USN | | | | | | | | | | |

Scheme of Evaluation with solutions
Internal Assessment Test 3 – JULY 2021

Sub:	**Big Data Analytics**				Sub Code:	17CS82	Branch:	ISE		
Date:	18/07/2021	Duration:	90 min's	Max Marks:	50	Sem / Sec:	VIII A,B		OBE	
Answer any FIVE FULL Questions								MARKS	CO	RBT

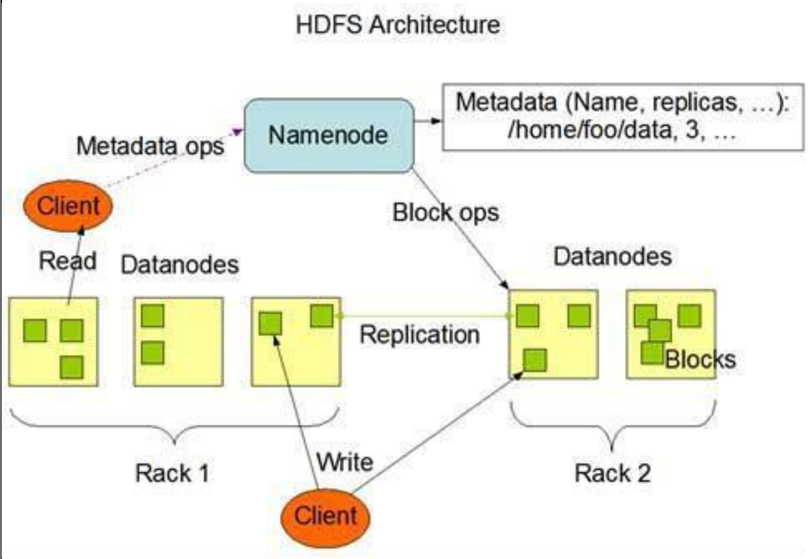
1	**What is HDFS? List and explain all components of HDFS. Write five HDFS commands.**	[10M]	CO1	L2

2M+5M+3M

The Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size. We describe the architecture of HDFS and report on experience using HDFS to manage 40 petabytes of enterprise data at Yahoo.

HDFS COMPONENTS

The design of HDFS is based on two types of nodes: a NameNode and multiple DataNodes. In a basic design, a single NameNode manages all the metadata needed to store and retrieve the actual data from the DataNodes.



Data Replication

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. Blocks belonging to a file are replicated for fault tolerance. The block size and replication factor are configurable per file. Files in HDFS are write-once and have strictly one writer at any time. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. The Namenode makes all decisions regarding replication of blocks. It periodically receives Heartbeat and a Blockreport from each of the Datanodes in the cluster.

SafeMode

On startup, the Namenode enters a special state called Safemode. Replication of data blocks does not occur when the Namenode is in Safemode state. The

Namenode receives Heartbeat and Blockreport from the Datanodes. A Blockreport contains the list of data blocks that a Datanode reports to the Namenode. Each block has a specified minimum number of replicas. A block is considered safely-replicated when the minimum number of replicas of that data block has checked in with the Namenode. When a configurable percentage of safely-replicated data blocks checks in with the Namenode (plus an additional 30 seconds), the Namenode exits the Safemode state.

Snapshots

Snapshots support storing a copy of data at a particular instant of time. One usage of the snapshot-feature may be to roll back a corrupted cluster to a previously known good point in time. HDFS current does not support snapshots but it will be supported it in future release.

Staging

A client-request to create a file does not reach the Namenode immediately. In fact, the HDFS client caches the file data into a temporary local file. An application-write is transparently redirected to this temporary local file. When the local file accumulates data worth over a HDFS block size, the client contacts the Namenode. The Namenode inserts the file name into the file system hierarchy and allocates a data block for it. The Namenode responds to the client request with the identity of the Datanode(s) and the destination data block. The client flushes the block of data from the local temporary file to the specified Datanode. When a file is closed, the remaining un-flushed data in the temporary local file is transferred to the Datanode. The client then instructs the Namenode that the file is closed. At this point, the Namenode commits the file creation operation into a persistent store.

Pipelining

When a client is writing data to a HDFS file, its data is first written to a local file as explained above. Suppose the HDFS file has a replication factor of three. When the local file accumulates a block of user data, the client retrieves a list of Datanodes from the Namenode. This list represents the Datanodes that will host a replica of that block. The client then flushes the data block to the first Datanode. The first Datanode starts receiving the data in small portions (4 KB), writes each portion to its local repository and transfers that portion to the second Datanode in the list.

1. version Command Name: version Command Usage: version Example: hadoop version Description: Shows the version of hadoop installed. 2. mkdir Command Name: mkdir Command Usage: mkdir Example: 1. hdfs dfs -mkdir /user/dataflair/dir1 Description: This command takes the as an argument and creates the directory. 3. Is Command Name: ls Command Usage: ls Example: 1. hdfs dfs -ls /user/dataflair Description: This command displays the contents of the directory specified by . It shows the name, permissions, owner, size and modification date of each entry. Second Example: 1. hdfs dfs -ls -R /user Description: This command behaves like ls but displays entries in all the sub-directories recursively 4. put Command Name: put Command Usage: put Example: 1. hdfs dfs -put /home/sample.txt /user/dataflair/dir1 Description: This command copies the file in the local filesystem to the file in DFS. 5. copyFrom Local Command Name: copyFrom Local Command Usage: copyFrom Local Example: 1. hdfs dfs -copyFromLocal /home/sample /user/dataflair/dir1 Description: This command is similar to put command. But the source should refer to local file. 6. get Command Name:get Command Usage: get Example: 1. hdfs dfs -get /user/dataflair/dir1 /home Description: This Hadoop shell command copies the file in HDFS identified by to file in local file system identified by Second Example: 1. hdfs dfs -getmerge /user/dataflair/dir1/sample.txt /user/dataflair/dir2/sample2.txt /home/sample1.txt Description: This HDFS

	command retrieves all files in the source path entered by the user in HDFS. And merges them into one single file created in the local file system identified by local destination. 1. **version Command Name:** version Command Usage: version Example: hadoop version Description: Shows the version of hadoop installed. 2. mkdir Command Name: mkdir Command Usage: mkdir Example: 1. hdfs dfs -mkdir /user/dataflair/dir1 Description: This command takes the as an argument and creates the directory. 3. Is Command Name: ls Command Usage: ls Example: 1. hdfs dfs -ls /user/dataflair Description: This command displays the contents of the directory specified by . It shows the name, permissions, owner, size and modification date of each entry. Second Example: 1. hdfs dfs -ls -R /user Description: This command behaves like ls but displays entries in all the sub-directories recursively 4. put Command Name: put Command Usage: put Example: 1. hdfs dfs -put /home/sample.txt /user/dataflair/dir1 Description: This command copies the file in the local filesystem to the file in DFS. 5. copyFrom Local Command Name: copyFrom Local Command Usage: copyFrom Local Example: 1. hdfs dfs -copyFromLocal /home/sample /user/dataflair/dir1 Description: This command is similar to put command. But the source should refer to local file. 6. get Command Name:get Command Usage: get Example: 1. hdfs dfs -get /user/dataflair/dir1 /home Description: This Hadoop shell command copies the file in HDFS identified by to file in local file system identified by Second Example: 1. hdfs dfs -getmerge /user/dataflair/dir1/sample.txt /user/dataflair/dir2/sample2.txt /home/sample1.txt Description: This HDFS command retrieves all files in the source path entered by the user in HDFS. And merges them into one single file created in the local file system identified by local destination.			
2	**Analyze Hadoop tools considering large data set of airline system. List and explain any five essential Hadoop tools with their features. Write Hadoop Installation steps.** **2M+5M+3M** **Airline system:** The tools used for the proposed method is Hadoop and Hive which is mainly used for structured data. Assuming all the Hadoop tools have been installed and having semi-structured information on airport data. The methodology used is as follows: • Our project is focused on the extraction of data and analysis of Airlines data. • Predicting flight delay • Busiest Routes • Month delay of flights. • Yearly delays of flights • Find the list of active airlines in the country • The big issue to manage or to handle such an anonymous or huge data. **Hadoop is an open source distributed processing framework which is at the center of a growing big data ecosystem.** Used to support advanced analytics initiatives, including predictive analytics, data mining and machine learning applications, Hadoop manages data processing and storage for big data applications and can handle various forms of structured and unstructured data. 1. **The Hadoop Distributed File System (HDFS) is** designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size. We describe the architecture of HDFS and report on experience using HDFS to manage 40 petabytes of enterprise data at Yahoo. Features: a. Rack awareness allows consideration of a node's physical location,	[10M]	CO1	L3

when allocating storage and scheduling tasks
b. Minimal data motion. MapReduce moves compute processes to the data on HDFS and not the other way around. Processing tasks can occur on the physical node where the data resides. This significantly reduces the network I/O patterns and keeps most of the I/O on the local disk or within the same rack and provides very high aggregate read/write bandwidth.
c. Utilities diagnose the health of the files system and can rebalance the data on different nodes
d. Rollback allows system operators to bring back the previous version of HDFS after an upgrade, in case of human or system errors e. Standby NameNode provides redundancy and supports high availability f. Highly operable. Hadoop handles different types of cluster that might otherwise require operator intervention. This design allows a single operator to maintain a cluster of 1000s of nodes.
2. Hbase HBase is a column-oriented database management system that runs on top of HDFS. It is well suited for sparse data sets, which are common in many big data use cases. Unlike relational database systems, HBase does not support a structured query language like SQL; in fact, HBase isn't a relational data store at all. HBase applications are written in Java much like a typical MapReduce application. HBase does support writing applications in Avro, REST, and Thrift. Features: a. Linear and modular scalability. b. Strictly consistent reads and writes. c. Automatic and configurable sharding of tables d. Automatic failover support between Region Servers. e. Convenient base classes for backing Hadoop MapReduce jobs with Apache HBase tables. f. Easy to use Java API for client access. g. Block cache and Bloom Filters for real-time queries. h. Query predicate push down via server side Filters
3. HIVE The Apache Hive data warehouse software facilitates querying and managing large datasets residing in distributed storage. Hive provides a mechanism to project structure onto this data and query the data using a SQL-like language called HiveQL. At the same time this language also allows traditional map/reduce programmers to plug in their custom mappers and reducers when it is inconvenient or inefficient to express this logic in HiveQL.Support for exporting metrics via the Hadoop metrics subsystem to files or Ganglia; or via JMX. Features: a. Indexing to provide acceleration, index type including compaction and Bitmap index as of 0.10, more index types are planned. b. Different storage types such as plain text, RCFile, HBase, ORC, and others. c. Metadata storage in an RDBMS, significantly reducing the time to perform semantic checks during query execution. d. Operating on compressed data stored into Hadoop ecosystem, algorithm including gzip, bzip2, snappy, etc. e. Built-in user defined functions (UDFs) to manipulate dates, strings, and other data-mining tools. Hive supports extending the UDF set to handle use-cases not supported by built-in functions. f. SQL-like queries (Hive QL), which are implicitly converted into map-reduce jobs.
4. Sqoop is a tool designed to transfer data between Hadoop and relational databases. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle into the Hadoop Distributed File System (HDFS), transform the data in Hadoop MapReduce, and then export the data back into an RDBMS. Features: a. Connecting to database server b. Controlling parallelism c. Controlling the import process d. Import data to hive e. Import data to Hbase
5. Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets. At the present time, Pig's infrastructure layer consists

of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist (e.g., the Hadoop subproject). Pig's language layer currently consists of a textual language called Pig Latin Features: a. Ease of programming. b. It is trivial to achieve parallel execution of simple, “embarrassingly parallel” data analysis tasks. Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain. c. Optimization opportunities. d. The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency. e. Extensibility. Users can create their own functions to do special-purpose processing

Installation stepsPart 1

Step 1:

• sudo addgroup hadoop_

• sudo adduser –ingroup Hadoop_ hduser_

• Enter your password, name and other details.

• NOTE: There is a possibility of below-mentioned error in this setup and installation
process.

• "hduser is not in the sudoers file. This incident will be reported."

• This error can be resolved by Login as a root user

• Execute the command

• Sudo adduser hduser_ sudo

• Re-login as hduser

Step 2) Configure SSH

In order to manage nodes in a cluster, Hadoop requires SSH access

• First, switch user, enter the following command

su -hduser_

• This command will create a new key.

ssh-keygen -t rsa -P “”

• Enable SSH access to local machine using this key.

cat $HOME/.ssh/id_rsa-pub >>$HOME/.ssh/authorized_keys

• Now test SSH setup by connecting to localhost as 'hduser' user.

ssh localhost

• Note: Please note, if you see below error in response to 'ssh localhost', then there is a
possibility
that SSH is not available on this system-

• To resolve this - Purge SSH using, sudo apt-get purge openssh -server

It is good practice to purge before the start of installation

• Install SSH using the command- sudo apt-get install openssh-server

Step 3) Next step is to Download Hadoop

Select Stable

Select the tar.gz file (not the file with src)

• Once a download is complete, navigate to the directory containing the tar file

Enter, sudo tar xzf Hadoop-2.2.0.tar/gz

• Now, rename hadoop-2.2.0 as hadoop

Sudo mv Hadoop-2.2.0 hadoop

Sudo chown -R hduser_:hadoop_ Hadoop

Part 2) Configure Hadoop

Step 1) Modify ~/.bashrc file

• Add following lines to end of file ~/.bashrc

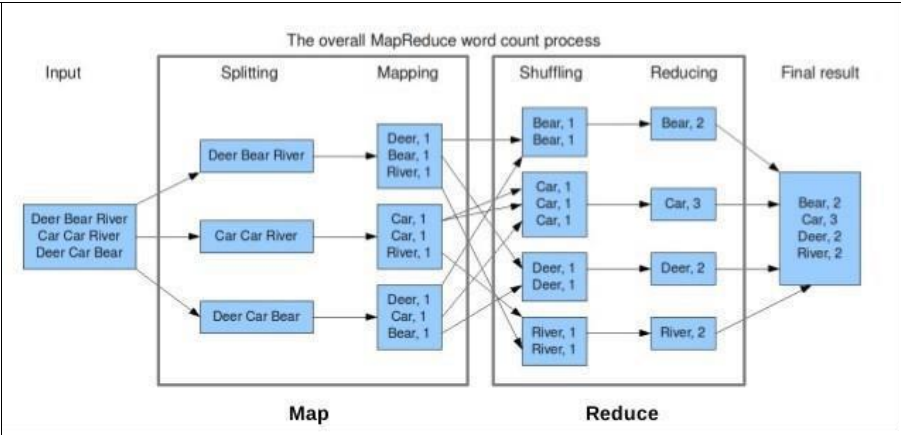
• #Set HADOOP_HOME

• export HADOOP_HOME=<Installation Directory of Hadoop>
• #Set JAVA_HOME
• export JAVA_HOME=<Installation Directory of Java>
• # Add bin/ directory of Hadoop to PATH
• export PATH=$PATH:$HADOOP_HOME/bin
• Now, source this environment configuration using below command
. ~/.bashrc
Step 2) Configurations related to HDFS
Set JAVA_HOME inside file $HADOOP_HOME/etc/hadoop/hadoop-env.sh
There are two parameters in $HADOOP_HOME/etc/hadoop/core-site.xml which need to beset 1. 'hadoop.tmp.dir' - Used to specify a directory which will be used by Hadoop to store its
data
files.
2. 'fs.default.name' - This specifies the default file system.
To set these parameters, open core-site.xml
sudo gedit $HADOOP_HOME/etc/hadoop/core-site.xml
Copy below line in between tags <configuration></configuration>

```
<property>
<name>hadoop.tmp.dir</name>
<value>/app/hadoop/tmp</value>
<description>Parent directory for other temporary directories.</description>
</property>
<property>
<name>fs.defaultFS </name>
<value>hdfs://localhost:54310</value>
<description>The name of the default file system. </description>
</property>
```

• Navigate to the directory $HADOOP_HOME/etc/Hadoop
•
Now, create the directory mentioned in core-site.xml
sudo mkdir -p <Path of Directory used in above setting>
•
Grant permissions to the directory
sudo chown -R hduser_:Hadoop_ <Path of Directory created in above step>
sudo chmod 750 <Path of Directory created in above step>
Step 3) Map Reduce Configuration
• Before you begin with these configurations, lets set HADOOP_HOME path
• sudo gedit /etc/profile.d/hadoop.sh
•
And Enter
export HADOOP_HOME=/home/guru99/Downloads/Hadoop
•
Next enter
sudo chmod +x
/etc/profile.d/hadoop.sh
•
Exit the Terminal and restart again
Type echo $HADOOP_HOME. To verify the path
• Now copy files
sudo cp $HADOOP_HOME/etc/hadoop/mapred-site.xml.template
$HADOOP_HOME/etc/hadoop/mapred-site.xml
•

	Open the mapred-site.xml file sudo gedit $HADOOP_HOME/etc/hadoop/mapred-site.xml • Add below lines of setting in between tags <configuration> and </configuration> <property> <name>mapreduce.jobtracker.address</name> <value>localhost:54311</value> <description>MapReduce job tracker runs at this host and port.</description> </property> • Open $HADOOP_HOME/etc/hadoop/hdfs-site.xml as below, sudo gedit $HADOOP_HOME/etc/hadoop/hdfs-site.xml • Add below lines of setting between tags <configuration> and </configuration> <property> <name>dfs.replication</name> <value>1</value> <description>Default block replication.</description> </property> <property> <name>dfs.datanode.data.dir</name> <value>/home/hduser_/hdfs</value> </property> • Create a directory specified in above setting sudo mkdir -p <Path of Directory used in above setting> sudo mkdir -p /home/hduser_/hdfs sudo chown -R hduser_:hadoop_ <Path of Directory created in above step> sudo chown -R hduser_:hadoop_ /home/hduser_/hdfs sudo chmod 750 <Path of Directory created in above step> sudo chmod 750 /home/hduser_/hdfs Step 4) • Before we start Hadoop for the first time, format HDFS using below command $HADOOP_HOME/bin/hdfs namenode -format Step 5) • Start Hadoop single node cluster using below command $HADOOP_HOME/sbin/start-dfs.sh $HADOOP_HOME/sbin/start-yarn.sh • Using 'jps' tool/command, verify whether all the Hadoop related processes are running or not. If Hadoop has started successfully then an output of jps should show NameNode, NodeManager, ResourceManager, SecondaryNameNode, DataNode. Step 6) Stopping Hadoop $HADOOP_HOME/sbin/stop-dfs.sh $HADOOP_HOME/sbin/stop-yarn.s			
3 (a)	**Create a MapReduce program for Pi Function. Explain with neat diagram MapReduce data flow.** **2M +3M**	[5M]	CO1	L3



Hadoop MapReduce is a programming paradigm at the heart of Apache Hadoop for providing massive scalability across hundreds or thousands of Hadoop clusters on commodity hardware. The MapReduce model processes large unstructured data sets with a distributed algorithm on a Hadoop cluster.

The term MapReduce represents two separate and distinct tasks Hadoop programs perform-Map Job and Reduce Job. Map job scales takes data sets as input and processes them to produce key value pairs. Reduce job takes the output of the Map job i.e. the key value pairs and aggregates them to produce desired results. The input and output of the map and reduce jobs are stored in HDFS.

The following word count example explains MapReduce method. For simplicity, let's consider a few words of a text document. We want to find the number of occurrence of each word. First the input is split to distribute the work among all the map nodes as shown in the figure. Then each word is identified and mapped to the number one. Thus the pairs also called as tuples are created. In the first mapper node three words Deer, Bear and River are passed. Thus the output of the node will be three key, value pairs with three distinct keys and value set to one. The mapping process remains the same in all the nodes. These tuples are then passed to the reduce nodes. A partitioner comes into action which carries out shuffling so that all the tuples with same key are sent to same node.

(b)	**Develop java code for MAP and REDUCE of word count Problem.**	[5M]	CO1	L3

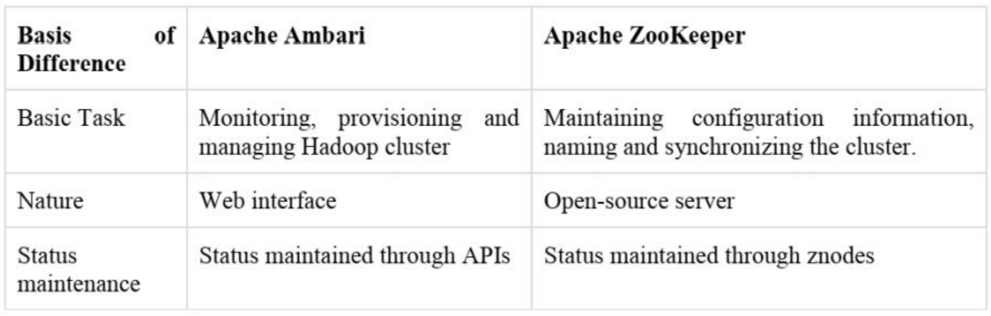
```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
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.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordCount {
  // Map function
  public static class MyMapper extends Mapper<LongWritable, Text, Text,
IntWritable>{
    private Text word = new Text();
    public void map(LongWritable key, Text value, Context context)
          throws IOException, InterruptedException {
      // Splitting the line on spaces
      String[] stringArr = value.toString().split("\\s+");
      for (String str : stringArr) {
        word.set(str);
        context.write(word, new IntWritable(1));
      }
    }
  }

  // Reduce function
  public static class MyReducer extends Reducer<Text, IntWritable, Text,
IntWritable>{
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
            throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
  public static void main(String[] args)  throws Exception{
    Configuration conf = new Configuration();

    Job job = Job.getInstance(conf, "WC");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(MyMapper.class);
    job.setReducerClass(MyReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

4 (a)	**Differentiate Apache Ambari and Zookeeper? Explain Oozie DAG workflow.** **3M+3M** This description may confuse you as Zookeeper performs the similar kind of tasks. But, there is a huge difference between the tasks performed by these two technologies if looked closely.	[6M]	CO2	L2



Basis of Difference	Apache Ambari	Apache ZooKeeper
Basic Task	Monitoring, provisioning and managing Hadoop cluster	Maintaining configuration information, naming and synchronizing the cluster.
Nature	Web interface	Open-source server
Status maintenance	Status maintained through APIs	Status maintained through znodes

Oozie is a server based *Workflow Engine* specialized in running workflow jobs with actions that run Hadoop Map/Reduce and Pig jobs.Oozie is a Java Web-Application that runs in a Java servlet-container.For the purposes of Oozie, a workflow is a collection of actions (i.e. Hadoop Map/Reduce jobs, Pig jobs) arranged in a control dependency DAG (Direct Acyclic Graph). "control dependency" from one action to another means that the second action can't run until the first action has completed.Oozie workflows definitions are written in hPDL (a XML Process Definition Language similar to JBOSS JBPM jPDL).Oozie workflow actions start jobs in remote systems (i.e. Hadoop, Pig). Upon action completion, the remote systems callback Oozie to notify the action completion, at this point Oozie proceeds to the next action in the workflow.Oozie workflows contain control flow nodes and action nodes.Control flow nodes define the beginning and the end of a workflow (start , end and fail nodes) and provide a mechanism to control the workflow execution path (decision , fork and join nodes).

(b)	**Discuss Different Views Supported by Apache Ambari.** **4*1=4M**	[4M]	CO2	L2

A Framework that enables developers to create UI components, or Views, that "plug into" the Ambari Web interface is what we call Apache Ambari Views. However, automatically Ambari creates and presents some instances of Views to users when the service used by that View is added to the cluster.
As an example, the YARN Queue Manager View displays to Ambari web users, if **Apache YARN** service is added to the cluster. Although, the Ambari Admin user must manually create a view instance, in other cases.
In order to extend and customize the Ambari web, views enable us and also they help us to meet our specific needs.

The following Ambari views currently available to you:

Yarn Queue Manager View
Provides a visual way to configure YARN capacity scheduler queue capacity.

Files View
Allows you to browse the HDFS file system.

SmartSense View
Allows you to capture bundles, set bundle capture schedule, and view and download captured bundles.

Workflow Manager View
Allows you to easily create and schedule workflows and monitor workflow jobs.

5	**What is the significance of Apache Pig and Hive in Hadoop context? Describe main components and working with a simple example.**	[10M]	CO2	L2

2M+6M+2M

Apache Pig is a high-level language that enables programmers to write complex MapReduce transformations using a simple scripting language. Pig Latin (the actual language) defines a set of transformations on a data set such as aggregate, join, and sort. Pig is often used to extract, transform, and load (ETL) data pipelines, quick research on raw data, and iterative data processing. Apache Pig has several usage modes. The first is a local mode in which all processing is done on the local machine. The non-local (cluster) modes are MapReduce and Tez. These modes execute the job on the cluster using either the MapReduce engine or the optimized Tez engine. (Tez, which is Hindi for "speed," optimizes multistep Hadoop jobs such as those found in many Pig queries.) There are also interactive and batch modes available; they enable Pig applications to be developed locally in interactive modes, using small amounts of data, and then run at scale on the cluster in a production mode.

Pig Example Walk-Through

For this example, the following software environment is assumed. Other environments should work in a similar fashion.

OS: Linux

Platform: RHEL 6.6

Hortonworks HDP 2.2 with Hadoop version: 2.6

Pig version: 0.14.0

If you are using the pseudo-distributed installation from Chapter 2, "Installation Recipes," instructions for installing Pig are provided in that chapter. More information on installing Pig by hand can be found on the Pig website: http://pig.apache.org/#Getting+Started. Apache Pig is also installed as part of the Hortonworks HDP Sandbox.

In this simple example, Pig is used to extract user names from the /etc/passwd file. A full description of the Pig Latin language is beyond the scope of this introduction, but more information about Pig can be found at http://pig.apache.org/docs/r0.14.0/start.html. The following example assumes the user is hdfs, but any valid user with access to HDFS can run the example.

To begin the example, copy the passwd file to a working directory for local Pig operation:

```
$ cp /etc/passwd .
```

Next, copy the data file into HDFS for Hadoop MapReduce operation:

```
$ hdfs dfs -put passwd passwd
```

You can confirm the file is in HDFS by entering the following command:

Click here to view code image

```
hdfs dfs -ls passwd
-rw-r--r-- 2 hdfs hdfs 2526 2015-03-17 11:08 passwd
```

In the following example of local Pig operation, all processing is done on the local machine (Hadoop is not used). First, the interactive command line is started:

```
$ pig -x local
```

If Pig starts correctly, you will see a grunt> prompt. You may also see a bunch of INFO messages, which you can ignore. Next, enter the following commands to load the passwd file and then grab the user name and dump it to the terminal. Note that Pig commands must end with a semicolon (;).

Click here to view code image

```
grunt> A = load 'passwd' using PigStorage(':');
grunt> B = foreach A generate $0 as id;
grunt> dump B;
```

The processing will start and a list of user names will be printed to the screen. To exit the interactive session, enter the command quit.

```
$ grunt> quit
```

To use Hadoop MapReduce, start Pig as follows (or just enter pig):

```
$ pig -x mapreduce
```

The same sequence of commands can be entered at the grunt> prompt. You may wish to change the $0 argument to pull out other items in the passwd file. In the case of this simple script, you will notice that the MapReduce version takes much longer. Also, because we are running this application under Hadoop, make sure the file is placed in HDFS.

If you are using the Hortonworks HDP distribution with tez installed, the tez engine can be used as follows:

```
$ pig -x tez
```

Pig can also be run from a script. An example script (id.pig) is available from the example code download (see Appendix A, "Book Webpage and Code Download"). This script, which is repeated here, is designed to do the same things as the interactive version:

Click here to view code image

```
/* id.pig */
A = load 'passwd' using PigStorage(':'); -- load the passwd fileB = foreach A
generate $0 as id; -- extract the user IDs
dump B;
store B into 'id.out'; -- write the results to a directory name id.out
```

Comments are delineated by /* */ and -- at the end of a line. The script will create a directory called id.out for the results. First, ensure that the id.out directory is not in your local directory, and then start Pig with the script on the command line:

```
$ /bin/rm -r id.out/
$ pig -x local id.pig
```

Apache Hive is a data warehouse infrastructure built on top of Hadoop for providing data summarization, ad hoc queries, and the analysis of large data sets using a SQL-like language called HiveQL. Hive is considered the de facto standard for interactive SQL queries over petabytes of data using Hadoop and offers the following features: Tools to enable easy data extraction, transformation, and loading (ETL) A mechanism to impose structure on a variety of data formats Access to files stored either directly in HDFS or in other data storage systems such as HBase Query execution via MapReduce and Tez (optimized MapReduce) Hive provides users who are already familiar with SQL the capability to query the data on Hadoop clusters. At the same time, Hive makes it possible for programmers who are familiar with the MapReduce framework to add their custom mappers and reducers to Hive queries. Hive queries can also be dramatically accelerated using the Apache Tez framework under YARN in Hadoop version 2

Hive Example Walk-Through

For this example, the following software environment is assumed. Other environments should work in a similar fashion.

OS: Linux

Platform: RHEL 6.6

Hortonworks HDP 2.2 with Hadoop version: 2.6

Hive version: 0.14.0

If you are using the pseudo-distributed installation from Chapter 2, instructions for installing Hive are provided in that chapter. More information on installation can be found on the Hive website: http://hive.apache.org. Hive is also installed as part of the Hortonworks HDP Sandbox. Although the following example assumes the user is hdfs, any valid user with access to HDFS can run the example.

To start Hive, simply enter the hive command. If Hive starts correctly, you should get a hive>prompt

```
hive> CREATE TABLE pokes (foo INT, bar STRING);
OK
Time taken: 1.705 seconds
hive> SHOW TABLES;
OK
pokes
Time taken: 0.174 seconds, Fetched: 1 row(s)
hive> DROP TABLE pokes;
```

	OK Time taken: 4.038 seconds			
6 (a)	**Describe various features of Hadoop YARN administration.** **4*1=4M** ▶ YARN has several built-in administrative features and commands. To find out more about them, examine the YARN commands documentation at ▶ https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YarnCommands.html#Administration_Commands. The main administration command is yarn rmadmin (resource manager administration). Enter yarn rmadmin -help to learn more about the various options. Decommissioning YARN Nodes ▶ If a NodeManager host/node needs to be removed from the cluster, it should be decommissioned first. ▶ Assuming the node is responding, you can easily decommission it from the Ambari web UI. Simply go to the Hosts view, click on the host, and select Decommission from the pull-down menu next to the NodeManager component. ▶ Note that the host may also be acting as a HDFS DataNode. Use the Ambari Hosts view to decommission the HDFS host in a similar fashion**.**	[4M]	CO2	L2
(b)	**Explain Various HDFS Administration features.** **3*2M** The following section covers some basic administration aspects of HDFS. ▶ **The NameNode User Interface** Monitoring HDFS can be done in several ways. One of the more convenient ways to get a quick view of HDFS status is through the NameNode user interface. This web-based tool provides essential information about HDFS and offers the capability to browse the HDFS namespace and logs ▶ **Adding Users to HDFS** For a full explanation of HDFS permissions, see the following document: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsPermissionsGuide.html. Keep in mind that errors that crop up while Hadoop applications are running are often due to file permissions. following steps **1.** Add the user to the group for your operating system on the HDFS client system. In most cases, the groupname should be that of the HDFS superuser, which is often hadoop or hdfs. `useradd -G <groupname> <username>` **2.** Create the username directory in HDFS. `hdfs dfs -mkdir /user/<username>` **3.** Give that account ownership over its directory in HDFS. `hdfs dfs -chown <username>:<groupname> /user/<username>`	[6M]	CO2	L2

	▶ **Decommissioning HDFS Nodes** If you need to remove a DataNode host/node from the cluster, you should decommission it first. Assuming the node is responding, it can be easily decommissioned from the Ambari web UI. ▶ **SecondaryNameNode** To avoid long NameNode restarts and other issues, the performance of the SecondaryNameNode should be verified. Recall that the SecondaryNameNode takes the previous file system image file (fsimage*) and adds the NameNode file system edits to create a new file system image file for the NameNode to use when it restarts.			