BIG DATA ANALYTICS AND VISUALIZATION LAB

Subject Code: MCAL31

A Practical Journal Submitted in Fulfillment
of the Degree of

MASTER IN COMPUTER APPLICATION

Year 2024-2025

By

Mr. Jadhav Harshal Sanjay

(Application Id: 53942)

Seat No. 1030199

Semester-III

Under the Guidance of

Prof. Bharati Gaikar

Centre for Distance and Online Education
Vidya Nagari, Kalina, Santacruz East – 400098.
University of Mumbai

PCP Center

[Satish Pradhan Dnyanasadhana College, Thane]

Institute of Distance and Open Learning
Vidya Nagari, Kalina, Santacruz East – 400098.

CERTIFICATE

This to certify that, **“Mr. Jadhav Harshal Sanjay "** appearing **Master’s In Computer Application (Semester III) Application Id: 53942** has satisfactorily completed the prescribed practical of **MCAL31 – Big Data Analytics And Visualization Lab** as laid down by the University of Mumbai for the academic year 2024-25.

Teacher In Charge	External Examiner	Coordinator – M.C.A

Date:

Place: -

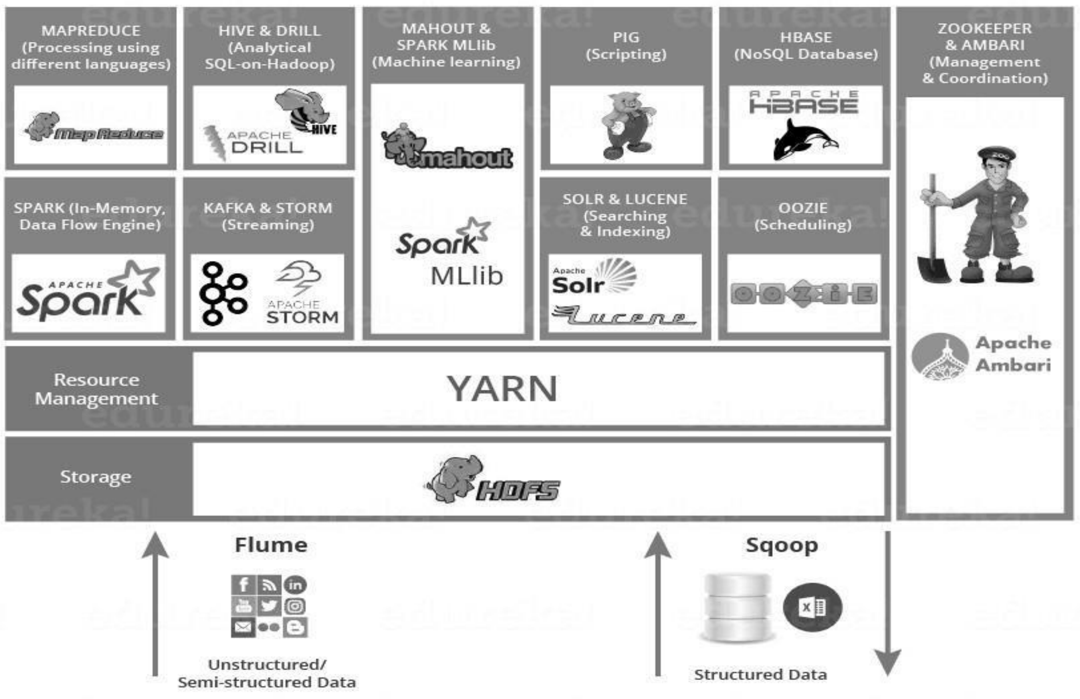
INDEX

Practical No – 1

Aim: Study of Hadoop ecosystem.

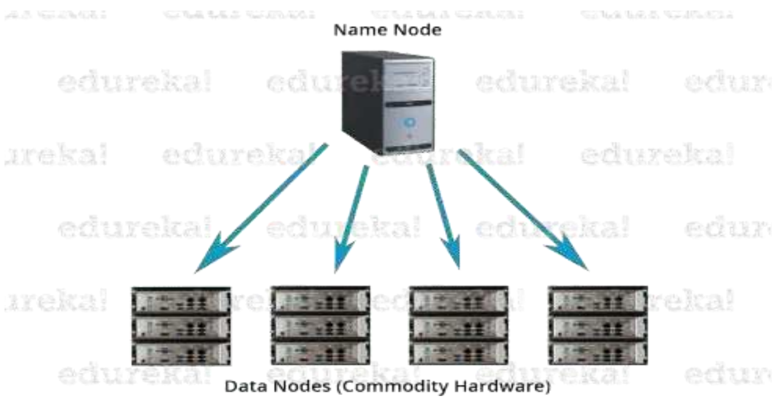
Components of a Hadoop ecosystem

- **HDFS ->**Hadoop Distributed FileSystem
- **YARN ->**Yet Another ResourceNegotiator
- **MapReduce ->** Data processing using programming
- **Spark ->** In-memory DataProcessing
- **PIG, HIVE->** Data Processing Services using Query(SQL-like)
- **HBase ->**NoSQLDatabase
- **Mahout, Spark MLlib ->** MachineLearning
- **Apache Drill ->** SQL onHadoop
- **Zookeeper ->** ManagingCluster
- **Oozie ->** JobScheduling
- **Flume, Sqoop ->** Data IngestingServices
- **Solr Lucene ->** Searching Indexing
- **Ambari ->** Provision, Monitor and Maintaincluster


MAPREDUCE (Processing using different languages)
HIVE & DRILL (Analytical SQL-on-Hadoop)
MAHOUT & SPARK MLlib (Machine learning)
PIG (Scripting)
HBASE (NoSQL Database)
ZOOKEEPER & AMBARI (Management & Coordination)
SPARK (In-Memory, Data Flow Engine)
KAFKA & STORM (Streaming)
SOLR & LUCENE (Searching & Indexing)
OOZIE (Scheduling)
Resource Management
YARN
Storage
HDFS
Flume
Unstructured/ Semi-structured Data
Sqoop
Structured Data
Apache Ambari

HDFS

- Hadoop Distributed File System is the core component or you can say,the backbone of HadoopEcosystem.
- HDFS is the one, which makes it possible to store different types of large data sets (i.e. structured, unstructured and semi structured data).
- It helps us in storing our data across various nodes and maintaining the log file about the stored data(metadata).
- HDFS has two core components, i.e. NameNode andDataNode.
    1. The NameNodeis the main node and it doesn't store the actual data. It contains metadata, just like a log file or you can say as a table of content. Therefore, it requires less storage and high computational resources.
    2. On the other hand, all your data is stored on the DataNodesand hence it requires more storage resources. These DataNodes are commodity hardware (like your laptops and desktops) in the distributed environment. That's the reason why Hadoop solutions are very cost effective.
    3. You always communicate to the NameNode while writing the data. Then, it internally sends a request to the client to store and replicate data on various DataNodes.



YARN

- ○ Consider YARN as the brain of your Hadoop Ecosystem. It performs all your processing activities by allocating resources and scheduling tasks.
- ○ It has two major components, i.e. ResourceManager andNodeManager.

1. ResourceManager is again a main node in the processing department.

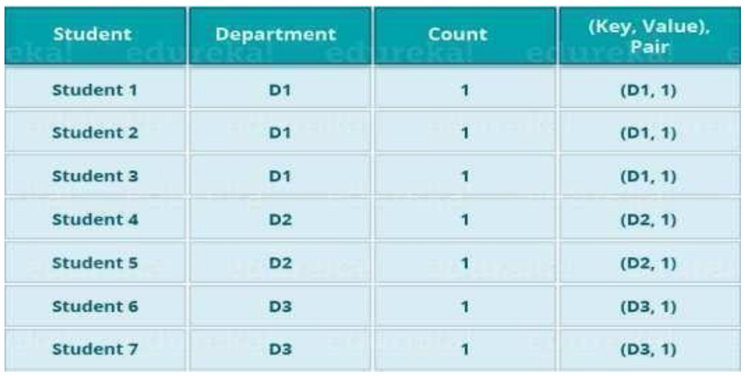
2. NodeManagersare installed on every DataNode. It is responsible for execution of tasks on every singleDataNode.

- Schedulers: Based on your application resource requirements, Schedulers perform scheduling algorithms and allocate the resources.
- ApplicationsManager: While ApplicationsManager accepts the job submission, negotiates to containers (i.e. the Data node environment where process executes) for executing the application specific ApplicationMaster and monitoring the progress. ApplicationMasters are the demons which reside on DataNode and communicates to containers for execution of tasks on each DataNode.ResourceManager has two components, i.e. Schedulers andApplicationsManager.

MAPREDUCE

It is the core component of processing in a Hadoop Ecosystem as it provides the logic of processing. In other words, MapReduce is a software framework which helps in writing applications that processes large data sets using distributed and parallel algorithms inside a Hadoop environment.

- In a MapReduce program, Map() and Reduce() are two functions.
  1. The Map function performs actions like filtering, grouping and sorting.
  2. While the Reduce function aggregates and summarizes the result produced by the map function.
  3. The result generated by the Map function is a key value pair (K, V) which acts as the

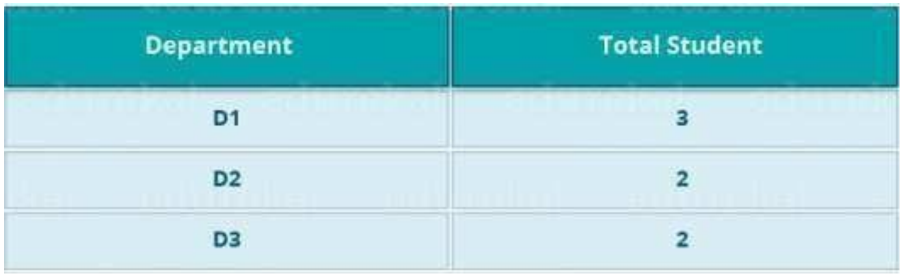


Student	Department	Count	(Key, Value), Pair
Student 1	D1	1	(D1, 1)
Student 2	D1	1	(D1, 1)
Student 3	D1	1	(D1, 1)
Student 4	D2	1	(D2, 1)
Student 5	D2	1	(D2, 1)
Student 6	D3	1	(D3, 1)
Student 7	D3	1	(D3, 1)

input for Reducefunction.

Let us take the above example to have a better understanding of a MapReduce program.
We have a sample case of students and their respective departments. We want to calculate the number of students in each department. Initially, the Map program will execute and calculate the students appearing in each department, producing the key value pair as mentioned above. This key value pair

is the input to the Reduce function.



Department	Total Student
D1	3
D2	2
D3	2

APACHE PIG

- PIG has two parts: Pig Latin, the language and the pig runtime, for the execution environment. You can better understand Java andJVM.
- It supports *pig latin language*, which has SQL like command structure.

  As everyone does not belong from a programming background. So, Apache PIG relieves them. You might be curious to know how?

Well, I will tell you an interesting fact:

10 line of pig latin = approx. 200 lines of Map-Reduce Java code

But don’t be shocked when I say that at the back end of Pig job, a map-reduce job executes.

- The compiler internally converts pig latin to MapReduce. It produces a sequential set of MapReduce jobs, and that’s an abstraction (which works like blackbox).
- PIG was initially developed byYahoo.
- It gives you a platform for building data flow for ETL (Extract, Transform and Load), processing and analyzing huge datasets.

How does the pig work?

In PIG, first the load command loads the data. Then we perform various functions on it like grouping, filtering, joining, sorting, etc. At last, either you can dump the data on the screen or you can store the result back in HDFS.

APACHE HIVE

- Facebook created HIVE for people who are fluent with SQL. Thus, HIVE makes them feel at home while working in a HadoopEcosystem.
- Basically, HIVE is a data warehousing component which performs reading, writing and managing large data sets in a distributed environment using SQL-like interface.

HIVE + SQL = HQL

- The query language of Hive is called Hive Query Language(HQL), which is very similar likeSQL.
- It has 2 basic components: Hive Command Line and JDBC/ODBCdriver.
- The Hive Command line interface is used to execute HQLcommands.

- Java Database Connectivity (JDBC) and Object Database Connectivity (ODBC) are used to establish connections from data storage.
- Secondly, Hive is highly scalable. As, it can serve both the purposes, i.e. large data set processing (i.e. Batch query processing) and real time processing (i.e. Interactive query processing).

APACHE SPARK

- Apache Spark is a framework for real time data analytics in a distributed computing environment.
- The Spark is written in Scala and was originally developed at the University of California, Berkeley.
- It executes in-memory computations to increase speed of data processing overMap-Reduce.
- It is 100x faster than Hadoop for large scale data processing by exploiting in-memory computations and other optimizations. Therefore, it requires higher processing power than Map-Reduce.

APACHE HBASE

- HBase is an open source, non-relational distributed database. In other words, it is a NoSQL database.
- It supports all types of data and that is why it's capable of handling anything and everything inside a Hadoop Ecosystem.
- The HBase was designed to run on top of HDFS and provides BigTablelikecapabilities.
- Itgivesusafaulttolerantwayofstoringsparsedata,whichiscommoninmostBigDatause cases.
- The HBase is written in Java, whereas HBase applications can be written in REST, Avro and ThriftAPIs.

APACHE ZOOKEEPER

- Apache Zookeeper is the coordinator of any Hadoop job which includes a combination of various services in a HadoopEcosystem.
- Apache Zookeeper coordinates with various services in a distributed environment.
- Due to the above problems, Zookeeper was introduced. It saves a lot of time by performing synchronization, configuration maintenance, grouping and naming.

<u>Conclusion:</u> The Hadoop ecosystem provides a comprehensive framework for distributed storage and processing of large datasets, enabling efficient big data analysis and management through its diverse tools.

Practical No – 2

Aim: To set up Multi node Hadoop and execute a Map- reduce program.

Apache Hadoop is an open source framework used for distributed storage as well as distributed processing of big data on clusters of computers which run on commodity hardware. Hadoop stores data in the Hadoop Distributed File System (HDFS) and the processing of this data is done using MapReduce. YARN provides an API for requesting and allocating resources in the Hadoop cluster.

The Apache Hadoop framework is composed of the following modules:

- Hadoop Common
- Hadoop Distributed File System(HDFS)
- YARN
- MapReduce

Steps in the installation

- Add users for Hadoop Environment
- Install and configure the Oracle JDK
- Configure passwordless less SSH
- Install Hadoop and configure necessary related xml files
- Start the Hadoop Cluster
- Access Name Node and Resource Manager Web UI

In standalone mode, all Hadoop operations will run in a single JVM. In Hadoop Standalone mode, a local file system is used as the storage and a single JVM that will perform all MR-related operations.
Setting up Apache Hadoop 2.8.0 in Standalone Mode

Step 1: Ensure package lists are updated.
```
sudo apt-get update
```

Step 2: Install latest version of Java
```
sudo apt-get install openjdk-11.8.0-jdk
java –version
```

Step 3: Install SSH
```
sudo apt-get install openssh-server
```

Step 4: Download hadoop-2.8.0.tar.gz using the browser and copy it to the home folder and extract the Hadoop binary tar file that you have built and copied in the home folder.
```
tar –xvzf hadoop-2.8.0.tar.gz
```

Step 5: Rename the extracted folder. (This is done for our comfort).
```
mv hadoop-2.8.0 hadoop2
```

Step 6: Setup Environment Variables to identify Hadoop executable, configurations, and dependencies. You will need to edit the .bashrc file that is available in the home folder.
```
sudo gedit .bashrc
```

```
#Add the below lines at the start of the file
export JAVA_HOME = /usr/lib/jvm/java-1.7.0-openjdk-amd64
export HADOOP_INSTALL=/home/hadoop/hadoop2
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS=”-Djava.library.path=$HADOOP_INSTALL/lib”
export HADOOP_CONF_DIR=$HADOOP_INSTALL/etc/hadoop
export YARN_CONF_DIR =$HADOOP_INSTALL/etc/hadoop
export PATH=$PATH:$HADOOP_CONF_DIR/bin
export PATH=$PATH:$YARN_CONF_DIR/sbin
```

step 7: Refresh and apply the environment variables.

```
exec bash
```

Step 8: Inform Hadoop where Java is gedit /home/ hadoop2/libexec/hadoop-config.sh

```
#Add the following line at the start of file
export  JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64
```

Step 9: Setup Hadoop Environment Variables. This is mostly used by HDFS shell scripts that are present in the sbin location of Hadoop framework.

```
gedit /home/hadoop2/etc/hadoop/hadoop-env.sh
#Add the following line at the start of file
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64/
export HADOOP_INSATLL=/home/hadoop2
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export YARN_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_INSTALL/lib/native
export HADOOP_OPTS=”-Djava.library.path=$HADOOP_INSTALL/lib”
```

Step 10: Setup YARN variables. This is mostly used by YARN shell scripts present in the sbin location of the Hadoop framework.

```
gedit /home/hadoop2/etc/hadoop/yarn-env.sh

#Add the following line at the start of file
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-amd64/
export HADOOP_HOME=/home/hadoop2
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export YARN_CONF_DIR=$HADOOP_HOME/etc/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
```

Step 11: Use the example jar file provided by the framework. This is available in the location /home/hadoop2/share/hadoop/mapreduce folder. Now open two terminal windows. In the first session, type the following command.

watch –n 1 jps
This command runs every 01 second which will enable monitoring of any newly created JVMs.
In the second session, type the following command to execute a "WordCount" Program:

```
hadoop jar /home/mrunali/hadoop2/share/hadoop/mapreduce/*-examples*.jar  wordcount
/home/mrunali/hadoop2/WordCountInput.txt OutputWC2
```

Syntax is :

```
hadoop jar <jar_file><program_name><input><output>
```

where,
jar_file – The location of MR JAR which is to be executed in Hadoop Cluster.
Program_file – The name of the class which is to be processed.
Input – The file input location that is to be processed.
Output – The folder output location where all the output will be stored.

Conclusion : After setting up all environment variables, Hadoop has been installed in Standalone mode. To Test the installation run the jsp command, and another session can be used to execute a MapReduce application in Standalone mode.

2] Setting up Apache Hadoop 2.8.0 in Pseudo-Distributed Mode (Single Node Cluster)

Step 1: Setup your mode in Standalone (CLI MiniCluster) mode:

Step 2: Setup SSH passwordless setup (required for Hadoop daemons since they get initialized using SSH). This can be done using the following steps:

Generate SSH keys.

Ssh-keygen (Press Enter till the command gets completed. No need to fill anything including password since we need to set up a 'passwordless' key).

This command will generate

i) the public key(id_rsa.pub) and
ii) the private key(id_rsa) in the home folder under .ssh directory

Register the public key in this node.

```
ssh-copy-id –i home/.ssh/id_rsa.pub hadoop@hadoopvm
```

where, hadoop is the username and hadoop vm is the Hostname of the machine.
After pressing the Enter key, you may get a prompt to register the system to known_hosts. Type 'yes' and press Enter. Once done, you will see a prompt for the password. Enter the user's password and press Enter. You will see a feedback line that says the key is being added successfully.

Step 3: Setup core-site.xml. This configuration file is responsible for informing the framework where the NameNode daemon will run. It also maintains all Hadoop core configurations. You can get all configurations with their default values and descriptions in the link given below:
https://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoop-common/core-default.xml

```
gedit /home/hadoop2/etc/hadoop/core-site.xml
<?xml version=”1.0” encoding=”UTF-8”?>
<?xml-stylesheet type=”text/xsl” href=”configuration.xsl”?>
<configuration>
<property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:8020</value>
</property>
<property>
        <name>hadoop.tmp.dir</name>
        <value>/home/hadoop/hdfsdrive</value>
</property>
</configuration>
```

Step 4: Setup mapred-site.xml. This configuration is responsible for maintaining MapReduce job configurations. Now configure MR jobs to run on top of YARN. This file is not available by default. A template file is provided.

```
cp /home/hadoop2/etc/hadoop/marped-site.xml.template /home/hadoop2/etc/hadoop/mapred-site.xml
gedit /home/hadoop2/etc/hadoop/mapred-site.xml
<?xml version=”1.0” encoding=”UTF-8”?>
<?xml-stylesheet type=”text/xsl” href=”configuration.xsl”?>
<configuration>
<property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
</property>
</configuration>
```

Step 5: Setup hdfs-site.xml. This configuration file is responsible for configuring your HDFS.

```
gedit /home/hadoop2/etc/hadoop/hdfs-site.xml
<?xml version=”1.0” encoding=”UTF-8”?>
<?xml-stylesheet type=”text/xsl” href=”configuration.xsl”?>
<configuration>
<property>
        <name>dfs.replication</name>
        <value>1</value>
</property>
</configuration>
```

Step 6: Setup yarn-site.xml. This configuration file is responsible for configuring the YARN related parameters.

```
gedit /home/hadoop2/etc/haoop/yarn-site.xml
<?xml version=”1.0” encoding=”UTF-8”?>
<?xml-stylesheet type=”text/xsl” href=”configuration.xsl”?>
<configuration>
<property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
```

```
</property>
<property>
        <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
</configuration>
```

Step 7: Create a folder named hdfsdrive inside the home folder. This is as per the value set in hadoop.tmp.dir in core-site.xml

```
mkdir –p /home/hadoop/hdfsdrive
```

Step 8: Format the NameNode. This setup ensures that Hadoop will create a filesystem in the location that is defined in hadoop.tmp.dir of core-site.xml. In this case, it is /home/hadoop/hdfsdrive. Be alert towards checking the Hadoop version prior to formatting as a form of Best Practice. This practice will enable you to be conscious of the Hadoop version that the system has accepted. This is required when you are connected to PuTTY using SSH since the environment may get updated, but the session may not get updated, but the session may not get refreshed. In case the session is not refreshed, it is recommended that you re-login the SSH shell.

```
hadoop version
hadoop namenode –format
```

In the format, if you view the success file as shown below, you can say that your HDFS has been created successfully.

Step 8: Let us now start Hadoop daemons. For now, we will learn how to start the daemons manually.

```
hadoop-daemon.sh  startnamenode
hadoop-daemon.sh  startdatanode
yarn-daemon.sh startresourcemanager
yarn-daemon.sh  startnodemanager
mr-jobhistory-daemon.sh  start hi story manager
```

After running the start script, use the jps Java utility to check that all the processes are running. You should see the output that follows, although the ordering and process IDs will differ

```
        $jps
                4433 Jps
                4358 NameNode
                3926 NodeManager
                2156 ResourceManger
                4159 DataNode
```

Mapreduce Program:

Word Count Program in Java

Steps

1. Open Eclipse> File > New > Java Project >(Name it – MRProgramsDemo) >Finish.
2. Right Click > New > Package (Name it - PackageDemo) >Finish.
3. Right Click on Package > New > Class (Name it -WordCount).
4. Add Following ReferenceLibraries:
    1. Right Click on Project > Build Path> AddExternal

1. */usr/lib/hadoop-0.20/*hadoop-core.jar
2. *Usr/lib/hadoop-0.20/lib/*Commons-cli-1.2.jar

5. Typecode

```
packagePackageDemo;
import java.io.IOException;
import
org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
importorg.apache.hadoop.io.IntWritable;
importorg.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;
importorg.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
public static void main(String [] args) throws Exception
{
Configuration c=new Configuration();
String[] files=new
 GenericOptionsParser(c,args).getRemainingArgs();
Path input=new Path(files[0]);
Path output=new Path(files[1]);
Job j=new Job(c,"wordcount");
j.setJarByClass(WordCount.class);
j.setMapperClass(MapForWordCount.class);
j.setReducerClass(ReduceForWordCount.class);
j.setOutputKeyClass(Text.class);
j.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(j, input);
FileOutputFormat.setOutputPath(j, output);
System.exit(j.waitForCompletion(true)?0:1);
 public static class MapForWordCount extends Mapper<LongWritable, Text, Text,
 IntWritable>{
  public void map(LongWritable key, Text value, Context con) throws IOException,
  InterruptedException
  {
  String line =value.toString();
  String[]words=line.split(",");
  for(String word: words)
  {
      Text outputKey = new Text(word.toUpperCase().trim());
    IntWritableoutputValue = new IntWritable(1);
    con.write(outputKey, outputValue);
  }
  }
  }
  public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>
  {
```

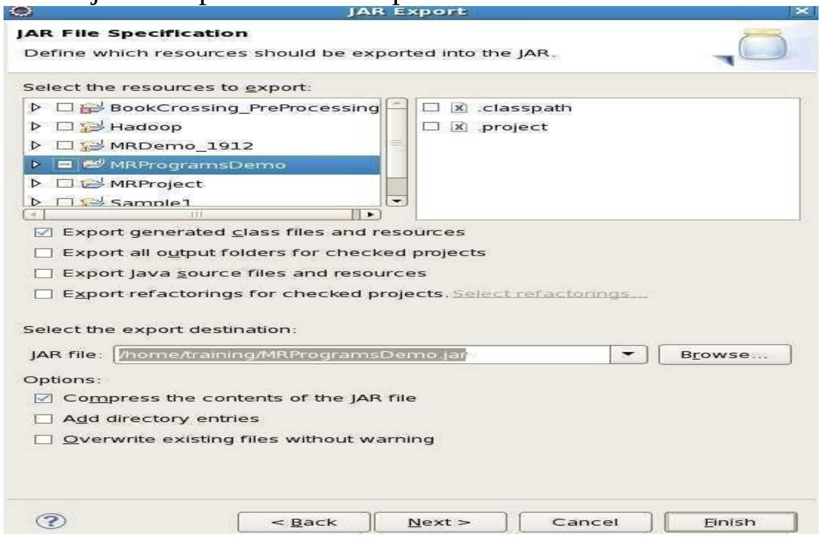
```
public void reduce(Text word, Iterable<IntWritable> values,Context con) throws
IOException, InterruptedException
            {
int sum = 0;
  for(IntWritable value : values)
  {
  sum += value.get();
  }
  con.write(word, new IntWritable(sum));
}
}
}
```

The above program consists of three classes:

- Driver class (Public, void, static, or main; this is the entrypoint).
- The Map class which extends the public class Mapper<KEYIN,VALUE IN,KEYOUT,VALUEOUT>and implements the Mapfunction.
- The Reduce class which extends the public class Reducer<KEYIN,VALUE IN,KEYOUT,VALUEOUT>and implements the Reducefunction.

6. Make a jar file

Right Click on Project> Export> Select export destination as Jar File > next>Finish.



JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

- ☐ BookCrossing_PreProcessing
- ☐ Hadoop
- ☐ MRDemo_1912
- ▣ MRProgramsDemo
- ☐ MRProject
- ☐ Sample1

- ☐ ☒ .classpath
- ☐ ☒ .project

- ☑ Export generated class files and resources
- ☐ Export all output folders for checked projects
- ☐ Export Java source files and resources
- ☐ Export refactorings for checked projects. Select refactorings...

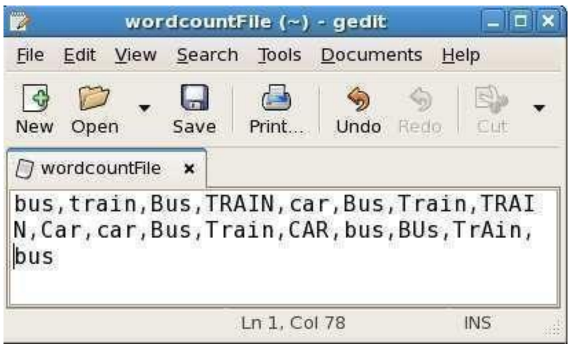
Select the export destination:

JAR file: /home/training/MRProgramsDemo.jar

Options:

- ☑ Compress the contents of the JAR file
- ☐ Add directory entries
- ☐ Overwrite existing files without warning

7. Take a text file and move it into HDFSformat:



wordcountFile (~) - gedit

```
bus,train,Bus,TRAIN,car,Bus,Train,TRAI
N,Car,car,Bus,Train,CAR,bus,BUs,TrAin,
bus
```

Ln 1, Col 78   INS

To move this into Hadoop directly, open the terminal and enter the following commands:

```
[training@localhost ~] $ hadoop fs -put word count File word Count File
```

8. Run the jar file:

(Hadoop jar jarfilename.jar packageName.ClassName PathToInputTextFilePathToOutput Directory)

```
[training@localhost ~]$ hadoop jarMRProgramsDemo.jar
PackageDemo.WordCountwordCountFile MRDir1
```

9. Open theresult:

```
[training@localhost ~]$ hadoop fs -ls MRDir1 Found 3 items
-rw-r--r--    1 training supergroup 03:36/user/training/MRDir1/_SUCCESS drwxr-xr-x     -
training supergroup
03:36 /user/training/MRDir1/_logs
-rw-r--r--    1 training supergroup
02016-02-23
02016-02-23
03:36 /user/training/MRDir1/part-r-00000
20 2016-02-23
[training@localhost ~]$ hadoopfs -catMRDir1/part-r-00000

BUS CAR TRAIN
7
4
6
```

Conclusion:

Configuring a multi-node Hadoop cluster and executing a Map-Reduce program demonstrates the scalability and parallel processing capabilities of Hadoop for handling large datasets.

Practical No – 3

Aim: Design flume agent to download the web tweets from twitter and analyze using HIVE/PIG

Word clouds are one of the simplest and most intuitive ways of visualizing text data. The data collected in the last two weeks, from most of all public tweets related to what the people are talking around the globe The w cloud are generated through the following process. First, a computer program takes a text and counts how frequent each word is.
for example if there was a character named “Cat” as well as generic animals referred to as “cat”. Also removes top words, which add little to the final visualization in many cases .Second ,creation of word frequency list and incorporating these changes, then the program puts them into the ready queue and starts to print them. The word most frequently appears is placed as the highest position.
For getting tweets from Twitter, there is a need to create a Twitter application.

Twitter application creation steps are given below

Step 1
First click with link https://apps.twitter.com/and sign in the twitter account and do some work with twitter Application window where you are able to create, delete, and manage Twitter Apps.

Step 2
In the next step click on the Create New App button. Then you will be going to a window where you will get an application form to fill in your detailed information.

Step 3
Then the new App will be created. New app is used to create Consumer Key, Access Key, and Access Token Key. This will be used to edit in the Flume.coffle. While fetching data from Twitter these Consumer Key, Access Key, Access Token Key is used to fetch data which is lively tweeting in the account.

Step 4
These keys are used to Access Tokens tab and it can observe a button with the name of Create my access token. By Clicking this we can generate the access token.

Step 5
Finally, click on the right side top Test OAuth button of the page, Which displays the Consumer key, secret, Access token. These are used to configure the agent in Flume.

Step 6
Consumer key, Secret, Access token are used to configure the Flume agent.

CONFIGURING FLUME
Twitter 1% Firehose Source
Only a single percentage(1%)of sample twitter firehose is experimented by streaming Application program interface(API) for converting them to Avro format, and sends
To a downstream Flume Sink. The jar fles source can be located in the lib folder.

Setting the class path

Then set the classpath in the lib folder of Flume by exporting
CLASSPATH=$CLASSPATH:/FLUME_HOME/lib/*
While configuring this source,you have to provide values to the Source type, consumerKey, consumerSecret, accessToken, accessTokenSecret, maxBatchSize by default value is 1000.

SETTING UP HADOOP CLUSTER

- Download and setup java-jdk.
- Download hadoop package format “hadoop-2.7.2.tar.gz” and unzip it.
- Setup up HADOOP_HOME environment.
- Configure the hadoop files inside the conf directory.
- Once all the configuration is done run the terminal command“jps”to view all the nodes are working.Now open the browser and enter into localhost:50070 to view the hadoop.

LOADING DATA INTO HDFS

HDFS sink comes stock with ApacheFlume.Easily separates files by creation time.Data“will be loaded into this hdfs location:-hdfs://localhost:8020/user/flume/news.

PERFORMANCE ANALYSIS ANALYSIS BASED ON SERVICE

The following Figure shows the framework for the computer Performance Measurement in Cloud Computing In this system establishes a set of performance criteria based on characteristics Fetching the data using Flume
After creating an application in the Twitter developer site, we need to fetch the data from Twitter. For that:
We will use the consumer key and secret key with the access token and secret values.
Further, we can fetch data through twitter that is required and it will be in JSON format and we will put this data in the HDFS in the location where we have saved all the data that comes from the Twitter.
The configuration file is used to get the real time data from the Twitter.
All the details sort the points of interest needed to fill in the flume-twitter.conf file i.e configuration file of Flume.

Query utilizing HQL

After setting the above design, the flume is runned, the Twitter data/information will automatically be saved into HDFS in different directories where we have set the storage path to save the Twitter data/information that was extracted by using Flume.
To keep the data in local file directory is not feasible because loading data in a local file directory is a lengthy process.
From the collected data we will create table “mytweets_raw” where the filtered data will be kept into a formatted structure so that we can clearly show that we have converted the unstructured data into structured data or in an organized way.
After loading the real time data in the hive table, more tables are created like a dictionary table which stores the polarity and the word and tweets_sentiment table which will contain all the tweets id and its sentiment. Many such more tables are created and different operations are done on data

Datasets And Result Analysis

Before applying queries on the data, we need to make sure that the Hive table can appropriately translate the JSON formatted data by using a JSON validator. Hive takes input files which use a delimited row format, yet our fetched data is in a JSON format, which will not work.

<u>Conclusion:</u>

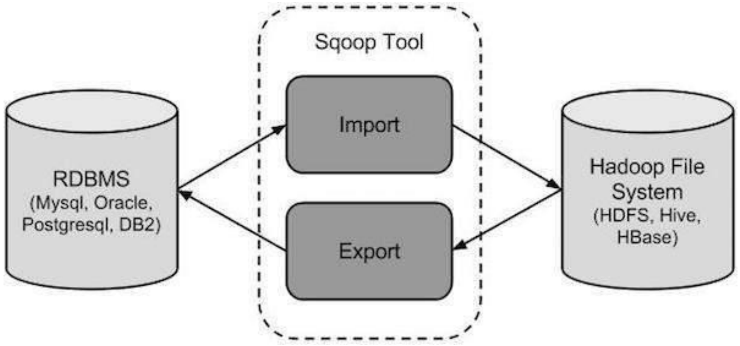
Using Flume to collect tweets and analyzing them with HIVE/PIG demonstrates real-time data processing and analysis.

Practical No – 4

Aim: Load the data from RDBMS e.g. customer data / student data using SQOOP and analyze it using HIVE/ PYSPARK/ Spark SQL

Theory :

How Sqoop Works?



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.

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 the table. Those are read and parsed into a set of records and delimited with user-specified delimiter.

Step 1: Verifying JAVA Installation

Step 2: Verifying Hadoop Installation

Step 2.1: Name Node Setup

```
hdfs namenode-format
```

Step 2.2: Verifying Hadoop Dfs

```
$ start-dfs.sh
```

Step 2.3: Verifying Yarn Script

```
$ start-yarn.sh
```

Step 2.4: Accessing Hadoop on Browser

```
http://localhost:50070/
```

Step 2.5: Verify All Applications for Cluster

```
http://localhost:8088/
```

Step 3: Downloading Sqoop

download the latest version of Sqoop from the following link https://sqoop.apache.org/

download sqoop-1.4.7.bin_hadoop-2.6.0.tar.gz

Step 4: Installing Sqoop

```
$tar -xvf sqoop-1.4.4.bin_hadoop-2.0.4-alpha.tar.gz
$mv sqoop-1.4.4.bin_hadoop-2.0.4-alpha /usr/lib/sqoop
```

Step 5: Configuring bashrc
set up the Sqoop environment by appending the following lines to ~/.bashrc file

```
#Sqoop
export SQOOP_HOME=/usr/lib/sqoop export PATH=$PATH:$SQOOP_HOME/bin
execute~/.bashrc file.
$ source~/.bashrc
```

Step 6: Configuring Sqoop

To configure Sqoop with Hadoop, you need to edit the sqoop-env.sh file, which is placed in the $SQOOP_HOME/conf directory. First of all, Redirect to Sqoop Config directory and copy the template file using the following command −

```
$ cd $SQOOP_HOME/conf
$ mv sqoop-env-template.sh sqoop-env.sh
```

Open sqoop-env.sh and edit the following lines −

```
export HADOOP_COMMON_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=/usr/local/hadoop
```

Step 7: Download and Configure mysql-connector-java
download mysql-connector-java-5.1.30.tar.gz file from the following link
http://ftp.ntu.edu.tw/MySQL/Downloads/Connector-J/mysql-connector-java-8.0.14.tar.gz

extract mysql-connector-java tarball and move mysql-connector-java-5.1.30-bin.jar to /usr/lib/sqoop/lib directory.

```
$ tar-zxf mysql-connector-java-5.1.30.tar.gz
$ su
password:
# cd mysql-connector-java-5.1.30
# mv mysql-connector-java-5.1.30-bin.jar /usr/lib/sqoop/lib
```

Step 8: Verifying Sqoop

The following command is used to verify the Sqoop version.

```
$ cd $SQOOP_HOME/bin
$ sqoop-version
```

Expected output −

```
14/12/17 14:52:32 INFO sqoop.Sqoop: Running Sqoop version: 1.4.5
Sqoop 1.4.5 git commit id 5b34accaca7de251fc91161733f906af2eddbe83
Compiled by abe on Fri Aug 1 11:19:26 PDT 2014
```

Sqoop:

Syntax : The following syntax is used to import data into HDFS.

```
$ sqoop import(generic-args)(import-args)
$  sqoop-import(generic-args)(import-args)
```

mysql -u hadoop -p123456

create database demo;

```
use demo;

create table emp
    -> (eid int,
    ->esal int);
        insert into emp values(1,1000); insert into emp values(2,2000); insert into emp values(3,3000);
insert into emp values(4,4000); insert into emp values(5,5000); insert into emp values(6,6000); select *
fromemp;
+.........+.........+
| eid | esal |
+.........+.........+
|    1 | 1000|
|    2 | 2000|
|    3 | 3000|
|    4 | 4000|
|    5 | 5000|
|    6 | 6000|
+.........+.........+
exit;
        sqoop import --connect jdbc:mysql://localhost:3306/demo --username hadoop --password
123456 -- table emp --target-dir /sqooptrf/emp -m 1

hive> create external table employee
        > (eidint,
        > eslint)
        > comment 'EmployeeDetails'
        > row format delimited
        > fields terminated by ','
        > lines terminated by '\n'
        >location sqoop rf/emp';
```

Conclusion: Loading relational data using SQOOP and analyzing it with modern tools like HIVE, PySpark, or Spark SQL bridges the gap between traditional databases and big data ecosystems, enabling seamless data transformation and analysis.

Practical No – 5

Aim: Create an HIVE Database for employee management systems and analyze it using HIVEQL.

hive

```
create database test1;
show databases;
hive> create table employee
      > (eid int,
      > name String,
      > salary String,
      > destination String)
      > comment 'Employee Details'
      > row format delimited
      > fields terminated by ','
      > lines terminated by '\n'
      > Stored As Text File;
```

Create sample.txt file

Add the following data in sample.txt and save it and close it.

```
1201 Gopal 45000 Technical manager
1202 Manisha 45000 Proof reader
1203 Masthan Vali 40000 Technical writer
1204 Krian 40000 Hr Admin
1205 Kranthi 30000 Op Admin
```

Load

We can insert data using the LOAD DATA statement. While inserting data into Hive, it is better to use LOAD DATA to store bulk records.

There are two ways to load data:

- local file system and
- Hadoop file system.

To LOAD data:

```
LOAD DATA LOCAL INPATH '/home/hadoop/hive/sample.txt' OVERWRITE INTO TABLE
employee;
```

Verify

```
select * from employee;
```

HiveQL Syntax

Select and Where Query

```
SELECT * FROM employee WHERE salary>30000;
```

OrderBy

```
SELECT Id, Name, Dept FROM employee ORDER BY DEPT;
```

GroupBy

```
SELECT Dept,count(*) FROM employee GROUP BY DEPT;
```

Drop a table

View tables

Drop a database

View databases

Joins

JOINS is a clause that is used for combining specific fields from two tables by using values common to each one. It is used to combine records from two or more tables in the database. It is more or less similar to SQL JOINS.

__Conclusion:__ **Creating an HIVE database and analyzing it with HIVEQL showcases efficient management and querying of structured data.**

Practical No – 6

Aim: Design NoSQL dataset using MongoDB and analyze it. Perform CRUD (Create, Read, Update and Delete) queries using Query Language.

Query Language.

MongoDB is an open-source document database and leading NoSQL database.
MongoDB is a cross-platform, document oriented database that provides high performance, high availability,and easy scalability.MongoDB works on the concept of collection and document.
MongoDB is a cross-platform, document oriented database that provides high performance, high availability, and easy scalability.
Any relational database has a typical schema design that shows the number of tables and the relationships between these tables. While in MongoDB, there is no concept of relationship.

Advantages of MongoDB over RDBMS

- Schemaless−MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
- Structure of a single object is clear.
- No complex joins.
- Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful asSQL.
- Tuning.
- Ease Scale-out–MongoDB is easy to scale.

# MongoDB_Lab - Installation Steps

Steps to install mongodb

1. Update ubuntu system sudo apt-get update
2. Upgrade Softwares
   i. sudo apt-get upgrade
3. Install Mongodb
   i. sudo apt-get install mongodb
4. Download following json dataset
   i. https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/dataset.json
   ii. Rename it as primer-dataset.json save it in the home directory
5. Start mongodb services

    i. sudo service mongodb start

6. Import restaurants dataset (primer-dataset.json) into mongodb database
    i. sudo mongoimport --db test --collection restaurants --drop --file primer-dataset.json
7. Get into mongo prompt for db query mongo

```
# End of File
MongoDB Operations

# Save primer-dataset.json in the home directory # Start mongodb services
sudo service mongodb start

# Import restaurants dataset (primer-dataset.json) into mongodb database sudo
mongoimport --db test --collection restaurants --drop --
file primer-dataset.json

# Get into mongo prompt for db query mongo
# Insert a Document
# Insert a document into a collection named restaurants. The operation will create the
collection if the collection does not currently exist.

db.restaurants.insert(
{
"address" :{    "street" : "2 Avenue", "zipcode" :"10075",    "building" :"1480",
 "coord" : [ -73.9557413, 40.7720266]
 },
"borough" :"Manhattan",
"cuisine" :"Italian",
"grades" :[
 {
   "grade" :"A","score" :11
 },
 {
    "grade" :"B","score" :17
 }],
```

"name" : "Vella","restaurant_id" : "41704620" }

Find or QueryData

# You can use the find() method to issue a query to retrieve data from a collection in MongoDB. All queries in MongoDB have the scope of a single collection.
# Queries can return all documents in a collection or only the documents that match a specified filter or criteria. You can specify the filter or criteria in a document and pass as a parameter to the find() method.

# The result set contains all documents in the restaurant's collection.
```
db.restaurants.find()
```

# The result set includes only the matching documents.
```
db.restaurants.find( { "borough": "Manhattan" } )
```

# The following operation specifies an equality condition on the zipcode field in the address embedded document.
# The result set includes only the matching documents.
```
db.restaurants.find( { "address.zip code": "10075" } )
```

# Update Data
# You can use the update() method to update documents of a collection. The method accepts as its parameters:

1. a filter document to match the documents to update,
2. an update document to specify the modification to perform,and
3. an options parameter (optional).

```
db.restaurants.update({'cuisine':'American'},{$set:{'cuisine':'Indian'}})
db.restaurants.update(
{ "name" : "Juni" },
{
        $set: { "cuisine": "American (New)" }
}) db.restaurants.update(
{ "restaurant_id" : "41156888" },
                {$set:{"address.street":"East31stStreet"}})
```

# Remove Data

# You can use the remove() method to remove documents from a collection. The method takes a condition document that determines the documents to remove.
```
db.restaurants.remove( { "borough": "Manhattan" } )
```

# End of File

<u>Conclusion:</u>

Implementing a NoSQL dataset in MongoDB and performing CRUD operations emphasizes the flexibility and scalability of MongoDB for handling unstructured and semi-structured data

Practical No – 7

Aim: Load dataset using SQOOP and analyze the same using PIG script.x

Prerequisites: It is essential that you have Hadoop and Java installed on your system before you go for Apache Pig.

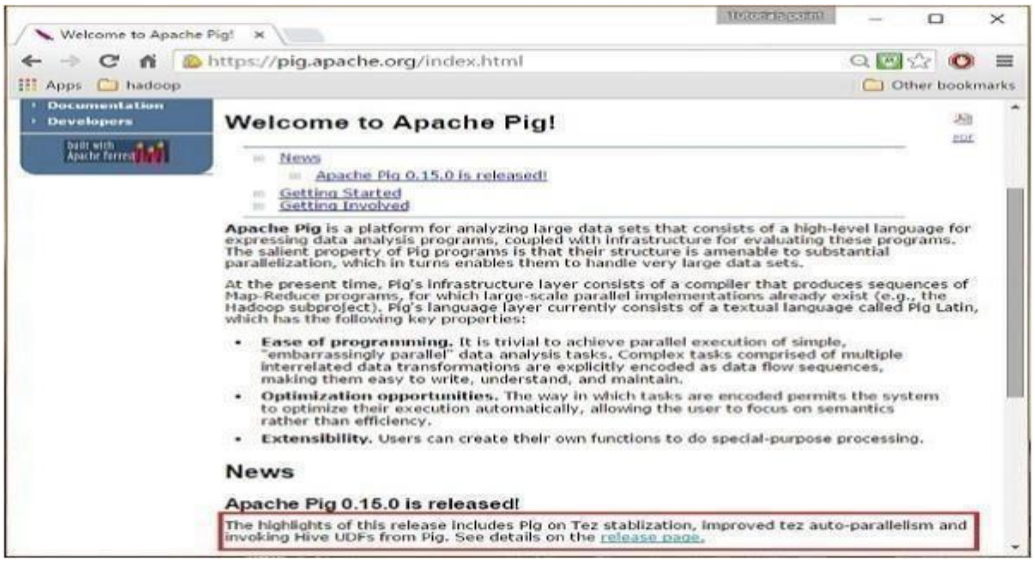
Download Apache Pig

First Also downloadthelatestversionofApachePigfrom the following website−

https://pig.apache.org/

Step 1

Open the homepage of Apache Pig website. Under the section News, click on the link release page as shown in the following snapshot.



Welcome to Apache Pig!

- News
  - Apache Pig 0.15.0 is released!
- Getting Started
- Getting Involved

Apache 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, which has the following key properties:

- **Ease of programming.** 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.
- **Optimization opportunities.** 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.
- **Extensibility.** Users can create their own functions to do special-purpose processing.

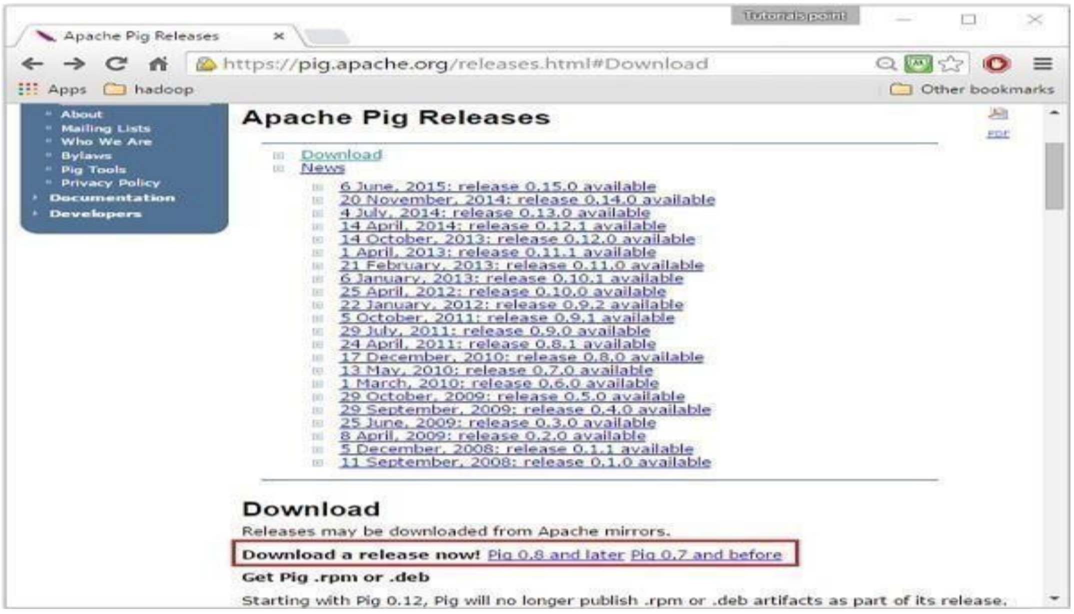
News

Apache Pig 0.15.0 is released!

The highlights of this release includes Pig on Tez stablization, improved tez auto-parallelism and invoking Hive UDFs from Pig. See details on the release page.

Step 2

On clicking the specified link, you will be redirected to the Apache Pig Releases page. On this page, under the Download section, you will have two links, namely, Pig 0.8 and later and Pig 0.7 and before. Click on the link Pig 0.8 and later, then you will be redirected to the page having a set of mirrors.



Apache Pig Releases

- Download
- News
  - 6 June, 2015: release 0.15.0 available
  - 20 November, 2014: release 0.14.0 available
  - 4 July, 2014: release 0.13.0 available
  - 14 April, 2014: release 0.12.1 available
  - 14 October, 2013: release 0.12.0 available
  - 1 April, 2013: release 0.11.1 available
  - 21 February, 2013: release 0.11.0 available
  - 6 January, 2013: release 0.10.1 available
  - 25 April, 2012: release 0.10.0 available
  - 22 January, 2012: release 0.9.2 available
  - 5 October, 2011: release 0.9.1 available
  - 29 July, 2011: release 0.9.0 available
  - 24 April, 2011: release 0.8.1 available
  - 17 December, 2010: release 0.8.0 available
  - 13 May, 2010: release 0.7.0 available
  - 1 March, 2010: release 0.6.0 available
  - 29 October, 2009: release 0.5.0 available
  - 29 September, 2009: release 0.4.0 available
  - 25 June, 2009: release 0.3.0 available
  - 8 April, 2009: release 0.2.0 available
  - 5 December, 2008: release 0.1.1 available
  - 11 September, 2008: release 0.1.0 available

Download

Releases may be downloaded from Apache mirrors.

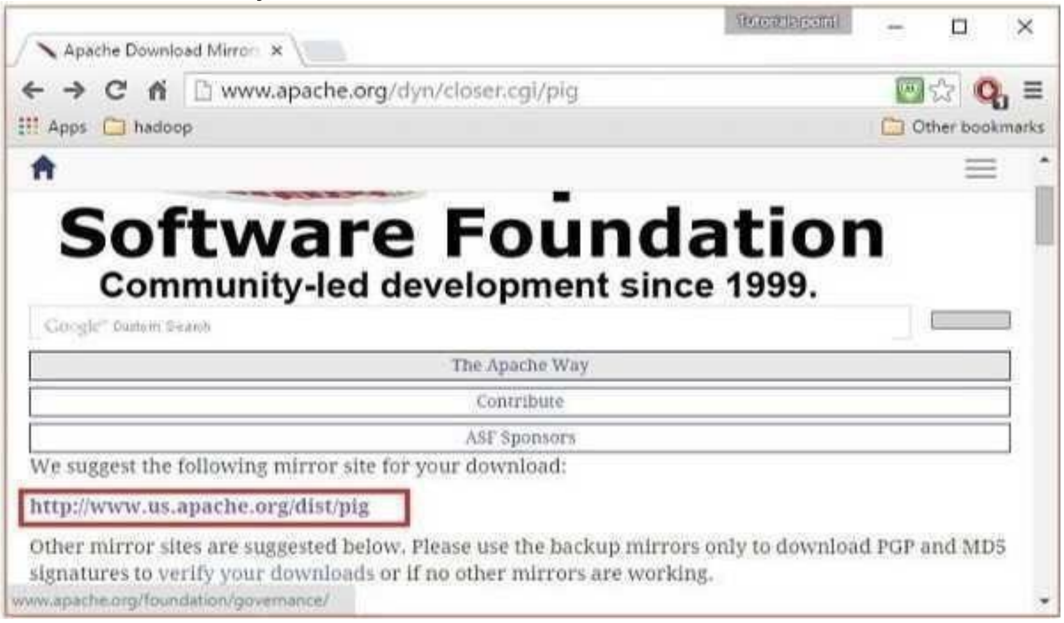
Download a release now! Pig 0.8 and later Pig 0.7 and before

Get Pig .rpm or .deb

Starting with Pig 0.12, Pig will no longer publish .rpm or .deb artifacts as part of its release.

Step 3

Choose and click any one of these mirrors as shown below.



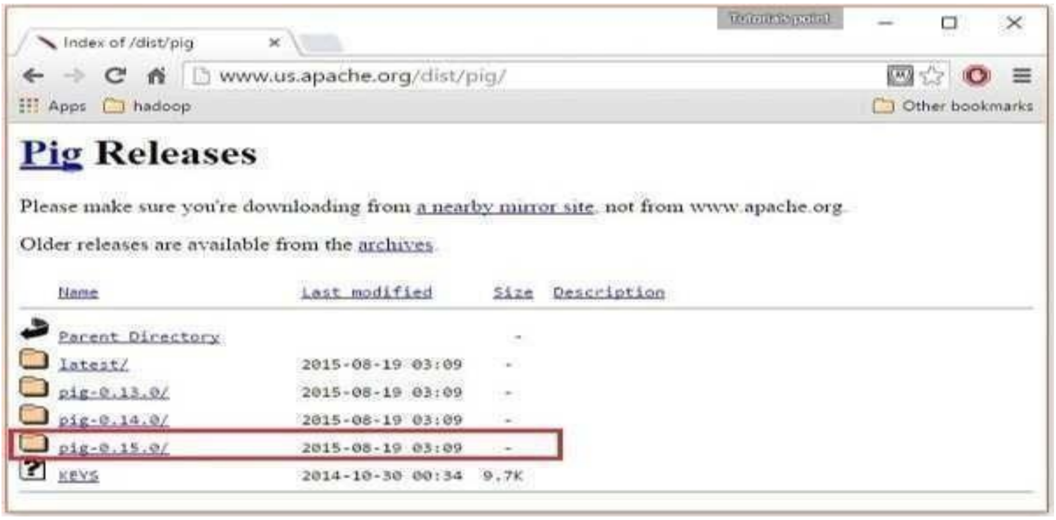
We suggest the following mirror site for your download:

http://www.us.apache.org/dist/pig

Other mirror sites are suggested below. Please use the backup mirrors only to download PGP and MD5 signatures to verify your downloads or if no other mirrors are working.

Step 4

These mirrors will take you to the Pig Releases page. This page contains various versions of Apache Pig. Click the latest version among them.



Index of /dist/pig

www.us.apache.org/dist/pig/

Pig Releases

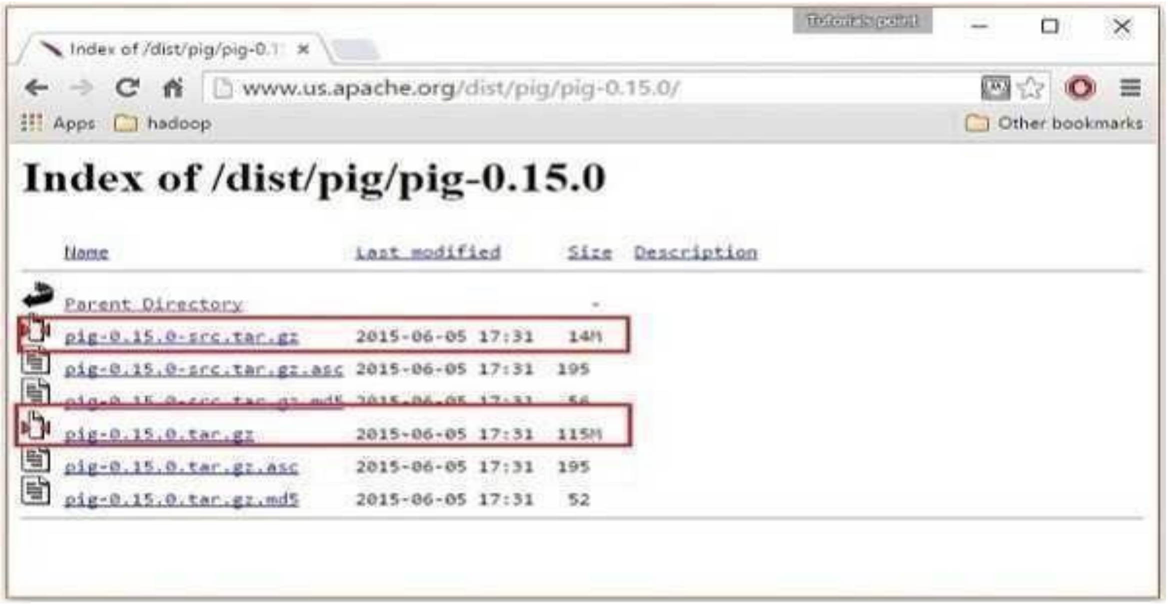
Please make sure you're downloading from a nearby mirror site, not from www.apache.org.

Older releases are available from the archives.

```
Name                Last modified      Size  Description

Parent Directory                        -
latest/             2015-08-19 03:09    -
pig-0.13.0/         2015-08-19 03:09    -
pig-0.14.0/         2015-08-19 03:09    -
pig-0.15.0/         2015-08-19 03:09    -
KEYS                2014-10-30 00:34   9.7K
```

Step 5

Within these folders, you will have the source and binary files of Apache Pig in various distributions. Download the tar files of the source and binary files of Apache Pig 0.15, pig0.15.0- src.tar.gz and pig-0.15.0.tar.gz.



Index of /dist/pig/pig-0.1

www.us.apache.org/dist/pig/pig-0.15.0/

Index of /dist/pig/pig-0.15.0

```
Name                         Last modified      Size  Description

Parent Directory                                 -
pig-0.15.0-src.tar.gz        2015-06-05 17:31   14M
pig-0.15.0-src.tar.gz.asc    2015-06-05 17:31   195
pig-0.15.0-src.tar.gz.md5    2015-06-05 17:31   56
pig-0.15.0.tar.gz            2015-06-05 17:31  115M
pig-0.15.0.tar.gz.asc        2015-06-05 17:31   195
pig-0.15.0.tar.gz.md5        2015-06-05 17:31   52
```

Install Apache Pig

After downloading the Apache Pig software, install it in your Linux environment by following the steps given below.

Step 1

Create a directory with the name Pig in the same directory where the installation directories of Hadoop, Java, and other software were installed. (In our tutorial, we have created the Pig directory in the user named Hadoop).

```
$ mkdir Pig
```

Step 2

Extract the downloaded tar files as shown below.

```
$ cd Downloads/
$ tar zxvf pig-0.15.0-src.tar.gz
$ tar zxvf pig-0.15.0.tar.gz
```

Step 3

Move the content of pig-0.15.0-src.tar.gz file to the Pig directory created earlier as shown below.

```
$ mv pig-0.15.0-src.tar.gz/* /home/Hadoop/Pig/
```

Configure Apache Pig
After installing Apache Pigs we have to configure it.To configure it two files
Bashrc and pig.properties.
.bashrc file
In The.bashrc files the following variables:

- PIG_HOME folder to the Apache Pig's installation folder,
- PATH environment variable to the bin folder,and
- PIG_CLASSPATH environment variable to the etc (configuration) folder of your Hadoop installations (the directory that contains the core-site.xml, hdfs-site.xml and mapred- site.xml files).

```
export PIG_HOME = /home/Hadoop/Pig
export PATH=$PATH:/home/Hadoop/pig/bin export PIG_CLASSPATH =$HADOOP_HOME/conf
```

pig.properties file
In the conf folder of Pig, we have a file named pig.properties. In the pig.properties file, you can set various parameters as given below.
pig -h properties
The following properties are supported−
Logging:

- verbose = true|false; default is false. This property is the same as-v switch brief=true|false;
default is false.
- This property is the same as -b switch debug=OFF|ERROR|WARN|INFO|DEBUG; default is INFO.
- This property is the same as -d switch aggregate.warning = true|false; default is true. If true, prints count of warnings of each type rather than logging each warning.

Performance tuning:

- pig.cachedbag.memusage=<mem fraction>; default is 0.2 (20% of all memory). Note that this memory is shared across all large bags used by the application.
- pig.skewed join.reduce.memusage=<mm fraction>; default is 0.3 (30% of all memory). Specifies the fraction of heap available for the reducer to perform the join.
- pig.exec.nocombiner = true|false; default is false.
  Only disable the combiner as a temporary workaround for problems. opt.multi query = true|false; multi query is on by default.
  Only disable multi query as a temporary workaround for problems. opt.fetch=true|false; fetch is on by default.
  Scripts containing Filter, Foreach, Limit, Stream, and Union can bedumped without MR jobs.
- pig.exec.mapPartAgg = true|false. Default is false.
  Determines if partial aggregation is done within map phase, before records are sent to the combiner.

- pig.exec.mapPartAgg.minReduction=<min aggregation factor>. Default is 10.
  If the in-map partial aggregation does not reduce the output num records by this factor, it gets disabled.

Miscellaneous:

- exectype = mapreduce|tez|local; default is mapreduce. This property is the same as -x switch
  pig.additional.jars.uris=<comma separated list of jars>. Used in place of register command.
  udf.import.list=<comma separated list of imports>. Used to avoid package names in UDF.
- stop.on.failure = true|false; default is false. Set to true to terminate on the First error.
- pig.datetime.default.tz=<UTC time offset>. e.g. +08:00. Default is the default timezone of the host.Determines the timezone used to handle datetime datatype and UDFs.
  Additionally, any Hadoop property can be specified.

Verifying the Installation

Verify the installation of Apache Pig by typing the version command. If the installation is successful, you will get the version of Apache Pig as shown below.

```
$ pig –version
Load dataset using SQOOP
mysql -u hadoop -p123456
create database demo; use demo;

create table emp
-> (eid int,
->esal int);
        insert into emp values(1,1000); insert into emp values(2,2000); insert into emp values(3,3000);
insert into emp values(4,4000); insert into emp values(5,5000); insert into emp values(6,6000); select *
fromemp;
+.........+.........+
| eid | esal |
+.........+.........+
|1 | 1000|
|2 | 2000|
|3 | 3000|
|4 | 4000|
|5 | 5000|
|6 | 6000|
+.........+.........+
...........
exit;
        sqoop import --connect jdbc:mysql://localhost:3306/demo --username hadoop --password
123456 -- table emp --target-dir /sqooptrf/emp -m 1
```

<u>Conclusion:</u>

Loading data with SQOOP and processing it with a PIG script showcases an efficient workflow for transforming and analyzing large datasets in a Hadoop environment.

Practical No – 8

Aim: Implement K-means clustering algorithm using map-Reduce/ Pyspark

The objective of this hands on is to let you reason about the parallelization of the K- Means clustering algorithm and use 2 platforms for implementing it: Spark and Hadoop.

Getting started with Spark

Start by launching Spark' python shell:

```
$ pyspark
```

K-means on Spark

We are going to use the machine learning module of Spark called MLlib designed to invoke machine learning algorithms on numerical data sets represented in RDD. By using RDD it is possible to interact with other components of Spark. When data is not numerical, MLlib requires additional data types like vectors generated using data transformation algorithms from text to numerical vectors (package pyspark.mllib).

MLlib implementation of k-means corresponds to the algorithm called K-Means\\5 which is a parallel version of the original one. The method header is defined as follows:

```
KMeans.train(k, maxIterations, initializationMode, runs)
```

- K: number of desired clusters
- maxIterations:the maximum number of iterations that the algorithm will perform. The more iterations the more precision in results but the execution time will increase.
- initializationMode: specifies the type of initialization of the algorithm.
- runs: number of times to execute the algorithm

Since K-means is not sure to find an optimum solution it can be executed many times on the same data set and the algorithm will return the best possible solution found in a given execution.

Creating an RDD

MLlib functions work on RDD so the first step is to create an RDD. We propose a simple file named example.txt with the following content:

```
0.0     0.0   0.0
0.1     0.1   0.1
0.2     0.2   0.2
9.0     9.0   9.0
9.1     9.1   9.1
9.2     9.2   9.2
```

Now load the data on Spark as an RDD object using the following command:

```
file ='hdfs://sandbox.hortonworks.com:8020/tmp/in/k/example.txt' data = sc.textFile(file)
```

You can test the content of the object RDD using the following commands:

```
def show(x):
print(x) data.foreach(show)
```

Convert text to numerical values As said before, MLlib algorithms use numbers as parameters.

In this example we convert the file example.txt to float:

```
from numpy import array parsedData = data.map(
lambda line: array([float(x) for x in line.split(' ')])
).cache() parsedData.foreach(show)
```

Training the algorithm

In order to create a model that can divide data into groups we need to import the package pyspark.mllib.clustering that contains the K-Means algorithm. Next we will create an instance of the object KMeans for grouping data into as many clusters as indicated by k.

```
from pyspark.mllib.clustering import KMeans
clusters = KMeans.train(parsedData, 2, maxIterations=10, runs=10,
initializationMode='random')
```

Evaluating the algorithm
Last step is to evaluate the obtained model and determine whether it is representative of the available data. Therefore, recall that the objective of the algorithm is to minimize the Euclidean distances among the points in every group. It is possible to consider the quadratic error generated by the algorithm known as Within Set Sum of Squared Error (WSSSE). It can be computed as follows:

```
from math import sqrt def error(point):
center =clusters.centers[clusters.predict(point)]
return sqrt(sum([x**2 for x in (point -center)]))
WSSSE = (parsedData.map(lambda point:error(point)).reduce(lambda x, y:x+y))
print('Within Set Sum of Squared Error = ' + str(WSSSE))
```

k-Means Clustering with MapReduceVector class:

```
public class Vector implements WritableComparable<Vector> {
private double[] vector; public Vector() {
super();
}
public Vector(Vector v){ super();
int l = v.vector.length; this.vector = new double[l];
System.arraycopy(v.vector, 0, this.vector, 0, l);
}
public Vector(double x, double y) { super();
this.vector = new double[] { x, y };
}
@Override
public void write(DataOutput out) throws IOException{ out.writeInt(vector.length);
for (int i = 0; i <vector.length;i++) out.writeDouble(vector[i]);
}
@Override
public void readFields(DataInput in) throws IOException{ int size = in.readInt();
vector = new double[size];
for (int i = 0; i < size; i++) vector[i] = in.readDouble();
}
@Override
publicintcompareTo(Vector o) {
boolean equals = true;
for (int i = 0; i <vector.length; i++) { int c = vector[i] - o.vector[i];
if (c != 0.0d) { return c;
}
return 0;
}
// get and set omitted
}
```

The distance measurement We need a measurement of a distance between two vectors, especially between a center and a vector . I've come up with the Manhattan distance because it doesn't require much computation overhead like square-rooting (Euclidean distance) and it is not too complex.

```
public static final double measureDistance(ClusterCenter center, Vector v){ double sum = 0;
int length = v.getVector().length; for (int i = 0; i < length; i++) {
sum += Math.abs(center.getCenter().getVector()[i]
- v.getVector()[i]);
}
return sum;
}
```

The Mapper

Let's assume that there is a list or a list-like sequencefile-iterating interface that is called centers. It contains ClusterCenter objects that represent the current centers. The DistanceMeasurer class contains the static method we defined in the last part.

```
// setup and cleanup stuffz omitted @Override
protected void map(ClusterCenter key, Vector value, Context Context) throws IOException,
InterruptedException {
ClusterCenter nearest = null;
doublenearestDistance = Double.MAX_VALUE; for (ClusterCenter c : centers) {
double dist = DistanceMeasurer.measureDistance(c, value);
if (nearest == null) { nearest = c; nearestDistance = dist;
} else {
if (nearestDistance>dist){ nearest = c; nearestDistance =dist;
}}}
context.write(nearest, value);}
```

The Reducer

Once again let's have a list or a list-like sequencefile-iterating interface that is called centers. Here we need it for storage reasons.

```
// setup and cleanup stuffz omitted once again @Override
protected void reduce(ClusterCenter key, Iterable<Vector>values, Context context) throws
IOException, InterruptedException {
Vector newCenter = new Vector();
List<Vector>vectorList = new LinkedList<Vector>(); intvectorSize
=key.getCenter().getVector().length; newCenter.setVector(new double[vectorSize]);
for (Vector value : values) { vectorList.add(new Vector(value));
for (int i = 0; i <value.getVector().length; i++) { newCenter.getVector()[i] +=
value.getVector()[i];
}
}
for (int i = 0; i <newCenter.getVector().length; i++){ newCenter.getVector()[i] =

newCenter.getVector()[i]
/ vectorList.size();
}
ClusterCenter center = newClusterCenter(newCenter); centers.add(center);
for (Vector vector : vectorList) { context.write(center, vector);
}
if (center.converged(key)) context.getCounter(Counter.CONVERGED).increment(1)
```

<u>Conclusion:</u>

Implementing the K-means clustering algorithm highlights the application of distributed computing frameworks like Map-Reduce and PySpark for scalable and efficient machine learning tasks.

Practical No – 9

Aim: Implement word count / frequency programs using MapReduce

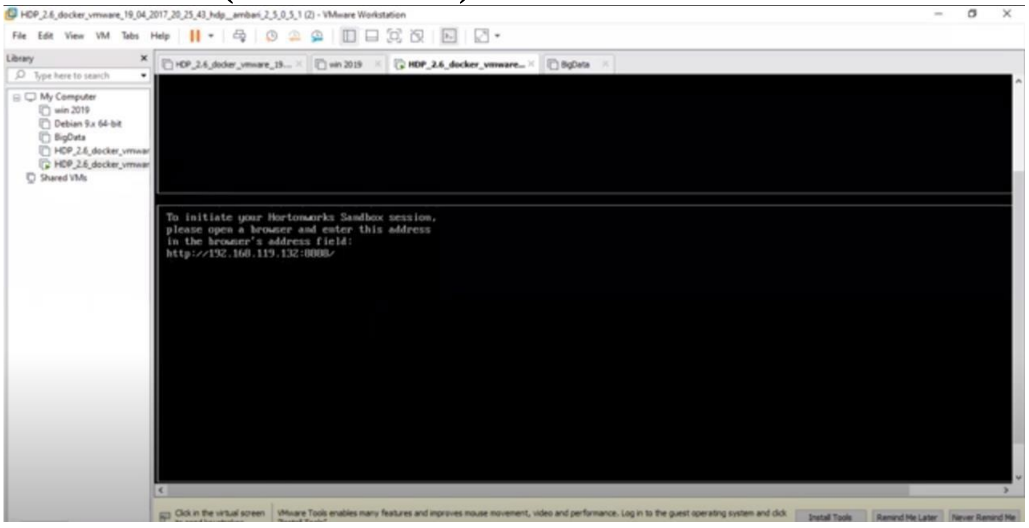
Map Reduce as two components Map and Reduce.
Java program:
Write Program Save as WordCount.java

```
//////////////////////
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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 {
 public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{
 private final static IntWritable one = new IntWritable(1);
 private Text word = new Text();
 public void map(Object key, Text value, Context context
 ) throws IOException, InterruptedException {
StringTokenizeritr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) {//"This is the output is the"
word.set(itr.nextToken());
context.write(word, one);
 } } }
 public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
 private IntWritable result = new IntWritable();
 public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException,
InterruptedException
 {//is,3
 int sum = 0;
 for (IntWritableval : 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, "word count");
job.setJarByClass(WordCount.class);
job.setMapperClass(TokenizerMapper.class);
job.setCombinerClass(IntSumReducer.class);
job.setReducerClass(IntSumReducer.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);}}
```

Output Text File:

Hello World

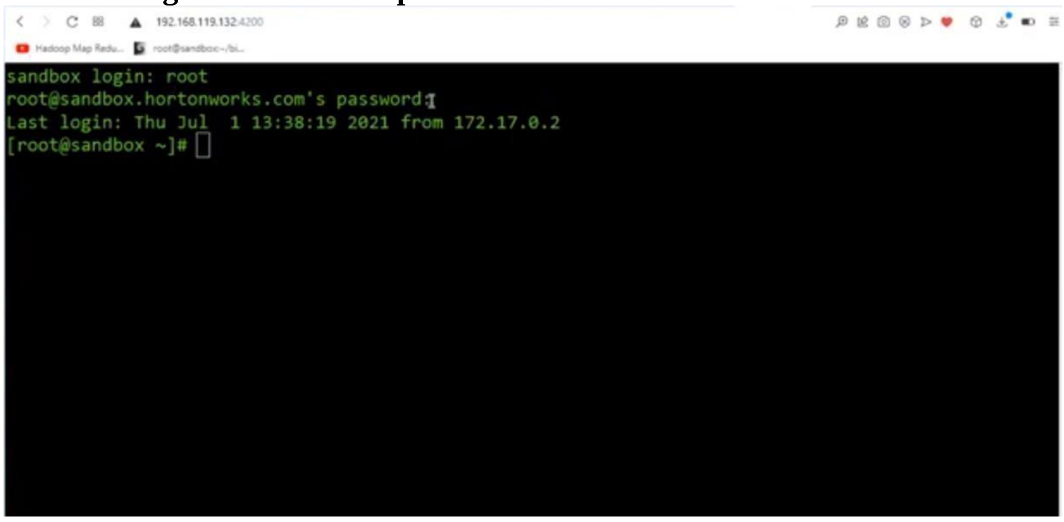
Start the server (Horton Sandbox)



```
To initiate your Hortonworks Sandbox session,
please open a browser and enter this address
in the browser's address field:
http://192.168.119.132:8888/
```

Open the terminal with 192.168.119.132/4200

Enter the login: root and the password and enter

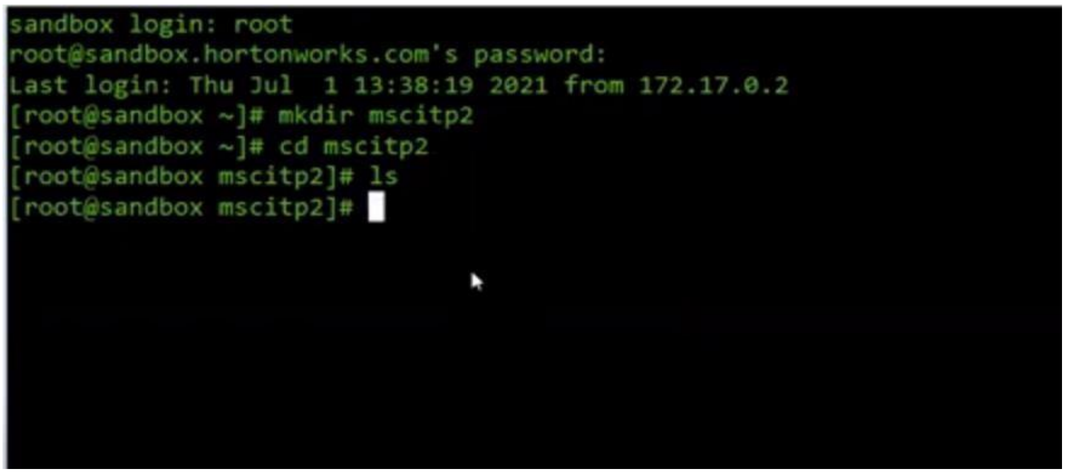


```
sandbox login: root
root@sandbox.hortonworks.com's password:
Last login: Thu Jul  1 13:38:19 2021 from 172.17.0.2
[root@sandbox ~]#
```

Create a folder in the local directory.

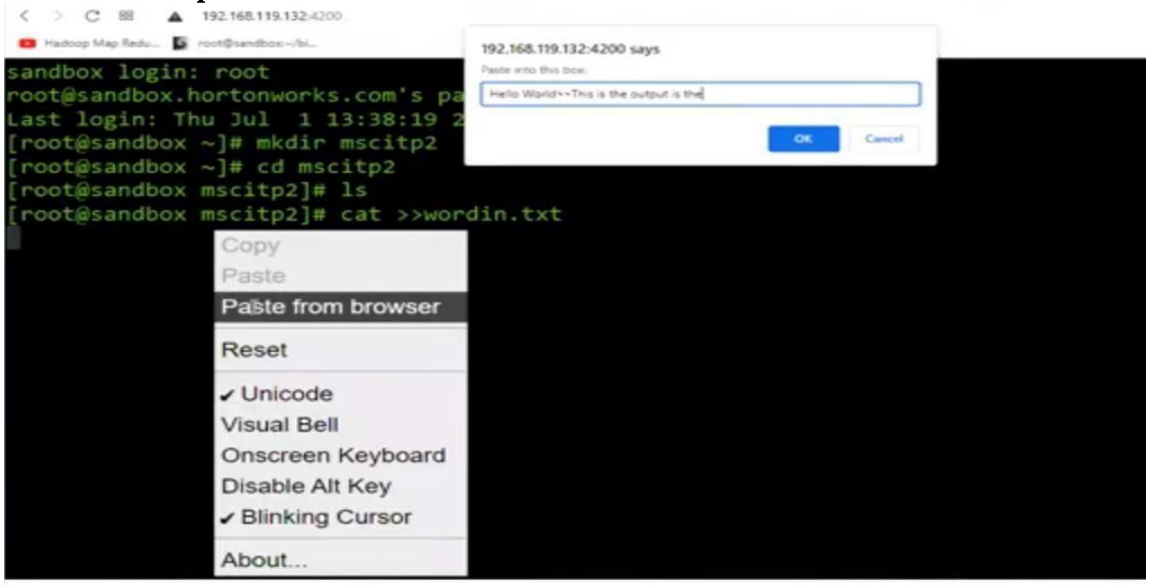
Command: mkdir mscitp2

Change the directory cd mcitp 2



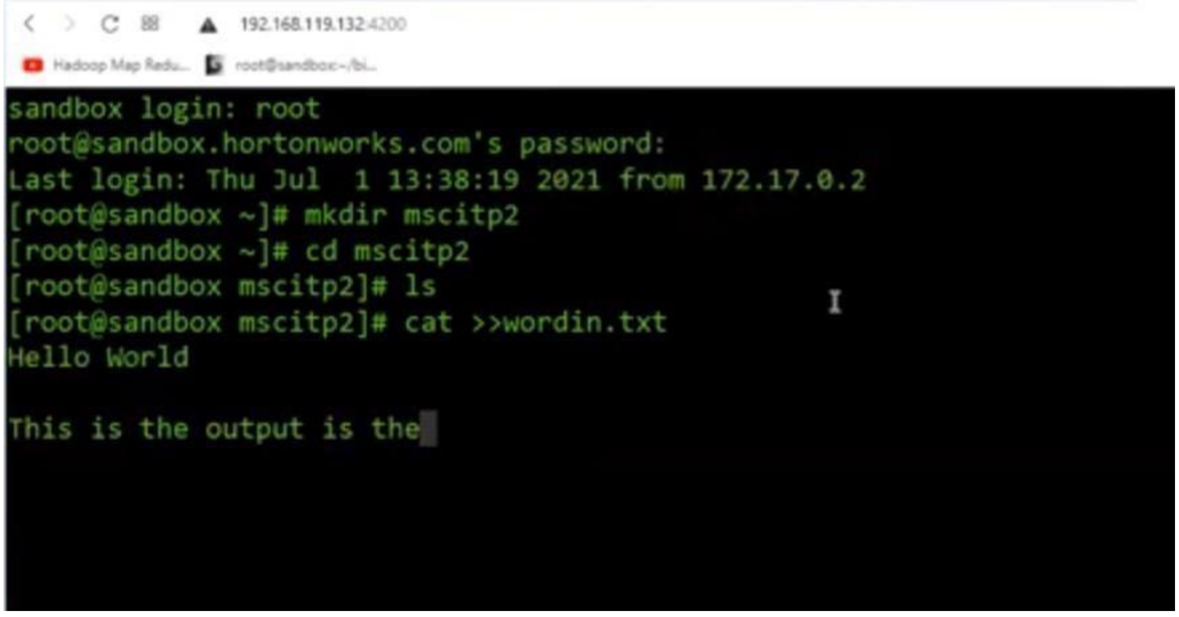
```
sandbox login: root
root@sandbox.hortonworks.com's password:
Last login: Thu Jul  1 13:38:19 2021 from 172.17.0.2
[root@sandbox ~]# mkdir mscitp2
[root@sandbox ~]# cd mscitp2
[root@sandbox mscitp2]# ls
[root@sandbox mscitp2]# 
```

Now create input file
Command: cat >> wordin.txt
Paste the text by right clicking on terminal
Hello World
This is the output is the



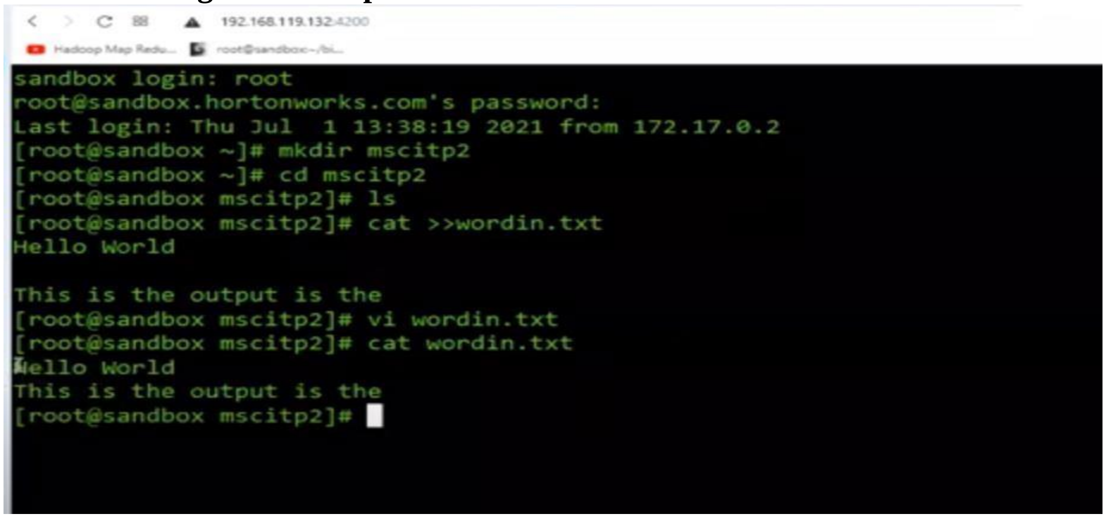
```
sandbox login: root
root@sandbox.hortonworks.com's pa
Last login: Thu Jul  1 13:38:19 2
[root@sandbox ~]# mkdir mscitp2
[root@sandbox ~]# cd mscitp2
[root@sandbox mscitp2]# ls
[root@sandbox mscitp2]# cat >>wordin.txt
```

192.168.119.132:4200 says
Paste into this box:
Hello World~~This is the output is the



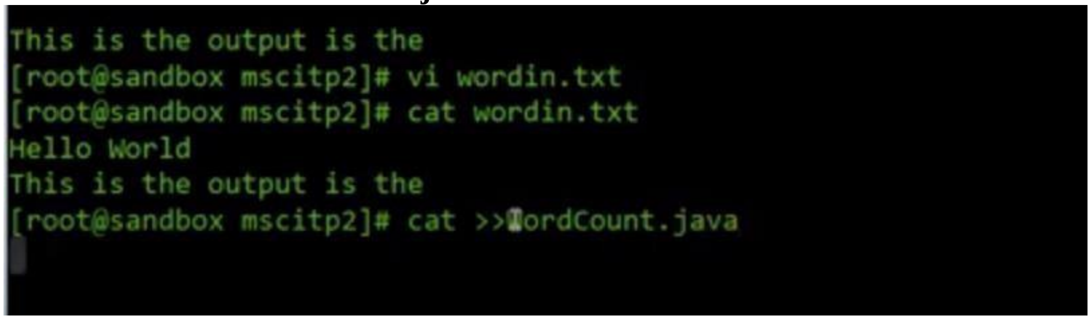
```
sandbox login: root
root@sandbox.hortonworks.com's password:
Last login: Thu Jul  1 13:38:19 2021 from 172.17.0.2
[root@sandbox ~]# mkdir mscitp2
[root@sandbox ~]# cd mscitp2
[root@sandbox mscitp2]# ls
[root@sandbox mscitp2]# cat >>wordin.txt
Hello World

This is the output is the
```

To remove the extra space type command vi wordin.txt
After removing the extra space check the content of the file cat wordin.txt



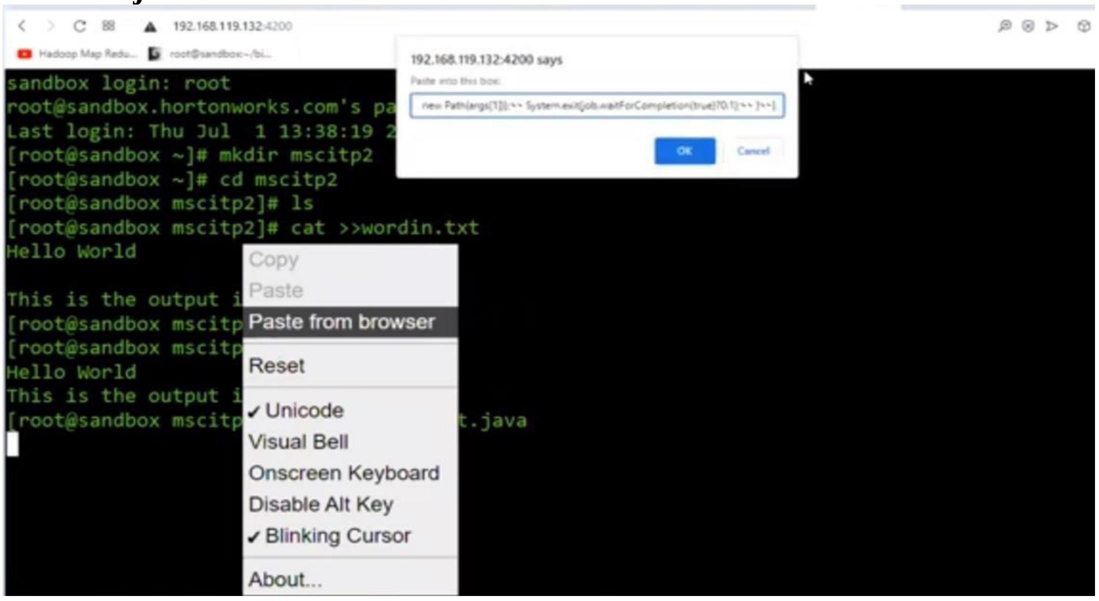
```
sandbox login: root
root@sandbox.hortonworks.com's password:
Last login: Thu Jul  1 13:38:19 2021 from 172.17.0.2
[root@sandbox ~]# mkdir mscitp2
[root@sandbox ~]# cd mscitp2
[root@sandbox mscitp2]# ls
[root@sandbox mscitp2]# cat >>wordin.txt
Hello World

This is the output is the
[root@sandbox mscitp2]# vi wordin.txt
[root@sandbox mscitp2]# cat wordin.txt
Hello World
This is the output is the
[root@sandbox mscitp2]#
```

Create another file wordcount.java



```
This is the output is the
[root@sandbox mscitp2]# vi wordin.txt
[root@sandbox mscitp2]# cat wordin.txt
Hello World
This is the output is the
[root@sandbox mscitp2]# cat >>WordCount.java
```

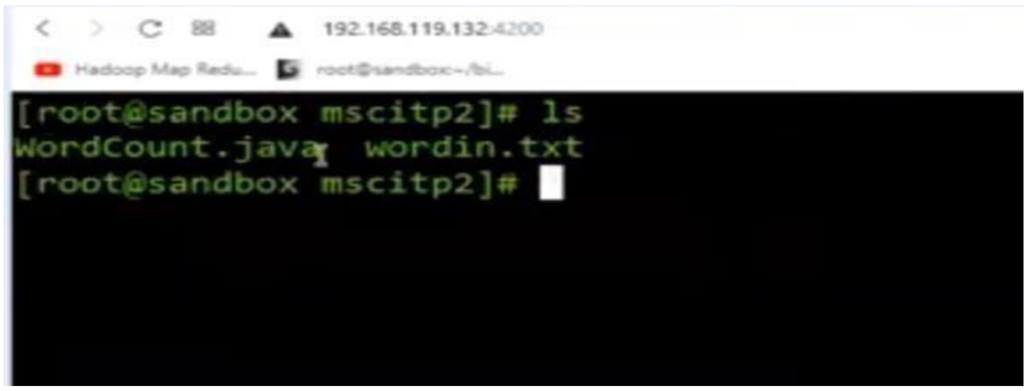
Paste the java code.



```
sandbox login: root
root@sandbox.hortonworks.com's pa
Last login: Thu Jul  1 13:38:19 2
[root@sandbox ~]# mkdir mscitp2
[root@sandbox ~]# cd mscitp2
[root@sandbox mscitp2]# ls
[root@sandbox mscitp2]# cat >>wordin.txt
Hello World

This is the output i
[root@sandbox mscitp
[root@sandbox mscitp
Hello World
This is the output i
[root@sandbox mscitp                    t.java
```

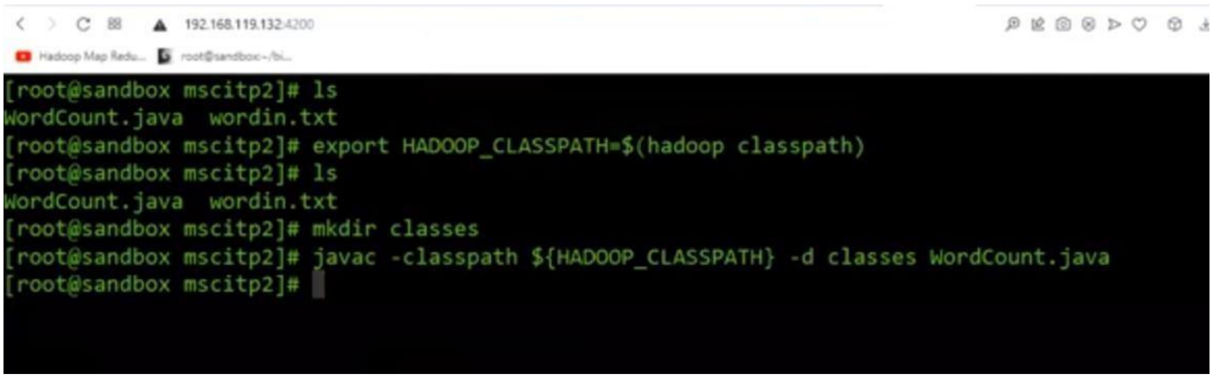
192.168.119.132:4200 says
Paste into this box:
new Path(args[1]);↵↵ System.exit(job.waitForCompletion(true)?0:1);↵↵ }↵↵}

Press control d to save the file
Check both the files create with command ls



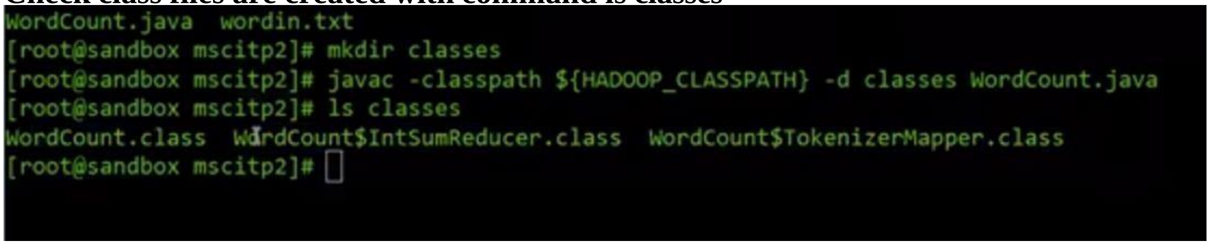
```
[root@sandbox mscitp2]# ls
WordCount.java  wordin.txt
[root@sandbox mscitp2]#
```

Now, to compile the java file
export HADOOP_CLASSPATH=$(hadoop classpath)
mkdir classes (To keep the compile files)
javac -classpath ${HADOOP_CLASSPATH} -d classes WordCount.java



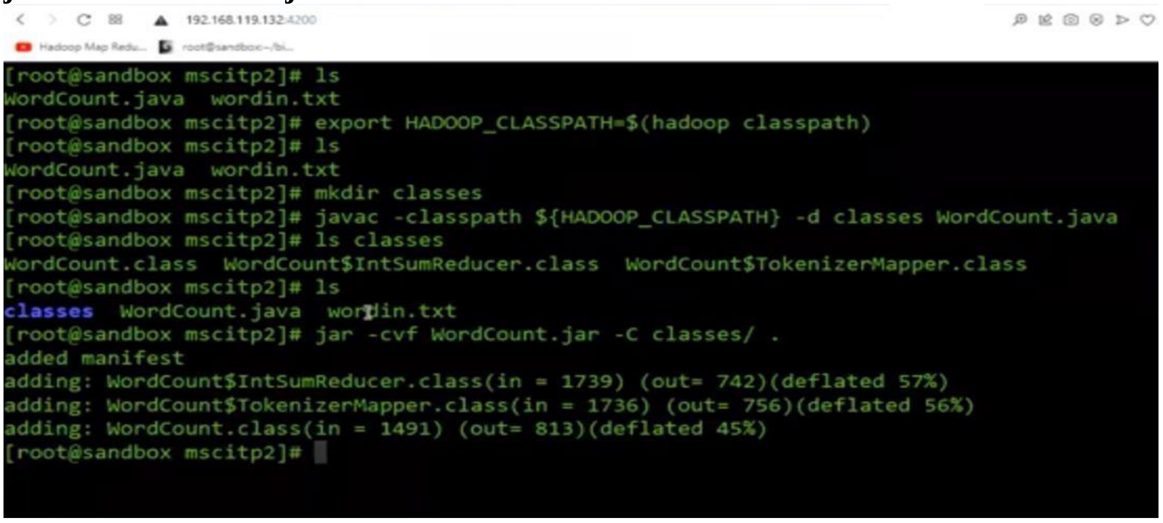
```
[root@sandbox mscitp2]# ls
WordCount.java  wordin.txt
[root@sandbox mscitp2]# export HADOOP_CLASSPATH=$(hadoop classpath)
[root@sandbox mscitp2]# ls
WordCount.java  wordin.txt
[root@sandbox mscitp2]# mkdir classes
[root@sandbox mscitp2]# javac -classpath ${HADOOP_CLASSPATH} -d classes WordCount.java
[root@sandbox mscitp2]#
```

Check class files are created with command ls classes



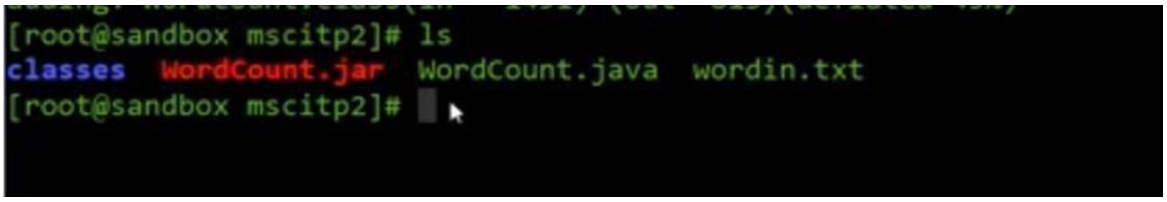
```
WordCount.java  wordin.txt
[root@sandbox mscitp2]# mkdir classes
[root@sandbox mscitp2]# javac -classpath ${HADOOP_CLASSPATH} -d classes WordCount.java
[root@sandbox mscitp2]# ls classes
WordCount.class  WordCount$IntSumReducer.class  WordCount$TokenizerMapper.class
[root@sandbox mscitp2]#
```

Now we have to bind all the class into single jar file with below command
jar -cvf WordCount.jar -C classes/ .



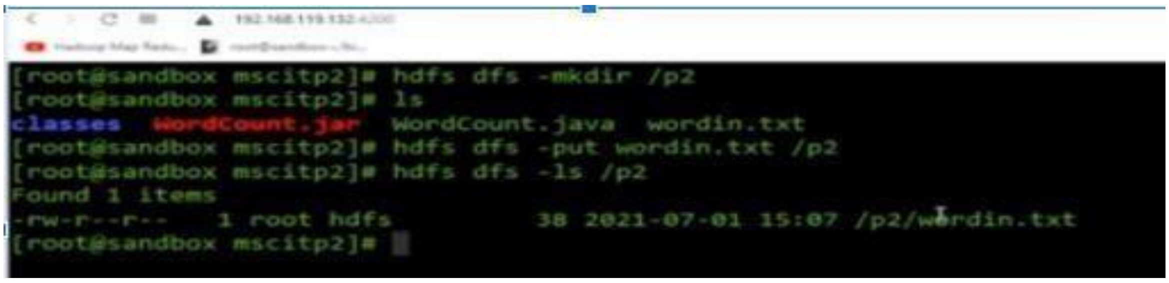
```
[root@sandbox mscitp2]# ls
WordCount.java  wordin.txt
[root@sandbox mscitp2]# export HADOOP_CLASSPATH=$(hadoop classpath)
[root@sandbox mscitp2]# ls
WordCount.java  wordin.txt
[root@sandbox mscitp2]# mkdir classes
[root@sandbox mscitp2]# javac -classpath ${HADOOP_CLASSPATH} -d classes WordCount.java
[root@sandbox mscitp2]# ls classes
WordCount.class  WordCount$IntSumReducer.class  WordCount$TokenizerMapper.class
[root@sandbox mscitp2]# ls
classes  WordCount.java  wordin.txt
[root@sandbox mscitp2]# jar -cvf WordCount.jar -C classes/ .
added manifest
adding: WordCount$IntSumReducer.class(in = 1739) (out= 742)(deflated 57%)
adding: WordCount$TokenizerMapper.class(in = 1736) (out= 756)(deflated 56%)
adding: WordCount.class(in = 1491) (out= 813)(deflated 45%)
[root@sandbox mscitp2]#
```

Run ls command we can see a jar file is created.



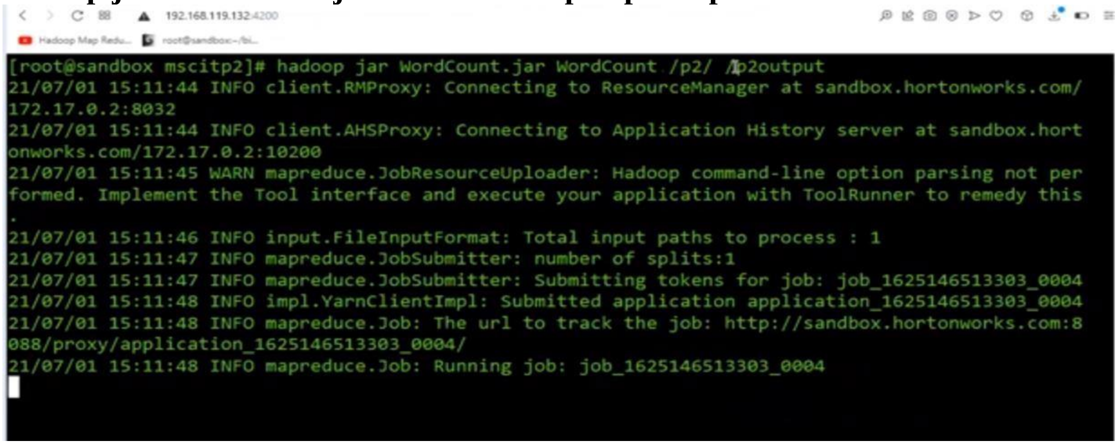
```
[root@sandbox mscitp2]# ls
classes  WordCount.jar  WordCount.java  wordin.txt
[root@sandbox mscitp2]#
```

wordin.txt should be present in the word directory of hdfs. So we need to upload wordin.txt file.



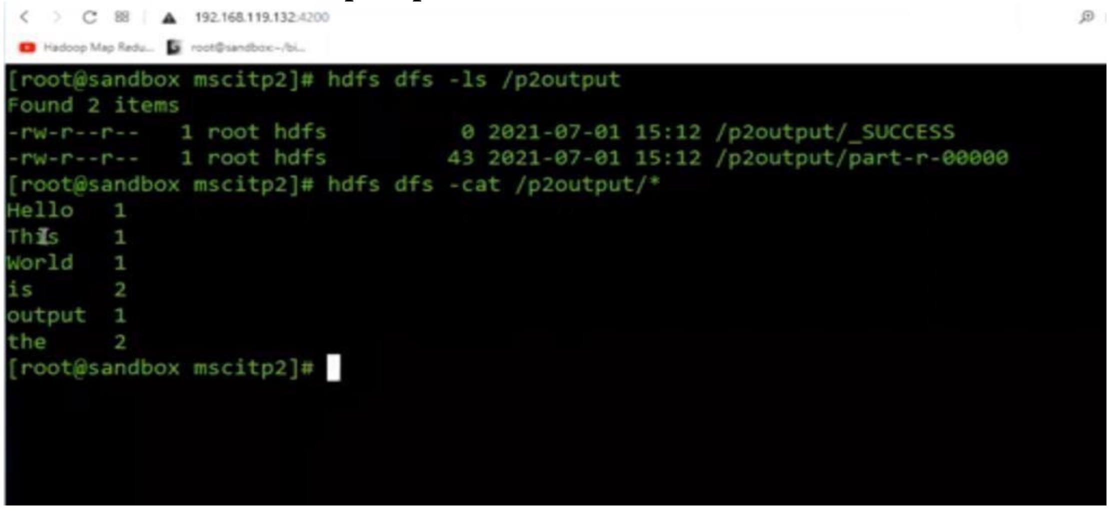
```
[root@sandbox mscitp2]# hdfs dfs -mkdir /p2
[root@sandbox mscitp2]# ls
classes  WordCount.jar  WordCount.java  wordin.txt
[root@sandbox mscitp2]# hdfs dfs -put wordin.txt /p2
[root@sandbox mscitp2]# hdfs dfs -ls /p2
Found 1 items
-rw-r--r--   1 root hdfs         38 2021-07-01 15:07 /p2/wordin.txt
[root@sandbox mscitp2]#
```

We need to put the final output p2 output.
hadoop jar WordCount.jar WordCount /p2/ /p2 output



```
[root@sandbox mscitp2]# hadoop jar WordCount.jar WordCount /p2/ /p2output
21/07/01 15:11:44 INFO client.RMProxy: Connecting to ResourceManager at sandbox.hortonworks.com/
172.17.0.2:8032
21/07/01 15:11:44 INFO client.AHSProxy: Connecting to Application History server at sandbox.hort
onworks.com/172.17.0.2:10200
21/07/01 15:11:45 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not per
formed. Implement the Tool interface and execute your application with ToolRunner to remedy this
.
21/07/01 15:11:46 INFO input.FileInputFormat: Total input paths to process : 1
21/07/01 15:11:47 INFO mapreduce.JobSubmitter: number of splits:1
21/07/01 15:11:47 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1625146513303_0004
21/07/01 15:11:48 INFO impl.YarnClientImpl: Submitted application application_1625146513303_0004
21/07/01 15:11:48 INFO mapreduce.Job: The url to track the job: http://sandbox.hortonworks.com:8
088/proxy/application_1625146513303_0004/
21/07/01 15:11:48 INFO mapreduce.Job: Running job: job_1625146513303_0004
```

Print the content of the output file
Command: hdfs dfs -cat /p2ouput/*



```
[root@sandbox mscitp2]# hdfs dfs -ls /p2output
Found 2 items
-rw-r--r--   1 root hdfs          0 2021-07-01 15:12 /p2output/_SUCCESS
-rw-r--r--   1 root hdfs         43 2021-07-01 15:12 /p2output/part-r-00000
[root@sandbox mscitp2]# hdfs dfs -cat /p2output/*
Hello   1
This    1
World   1
is      2
output  1
the     2
[root@sandbox mscitp2]#
```

Ctrl + l to clear the screen.
vi filename.txt= this command will create/ open filename.txt
two modes of vi editor

1) **Insert mode – i (press i key)**
2) **Command mode – esc key**

:wq is to save and exit

__**Conclusion:**__ Word count using MapReduce demonstrates basic distributed data processing.