,Hank,2)
(2,Tie,Joe,2)
(3,Hat,Eve,3)
(4,Coat,Hank,4)
```

This gives the same result as

```
JOIN A BY $0, B BY $1.
```

If the join key is composed of several fields, they should be specified in the BY clauses of the JOIN or COGROUP statement. The number of fields in each BY clause should be same.

UNIT - III

VI (a) **Describe the important properties that are used to configure Hive metastore.**

(6 Marks)

Answer:

< 1 Mark for each property >

Refer Table 1 for the important properties that are used to configure Hive metastore.

Table 1: Important metastore configuration properties

Property name	**Type**	**Default Value (optional value)**	**Description**
hive.metastore.warehouse.dir	URI	/user/hive/warehouse	The directory relative to fs.defaultFS where managed tables are stored
hive.metastore.uris	Comma separated URIs	Not set (thrift://host:port)	If not set (the default), use an inprocess metastore; otherwise, connect to one or more remote metastores, specified by a list of URIs. Clients connect in a roundrobin fashion when there are multiple remote servers.
javax.jdo.option. ConnectionURL	URI	jdbc:derby:;databaseName= metastore_db; create=true (jdbc:mysql://host/dbname? createDatabaseIfNotExist=true)	The JDBC URL of the metastore database.
javax.jdo.option. ConnectionDriverName	String	org.apache.derby. jdbc.EmbeddedDriver (com.mysql.jdbc.Driver)	The JDBC driver classname.
javax.jdo.option. ConnectionUserName	String	user_name for dbms server	The JDBC username.
javax.jdo.option. ConnectionPassword	String	password to access dbms server	The JDBC password.

(b) **Compare different metastore configurations that are possible in Hive.**

(6 Marks)

Answer:

< 2 Marks for each configuration >

The metastore is the central repository of Hive metadata. The metastore is divided into two pieces: a service and the backing store for the data.

- **Embedded metastore configuration** : the metastore service runs in the same JVM as the Hive service and contains an embedded Derby database instance backed by the local disk. Only one connection can be established to the metastore as a result only one Hive session is possible.
- **Local metastore configuration** : the metastore service still runs in the same process as the Hive service but connects to a database running in a separate process, either on the same machine or on a remote machine. Any JDBC-compliant database may be used by setting the javax.jdo.option.* configuration properties listed in Table 1.
- **Remote metastore configuration** : One or more metastore servers run in separate processes on a remote machine to the Hive service. This brings better manageability and security because the database tier can be completely firewalled off, and the clients no longer need the database credentials.

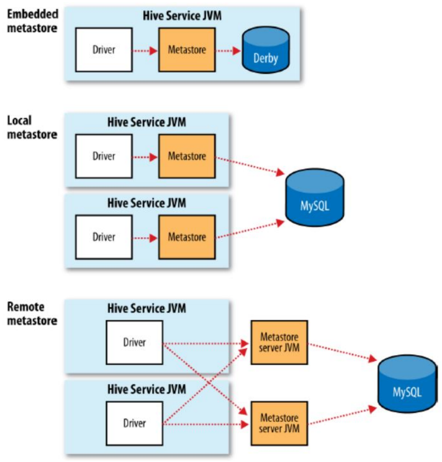


Figure 2: Metastore configurations

OR

VII (a) **How table partitioning improves Hive query performance?**

(6 Marks)

Answer:

< 2 Marks >

The concept of partitions can be explained by taking logfiles as example where each record includes a timestamp. If the data is partition by date, then records for the same date will be stored in the same partition. The advantage to this scheme is that queries that are restricted to a particular date or set of dates can run much more efficiently, because they only need to scan the files in the partitions that the query pertains to. Partitioning doesn't preclude more wide-ranging queries: it is still feasible to query the entire dataset across many partitions.

A table may be partitioned in multiple dimensions. For example, in addition to partitioning logs by date, it is also possible to subpartition each date partition by country to permit efficient queries by location. Partitions are defined at table creation time using the **PARTITIONED BY** clause, which takes a list of column definitions.

< 2 Marks >

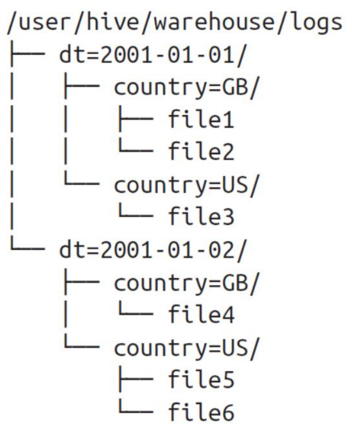
```
CREATE TABLE logs (ts BIGINT, line STRING)
PARTITIONED BY (dt STRING, country STRING);
```

When we load data into a partitioned table, the partition values are specified explicitly:

```
LOAD DATA LOCAL INPATH 'input/hive/partitions/file1'
INTO TABLE logs
PARTITION (dt='2001-01-01', country='GB');
```

< 2 Marks >

At the filesystem level, partitions are simply nested sub directories of the table directory. After loading a few more files into the logs table, the directory structure is shown in Fig.3.



```
/user/hive/warehouse/logs
├── dt=2001-01-01/
│   ├── country=GB/
│   │   ├── file1
│   │   └── file2
│   └── country=US/
│       └── file3
└── dt=2001-01-02/
    ├── country=GB/
    │   └── file4
    └── country=US/
        ├── file5
        └── file6
```

Figure 3: Directory structure for **logs** table when partitioned by columns **dt** and **country**

To know the partitions of a table the following command can be used `SHOW PARTITIONS logs;`.

(b) **Write about Hive architecture.**

(6 Marks)

Answer:
Hive framework comprises of Hive services and Hive clients.

< 2 Marks >
Hive Services To use a specific Hive service one can use the command `hive --service service_name`. The list of available Hive service names can be obtained by using the same command with out specifying the service_name `hive --service`.

- CLI : this is the command-line interface to Hive (the shell). This is the default service.
- Hiveserver2 : runs Hive as a server exposing a Thrift service, enabling access from a range of clients written in different languages.
- Beeline : a command-line interface to Hive that works in embedded mode (like the regular CLI), or by connecting to a HiveServer 2 process using JDBC.
- hwi : the Hive Web Interface. A simple web interface that can be used as an alternative to the CLI without having to install any client software.
- Metastore : By default, the metastore is run in the same process as the Hive service. Using this service, it is possible to run the metastore as a standalone (remote) process. The **METASTORE_PORT** environment variable (or the -p command-line option) is used to specify the port the server will listen on (default port at which metastore service is listening is 9083).
- Execution engine : Hive was originally written to use MapReduce as its execution engine, and that is still the default. It is also possible to run Hive using Apache Tez or Spark as its execution engine too. Both Tez and Spark engines offer more flexibility and higher performance than MapReduce.

< 2 Marks >
Hive clients There are a number of different mechanisms for connecting to Hive server from applications. The relationship between Hive clients and Hive services is illustrated in Fig. 4.

- Thrift Client : Any programming language that supports Thrift can use the Thrift client to interact with it the Hive server (Thrift service).
- JDBC driver : When configured with a JDBC URI of the form `jdbc:hive2://host:port/dbname`, a Java application can connect to a Hive server running in a separate process at the given host and port. Hive provides a Type 4 (pure Java) JDBC driver, defined in the class `org.apache.hadoop.hive.jdbc.HiveDriver`.
- ODBC driver : An ODBC driver allows applications that support the ODBC protocol (such as business intelligence software) to connect to Hive.

< 2 Marks >

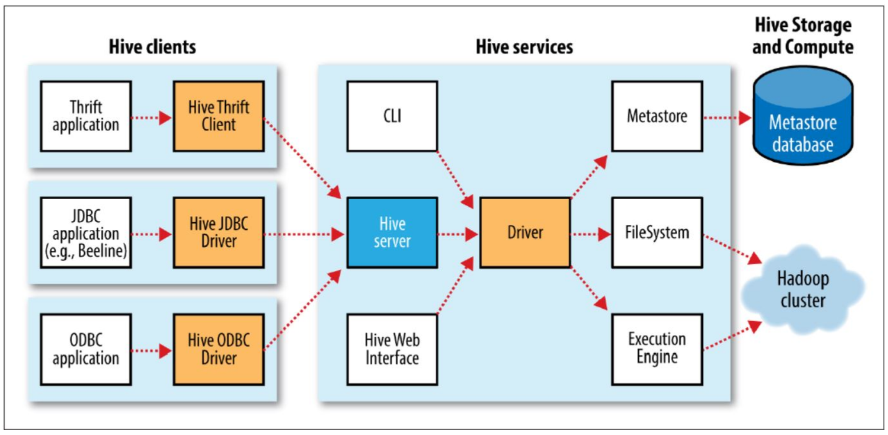


Figure 4: Hive Architecture

UNIT - IV

VIII (a) **How Spark runs a job?**

(6 Marks)

Answer:

< 3 Marks >

When a Spark job is executed at the highest level, there are two independent entities: the driver, which hosts the application (SparkContext) and schedules tasks for a job; and the executors, which are exclusive to the application, run for the duration of the application, and execute the application's tasks.

Figure 5 illustrates how Spark runs a job. A 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 in Figure 19-1), 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. 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. Assigned tasks are launched through a scheduler backend (step 4 in Fig. 5), which sends a remote launch task message (step 5) to the executor backend to tell the executor to run the task (step 6). An executor runs the task code (step 7).

< 3 Marks >

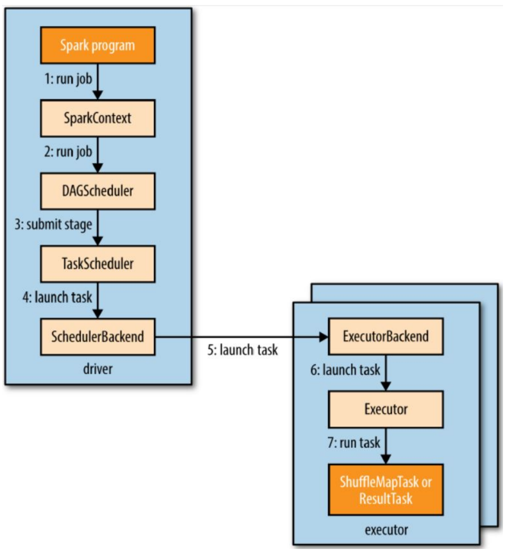


Figure 5: Spark job work flow

(b) **Describe the interaction between Spark and Yarn cluster mode.**

(6 Marks)

Answer:

< 3 Marks >

In YARN cluster mode, the user's driver program runs in a YARN application master process. The spark-submit command is used with a master URL of yarn-cluster:

```
spark-submit --master yarn-cluster ...
```

The spark-submit client will launch the YARN application (step 1 in Fig. 6), but it doesn't run any user code. The rest of the process is the same as client mode, except the application master starts the driver program (step 3b) before allocating resources for executors (step 4).

< 3 Marks >

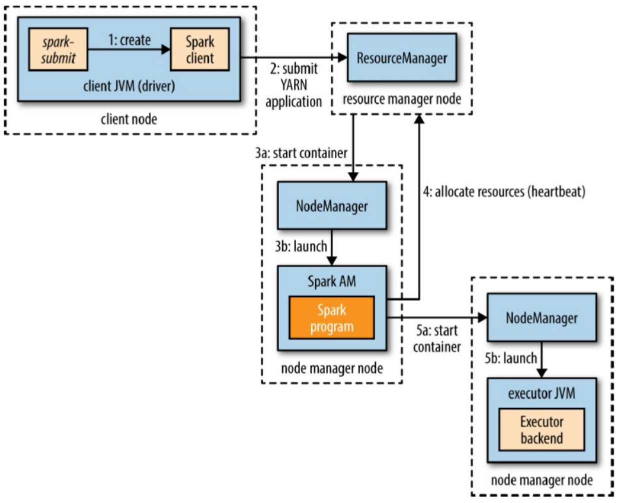


Figure 6: Spark executors invocation in YARN cluster mode

OR

IX (a) **Describe Sqoop import command.**

(6 Marks)

Answer:

< 2 Marks >

Sqoop's import tool is used to import data from a structured data store such as Relational database into Hadoop for further processing. The usage of Sqoop's import tool is as follows:

```
sqoop import [GENERIC-ARGS] [TOOL-ARGS]
```

< Explanation for any 4 arguments >

Common arguments:

`--connect <jdbc-uri>`	Specify JDBC connect string
`--driver <class-name>`	Manually specify JDBC driver class to use
`--hadoop-home <dir>`	Override $HADOOP_HOME
`--help`	Print usage instructions
`-P`	Read password from console
`--password <password>`	Set authentication password
`--username <username>`	Set authentication username
`--verbose`	Print more information while working
`--table`	Name of the table to be imported
`--columns`	Columns names in the table to be imported
`--where`	Boolean expression defined on one or more columns of the table to specify the tuples to be imported from the table.
`-m`	Number of map tasks to be used for import

< 2 Marks >

Assume a table with following schema

```
student(regno CHAR(9)PRIMARY KEY, name VARCHAR(60), dob DATE, cgpa DECIMAL(4,2))
```

is available in a database by name "demo" of mysql server running in a local system. The Sqoop's import command is:

```
sqoop import --connect jdbc:mysql://localhost/demo --username root -P --table student
--columns "regno,cgpa" -m 1
```

For the above command to work a JDBC driver JAR file for MySQL (Connector/J) is to be added to Sqoop's classpath, which is simply achieved by placing it in Sqoop's lib directory.

< 2 Marks >

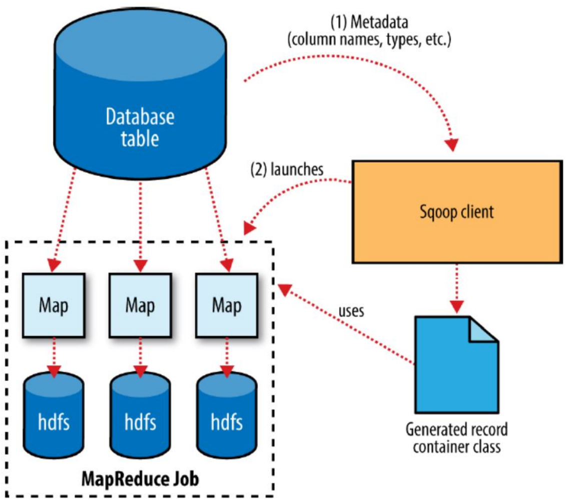


Figure 7: Sqoop's import process

(b) **How Sqoop export command works?**

(6 Marks)

Answer:

< 3 Marks >

When the final results of an analytic pipeline are available, Sqoop's export tool can be used to export these results back to a data store such as Relational database for consumption by other clients. Before exporting a table from HDFS to a database, a table that has target columns in the same order, with the appropriate SQL types is to be created in the destination database.
Assume a table is created with the following schema

```
student(regno CHAR(9)PRIMARY KEY, name VARCHAR(60), dob DATE, cgpa DECIMAL(4,2))
```

in a database by name "demo" of MySQL server running in a local system. The Sqoop's export command to export table by name "source" in Hive's data warehouse to a "student" table in "demo" database of MySql server running on local machine is:

```
sqoop export --connect jdbc:mysql://localhost/demo --username root -P --table student
--export-dir /user/hive/warehouse/source --input-fields-terminated-by ',' -m 1
```

< 3 Marks >

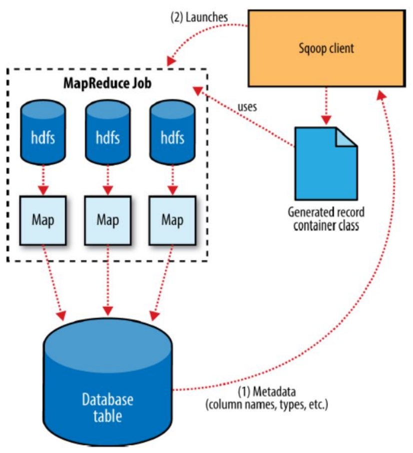


Figure 8: Sqoop's export process

Scheme prepared by

(N.Sivaram Prasad, IT Dept.)

Signature of HOD

Signatures of evaluators

SNo	Name of the evaluator	College Name	Signature