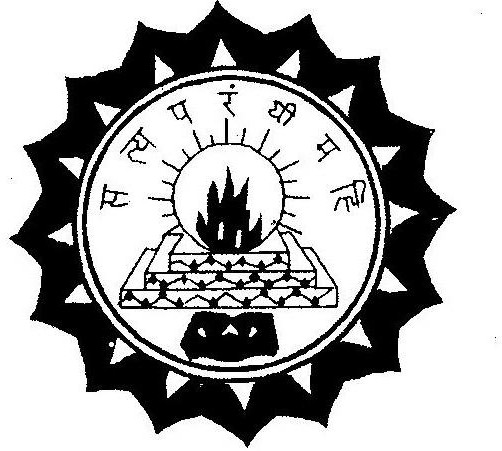
# DWARAKA DOSS GOVERDHAN DOSS VAISHNAV DAY COLLEGE (AUTONOMOUS)

**ARUMBAKKAM, CHENNAI - 600 106**



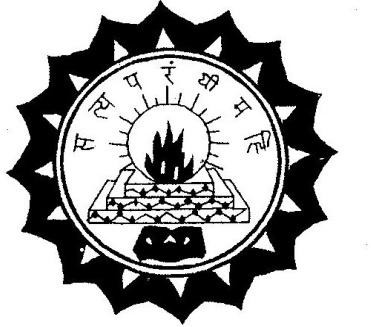
## PG DEPARTMENT OF COMPUTER SCIENCE

**III M.C.A**

**BIG DATA ANALYTICS LAB**

**DWARAKA DOSS GOVERDHAN DOSS VAISHNAV DAY COLLEGE (AUTONOMOUS)**

**ARUMBAKKAM, CHENNAI - 600106**



**PG DEPARTMENT OF COMPUTER SCIENCE BIG DATA ANALYTICS LAB**

Certified that this is a record work done by Mr./Ms………………….Whose Register No. is………………………of III MCA, during the academic year 2020- 2021.

## Faculty In-charge Head of the Department

Submitted for practical examination held on at Dwaraka

Doss Goverdhan Doss Vaishnav College, Chennai-106.

## Internal Examiner External Examiner

**Index**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ex No.** | **Date** | **Content** | **Page no.** | **Signature** |
| 1 |  | Performing Setup and Installing Hadoop | 4 |  |
| 2 |  | HDFS Basic Command Line File Operations | 15 |  |
| 3 |  | Run a Basic Wordcount Mapreduce Program | 18 |  |
| 4 |  | Map Reduce Program that Mines Weather Data | 21 |  |
| 5 |  | Matrix Multiplication | 26 |  |
| 6 |  | Pig Latin Installation | 31 |  |
| 7 |  | Hive Installation | 37 |  |
| 8 |  | Performing Pig Operations on Customer database | 43 |  |
| 9 |  | Performing Hive Query Operations on Electricity Bill database | 45 |  |
| 10 |  | Create Student database using hive | 48 |  |

|  |  |
| --- | --- |
| **EX.No:1** | **PERFORMING SETUP AND INSTALLING HADOOP** |
| **DATE:** |

**AIM:**

To perform the setup and installation of hadoop.

## SOURCE CODE:

1. **STANDALONE MODE:**

**Installation of jdk 7**

**Command:** sudo apt-get install openjdk-7-jdk Download and extract Hadoop

## Command:

wget <http://archive.apache.org/dist/hadoop/core/hadoop-1.2.0/hadoop-1.2.0.tar.gz>

**Command:** tar -xvf hadoop-1.2.0.tar.gz **Command:** sudo mv hadoop-1.2.0 /usr/lib/hadoop **Set the path for java and hadoop**

## Command:

sudo gedit $HOME/.bashrc

export JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-i386 export PATH=$PATH:$JAVA\_HOME/bin

export HADOOP\_COMMON\_HOME=/usr/lib/hadoop export HADOOP\_MAPRED\_HOME=/usr/lib/hadoop

export PATH=$PATH:$HADOOP\_COMMON\_HOME/bin export

PATH=$PATH:$HADOOP\_COMMON\_HOME/Sbin

**Checking of java and hadoop Command:** java -version

**Command:** hadoop version

## PSEUDO MODE:

Hadoop single node cluster runs on single machine. The namenodes and datanodes are performing on the one machine. The installation and configuration steps as given below:

## Installation of secured shell:

**Command:** sudo apt-get install openssh-server

**Create a ssh key for passwordless ssh configuration: Command:** ssh-keygen -t rsa –P "" **Moving the key to authorized key:**

**Command:** cat $HOME/.ssh/id\_rsa.pub >> $HOME/.ssh/authorized\_keys

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESTART THE COMPUTER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**Checking of secured shell login: Command:** ssh localhost

**Add JAVA\_HOME directory in hadoop-env.sh file: Command:** sudo gedit /usr/lib/hadoop/conf/hadoop-env.sh export JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-i386 **Creating namenode and datanode directories for hadoop:**

**Command:** sudo mkdir -p /usr/lib/hadoop/dfs/namenode **Command:** sudo mkdir -p /usr/lib/hadoop/dfs/datanode **Configure core-**

## site.xml:

**Command:**

sudo gedit /usr/lib/hadoop/conf/core-site.xml

<property>

<name>fs.default.name</name>

<value>hdfs://localhost:8020</value>

</property>

## Configure hdfs- site.xml:

**Command:**

sudo gedit /usr/lib/hadoop/conf/hdfs-site.xml

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.permissions</name>

<value>false</value>

</property>

<property>

<name>dfs.name.dir</name>

<value>/usr/lib/hadoop/dfs/namenode</value>

</property>

<property>

<name>dfs.data.dir</name>

<value>/usr/lib/hadoop/dfs/datanode</value>

</property> **Configure mapred- site.xml:**

## Command:

sudo gedit /usr/lib/hadoop/conf/mapred-site.xml

<property>

<name>mapred.job.tracker</name>

<value>localhost:8021</value>

</property>

## Format the name node:

**Command:** hadoop namenode -format

## Start the namenode datanode:

**Command:** start-dfs.sh

**Start the task tracker and job tracker: Command:** start-mapred.sh

## To check if Hadoop started correctly:

**Command:** jps namenode secondarynamenode datanode Jobtracker

## FULLY DISTRIBUTED MODE:

All the demons like name nodes and data nodes are runs on different machines. The data will replicate according to the replication factor in client machines. The secondary name node will store the mirror images of name node periodically. The name node having the metadata where the blocks are stored and number of replicas in the client machines. The slaves and master communicate each other periodically. The configurations of multimode cluster are given below:

**Configure the hosts in all nodes/machines: Command:** sudo gedit /etc/hosts/

192.168.1.58

pcetcse1 192.168.1.4

pcetcse2 192.168.1.5

pcetcse3 192.168.1.7

pcetcse4 192.168.1.8

pcetcse5

Passwordless Ssh Configuration

## Create ssh key on namenode/master.:

**Command:** ssh-keygen -t rsa -p “”

## Copy the generated public key all datanodes/slaves.:

**Command:** ssh-copy-id -i ~/.ssh/id\_rsa.pub huser@pcetcse2 **Command:** ssh-copy-id -i ~/.ssh/id\_rsa.pub huser@pcetcse3 **Command:** ssh-copy-id -i ~/.ssh/id\_rsa.pub huser@pcetcse4 **Command:** ssh-copy-id -i ~/.ssh/id\_rsa.pub huser@pcetcse5

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*RESTART ALL NODES/COMPUTERS/MACHINES \*\*\*\*\*\*\*\*\*\*\*\*/

**NOTE**: Verify the passwordless ssh environment from name node to all data nodes as “huser” user.

Login to master node **Command:** ssh pcetcse1 **Command:** ssh pcetcse2

**Command:** ssh pcetcse3 **Command:** ssh pcetcse4 **Command:** ssh pcetcse5

## Add JAVA\_HOME directory in hadoop-env.sh file in all nodes/machines:

**Command:**

sudo gedit /usr/lib/hadoop/conf/hadoop-env.sh export JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-i386

**Creating namenode directory in namenode/master: Command:** sudo mkdir -p /usr/lib/hadoop/dfs/namenode

**Creating namenode directory in datanonodes/slaves: Command:** sudo mkdir -p /usr/lib/hadoop/dfs/datanode

## Configure core-site.xml in all nodes/machines: Command:

sudo gedit /usr/lib/hadoop/conf/core-site.xml

<property>

<name>fs.default.name</name>

<value>hdfs://pcetcse1:8020</value>

</property

## Configure hdfs-site.xml in namenode/master: Command:

sudo gedit /usr/lib/hadoop/conf/hdfs-site.xml

<property>

<name>dfs.replication</name>

<value>3</value>

</property>

<property>

<name>dfs.permissions</name>

<value>false</value>

</property>

<property>

<name>dfs.name.dir</name>

<value>/usr/lib/hadoop/dfs/namenode</value>

</property>

## Configure hdfs-site.xml in datanodes/slaves: Command:

sudo gedit /usr/lib/hadoop/conf/hdfs-site.xml

<property>

<name>dfs.replication</name>

<value>3</value>

</property>

<property>

<name>dfs.permissions</name>

<value>false</value>

</property>

<property>

<name>dfs.data.dir</name>

<value>/usr/lib/hadoop/dfs/datanode</value>

</property>

## Configure mapred-site.xml in all nodes/machines: Command:

sudo gedit /usr/lib/hadoop/conf/mapred-site.xml

<property>

<name>mapred.job.tracker</name>

<value>pcetcse1:8021</value>

</property>

**Configure masters in all namenode/master give the secondary namenode hostname: Command:** sudo gedit /usr/lib/hadoop/conf/masters pcetcse2

**Configure masters in all datanodes/slaves give the namenode hostname: Command:** sudo gedit /usr/lib/hadoop/conf/masters pcetcse

## Configure slaves in all nodes/machines:

**Command:** sudo gedit /usr/lib/hadoop/conf/slaves pcetcse2

pcetcse3 pcetcse4 pcetcse5

## Format the name node:

**Command:** hadoop namenode -format

## Start the namenode, datanode:

**Command:** start-dfs.sh

## Start the task tracker and job tracker:

**Command:** start-mapred.sh

## To check if Hadoop started correctly check in all the nodes/machines: huser@pcetcse1:$ jps

namenode jobtracker **huser@pcetcse2:$** jps secondarynamenode tasktacker datanode **huser@pcetcse3:$** jps datanode

tasktracker **huser@pcetcse4:$** jps datanode

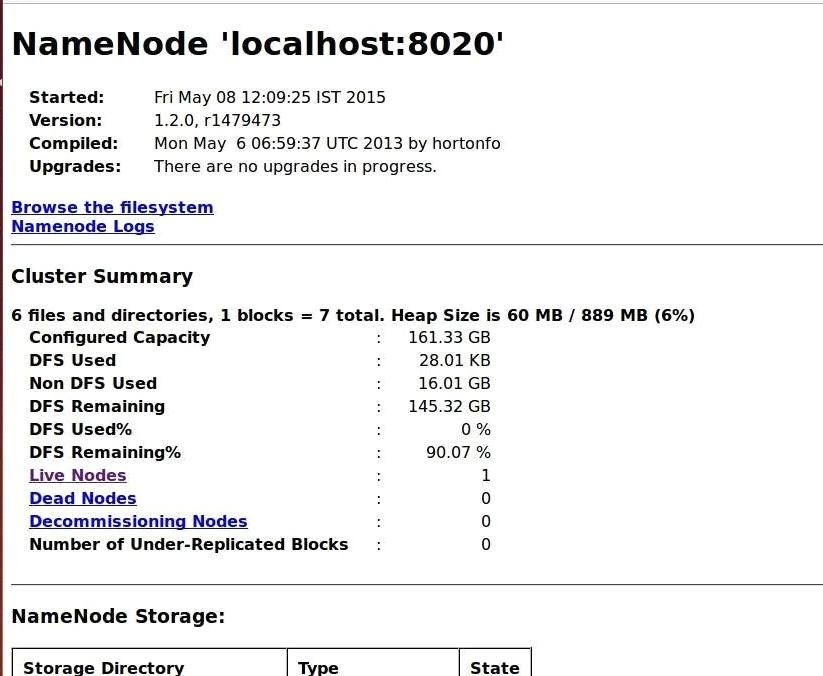
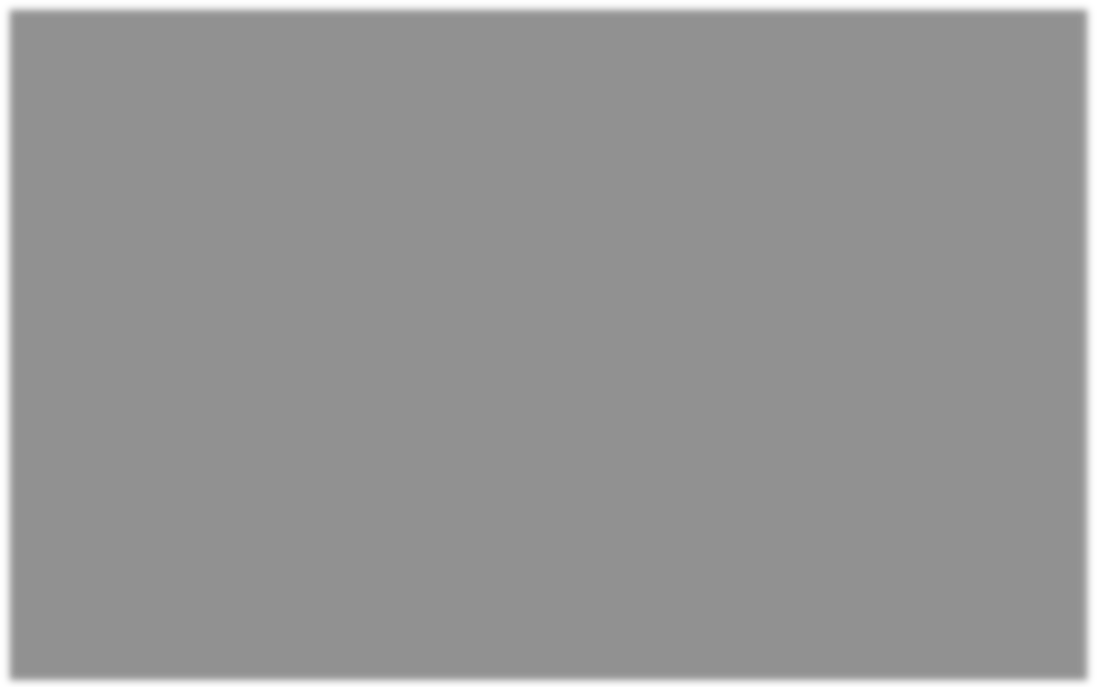
tasktracker **huser@pcetcse5:$** jps datanode

tasktracker

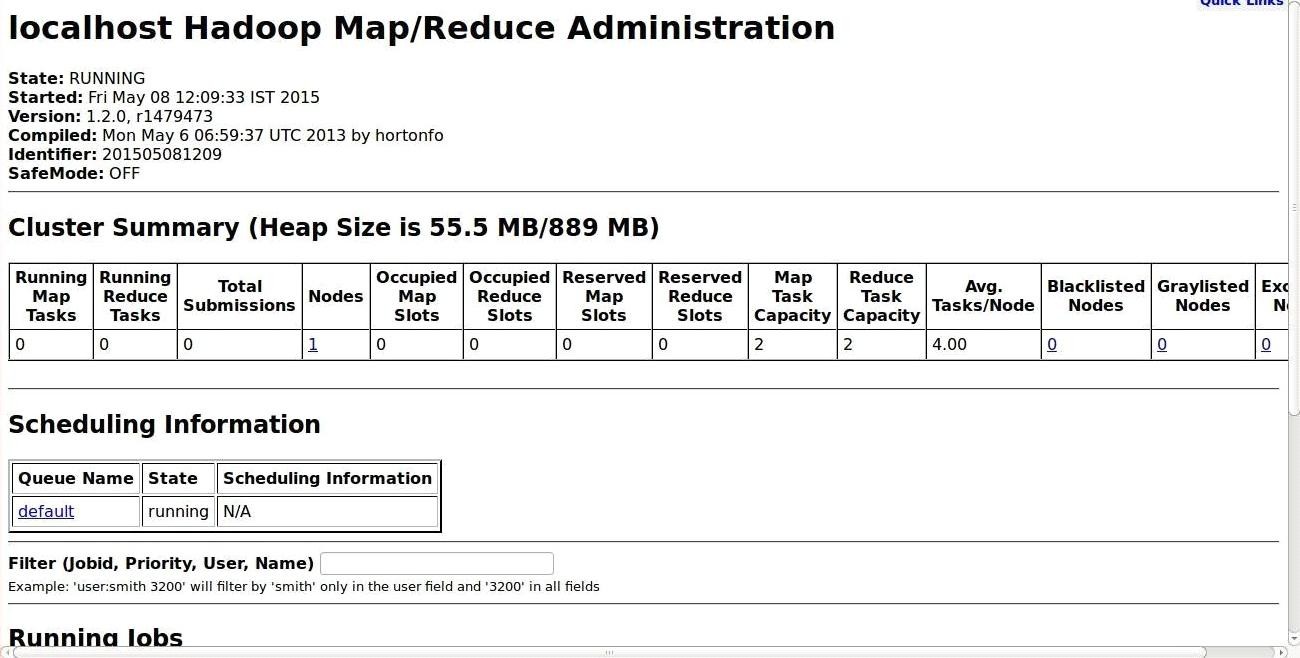
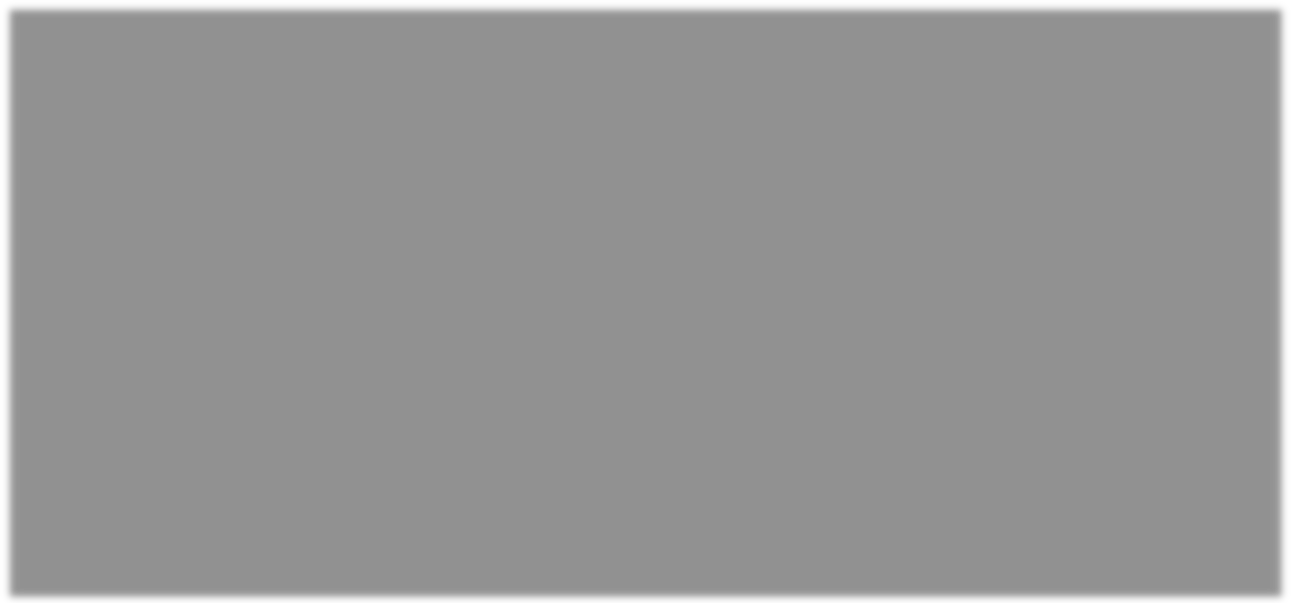
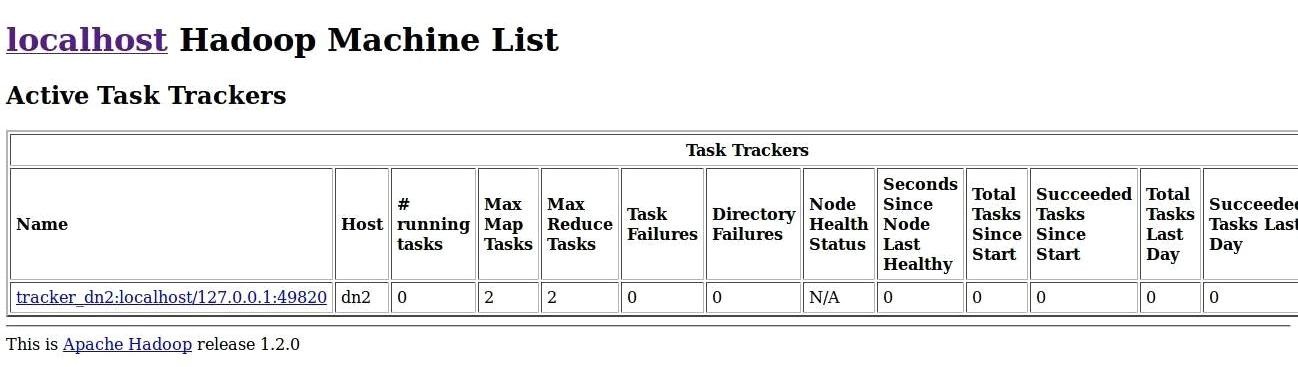
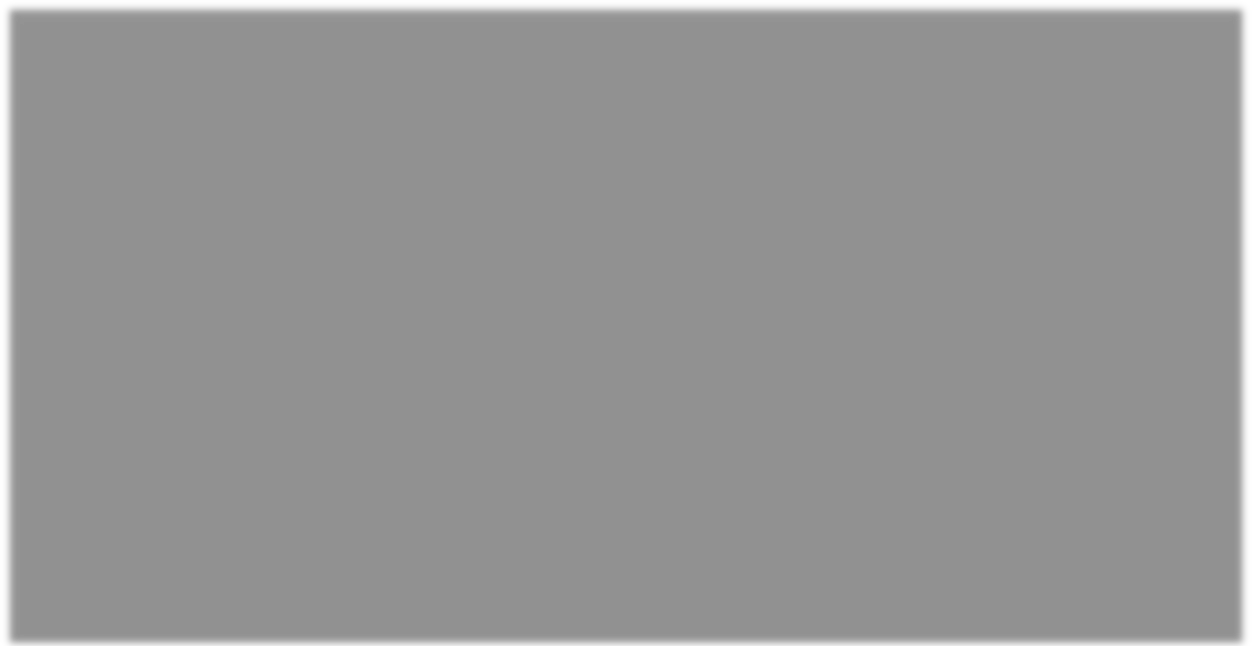
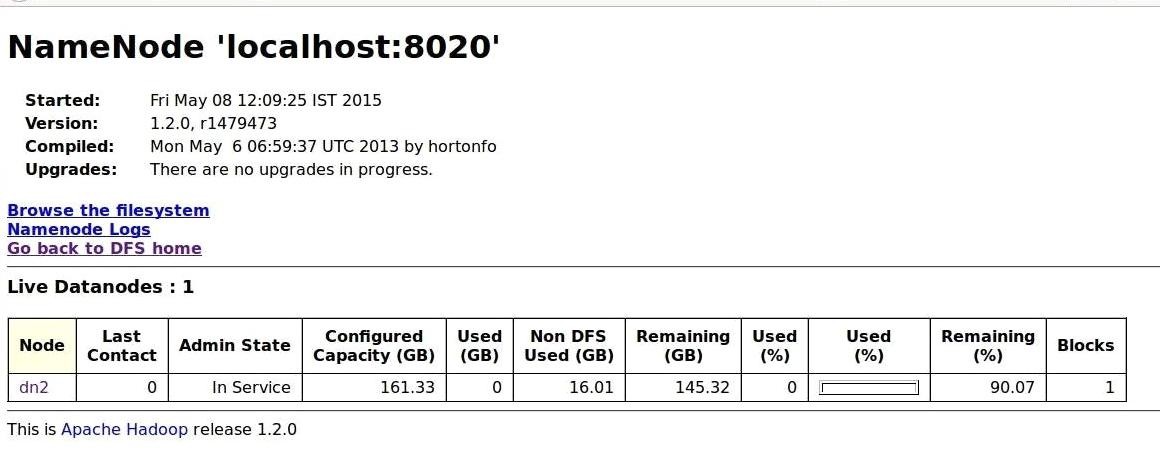
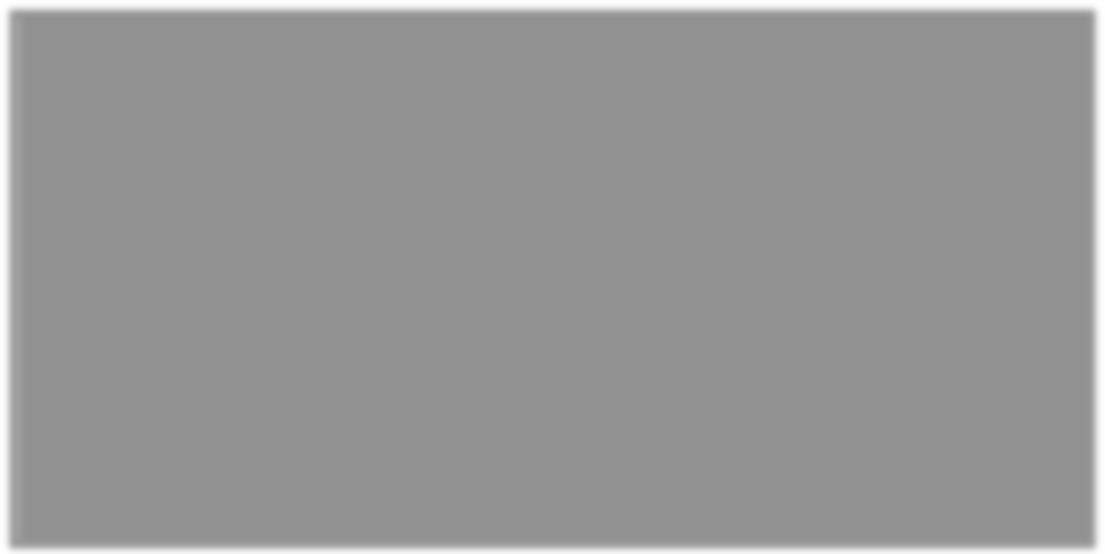
## OUTPUT:

**Using HDFS monitoring UI**

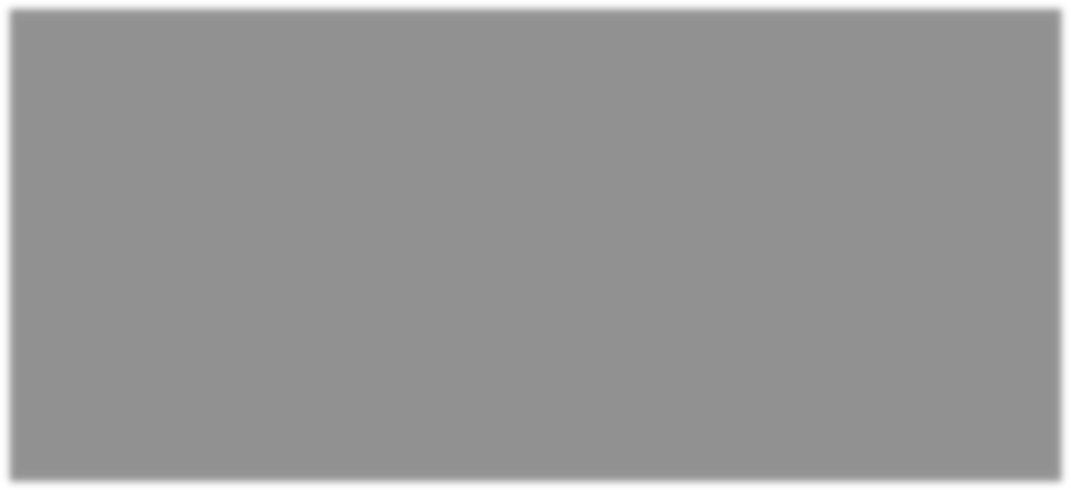
HDFS Namenode on UI http://locahost:50070/



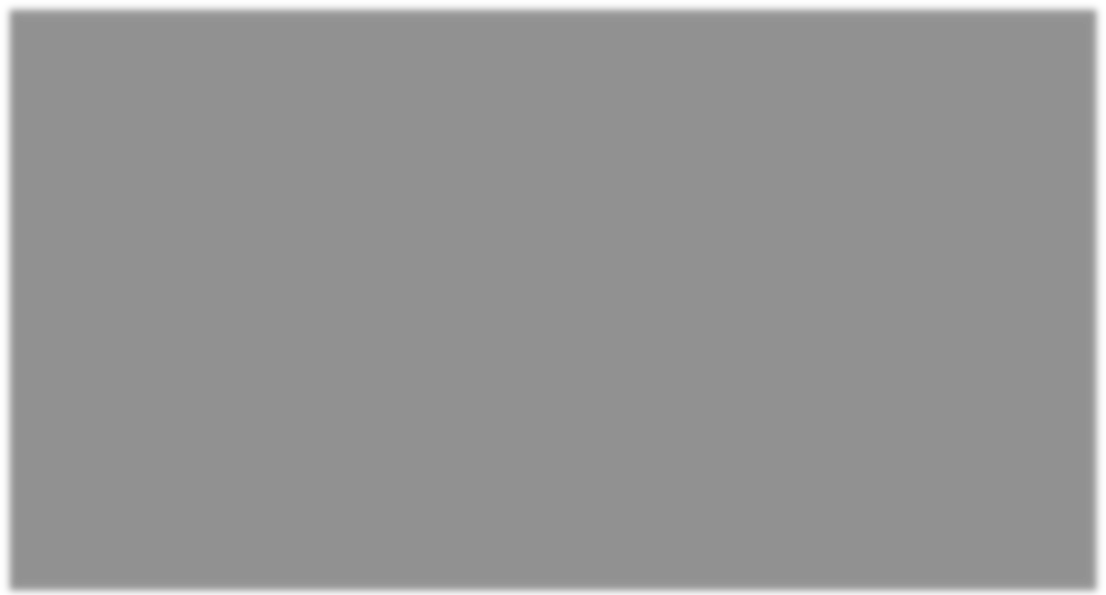
## HDFS Live Nodes list11



HDFS Logs



http://locahost:50070/logs/



## RESULT

Thus, the installation of the hadoop is done successfully.

|  |  |
| --- | --- |
| **EX.No:2** | **HDFS BASIC COMMAND LINE FILE OPERATIONS** |
| **DATE:** |

## AIM:

To perform basic HDFS command line file operations.

## SOURCE CODE:

1. **Create a directory in HDFS at given path(s): Command:** hadoop fs -mkdir <paths>
2. **List the contents of a directory: Command:** hadoop fs -ls <args>

## Upload and download a file in HDFS:

Upload: **Command:** hadoop fs -put <localsrc> <HDFS\_dest\_path> Download: **Command:** hadoop fs -get <HDFS\_src> <localdst>

## See contents of a file:

**Command:** hadoop fs -cat <path[filename]>

1. **Copy a file from source to destination: Command:** hadoop fs -cp <source>

<dest>

1. Copy a file from/To Local file system to HDFS:

**Command:** hadoop fs -copyFromLocal <localsrc> URI

**Command:** hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localsrc>

1. **Move file from source to destination: Command:** hadoop fs -mv <src> dest>

## Remove a file or directory in HDFS:

Remove files specified as argument. Delete directory only when it is empty.

**Command:** hadoop fs -rm <arg>Recursive version of delete

**Command:** hadoop fs -rmr <arg>

## Display last few lines of a file:

**Command:** hadoop fs –tail <path [filename]>

1. **Display the aggregate length of a file: Command:** hadoop fs -du <path>
2. **Getting help: Command:** hadoop fs - help

## Adding files and directories: Creating a directory:

**Command:** hadoop fs -mkdir input/

1. **Copying the files from localfile system to HDFS: Command:** hadoop fs -put inp/file01 input/

## Retrieving files:

**Command:** hadoop fs -get input/file01 localfs

1. **Deleting files and directories: Command:** hadoop fs -rmr input/file01

## RESULT

Thus, the basic HDFS command line file operations are done successfully.

|  |  |
| --- | --- |
| **EX.No:3** | **RUN A BASIC WORD COUNT MAP REDUCE PROGRAM** |
| **DATE:** |

## AIM:

To run a basic word count map reduce program to understand map reduce paradigm.

## SOURCE CODE:

**Textfile: Ww:**

hello hi

hi

# java file:

package wcc;

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 wc {

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

{

StringTokenizer itr = new StringTokenizer(value.toString()); while (itr.hasMoreTokens())

{ 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

{

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, "word count"); job.setJarByClass(wc.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);

}

}

## Hdfs:

**Create the temporary content file in the input directory:**

[cloudera@quickstart ~]$ **hadoop fs -mkdir w Put the file.txt into hdfs:**

[cloudera@quickstart ~]$ **hadoop fs -put /home/cloudera/Desktop/ww w/**

[cloudera@quickstart ~]$ **hadoop fs -ls w/**

Found 1 items

-rw-r--r-- 1 cloudera cloudera 13 2019-10-09 21:44 w/ww

[cloudera@quickstart ~]$ **hadoop fs -cat w/ww**

hello hi

hi

## Run WordCount jar file on input directory:

[cloudera@quickstart **~**]$ **hadoop jar w1.jar wcc.wc w/ww output**

19/10/09 21:47:14 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032

## To see the output:

[cloudera@quickstart ~]$ **hadoop fs -ls output/**

Found 2 items

-rw-r--r-- 1 cloudera cloudera 0 2019-10-09 21:47 output/\_SUCCESS

-rw-r--r-- 1 cloudera cloudera 13 2019-10-09 21:47 output/part-r-00000

## OUTPUT:

[cloudera@quickstart ~]$ **hadoop fs -cat output/part-r-00000**

hello 1

hi 2

## RESULT:

Thus, the words are counted using map reduce.

|  |  |
| --- | --- |
| **EX.No:4** | **MAP REDUCE PROGRAM THAT DETERMINES WEATHER DATA** |
| **DATE:** |

## AIM:

Weather sensors collecting data every hour at many locations across the globe gather

a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semi structured and record-oriented.

## SOURCE CODE:

import java.io.IOException; import java.util.Iterator;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat; import org.apache.hadoop.mapreduce.lib.input.TextInputFormat; import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper; import org.apache.hadoop.mapreduce.Reducer; import

org.apache.hadoop.conf.Configuration; public class MyMaxMin {

public static class MaxTemperatureMapper extends Mapper<LongWritable, Text, Text, Text>

{

@Override

public void map(LongWritable arg0, Text Value, Context context) throws IOException, InterruptedException {

String line = Value.toString(); if (!(line.length() == 0))

{

String date = line.substring(6, 14);

float temp\_Min = Float.parseFloat(line.substring(22, 28).trim()); float temp\_Max = Float.parseFloat(line.substring(32, 36).trim()); if (temp\_Max > 35.0)

{

context.write(new Text("Hot Day " + date),new Text(String.valueOf(temp\_Max)));

}

if (temp\_Min < 10) {

context.write(new Text("Cold Day " + date),new Text(String.valueOf(temp\_Min)))

}

}

}

}

public static class MaxTemperatureReducer extends Reducer<Text, Text, Text, Text> { public void reduce(Text Key, Iterator<Text> Values, Context context)throws IOException, InterruptedException {

String temperature = Values.next().toString(); context.write(Key, new Text(temperature));

}

}

public static void main(String[] args) throws Exception { Configuration conf = new Configuration(); Job job = new Job(conf, "weather example"); job.setJarByClass(MyMaxMin.class); job.setMapOutputKeyClass(Text.class); job.setMapOutputValueClass(Text.class);

job.setMapperClass(MaxTemperatureMapper.class); job.setReducerClass(MaxTemperatureReducer.class); job.setInputFormatClass(TextInputFormat.class); job.setOutputFormatClass(TextOutputFormat.class); Path OutputPath = new Path(args[1]); FileInputFormat.addInputPath(job, new Path(args[0])); FileOutputFormat.setOutputPath(job, new Path(args[1])); System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

## In hdfs environment:

**Create the temporary content file in the input directory:**

[cloudera@quickstart ~]$ **hadoop fs -mkdir weather\_dir**

## Put the file.txt into hdfs:

[cloudera@quickstart ~]$ **hadoop fs -put /home/cloudera/Desktop/wd.txt weather\_dir/**

[cloudera@quickstart ~]$ **hadoop fs -ls weather\_dir/**

Found 1 items

-rw-r--r-- 1 cloudera cloudera 41881 2019-10-09 22:16 weather\_dir/wd.txt

## To see the content of the file:

[cloudera@quickstart ~]$ **hadoop fs -cat weather\_dir/wd.txt**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 23907 20150101 | 2.423 -98.08 | 30.62 2.2 -0.6 0.8 0.9 | 6.2 | 1.47 C 3.7 1.1 |
| 2.5 99.9 85.4 | 97.2 0.369 | 0.308 -99.000 -99.000 -99.000 | 7.0 | 8.1 -9999.0 -9999.0 - |
| 9999.0 |  |  |  |  |
| 23907 20150102 | 2.423 -98.08 | 30.62 3.5 1.3 2.4 2.2 | 9.0 | 1.43 C 4.9 2.3 |
| 3.1 100.0 98.8 | 99.8 0.391 | 0.327 -99.000 -99.000 -99.000 | 7.1 | 7.9 -9999.0 -9999.0 - |
| 9999.0 |  |  |  |  |

23907 20150103 2.423 -98.08 30.62 15.9 2.3 9.1 7.5 2.9 11.00 C 16.4 2.9

7.3 100.0 34.8 73.7 0.450 0.397 -99.000 -99.000 -99.000 7.6 7.9 -9999.0 -9999.0 -

9999.0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 23907 20150104 | 2.423 | -98.08 | 30.62 9.2 -1.3 3.9 4.2 | 0.0 | 13.24 C 12.4 -0.5 |
| 4.9 82.0 40.6 | 61.7 | 0.414 | 0.352 -99.000 -99.000 -99.000 | 7.3 | 7.9 -9999.0 -9999.0 - |
| 9999.0 |  |  |  |  |  |
| 23907 20150105 | 2.423 | -98.08 | 30.62 10.9 -3.7 3.6 2.6 | 0.0 | 13.37 C 14.7 -3.0 |
| 3.8 77.9 33.3 | 57.4 | 0.399 | 0.340 -99.000 -99.000 -99.000 | 6.3 | 7.0 -9999.0 -9999.0 - |
| 9999.0 |  |  |  |  |  |
| 23907 20150106 | 2.423 | -98.08 | 30.62 20.2 2.9 11.6 10.9 | 0.0 | 12.90 C 22.0 1.6 |
| 9.9 67.7 30.2 | 49.3 | 0.395 | 0.335 -99.000 -99.000 -99.000 | 8.0 | 8.0 -9999.0 -9999.0 - |
| 9999.0 |  |  |  |  |  |

23907 20150107 2.423 -98.08 30.62 10.9 -3.4 3.8 4.5 0.0 12.68 C 12.4 -2.1

5.5 82.7 36.5

## Run wet jar file on input directory:

[cloudera@quickstart ~]$ **hadoop jar wet.jar weather.weat weather\_dir/wd.txt out**

## To see the output:

[cloudera@quickstart ~]$ **hadoop fs -ls out/**

Found 2 items

-rw-r--r-- 1 cloudera cloudera 0 2019-10-09 22:20 out/\_SUCCESS

-rw-r--r-- 1 cloudera cloudera 4632 2019-10-09 22:20 out/part-r-00000 [cloudera@quickstart ~]$ **hadoop fs -cat out/part-r-00000**

|  |  |
| --- | --- |
| Cold Day 20150101 | -98.0 |
| Cold Day 20150102 | -98.0 |
| Cold Day 20150103 | -98.0 |
| Cold Day 20150104 | -98.0 |
| Cold Day 20150105 | -98.0 |
| Cold Day 20150106 | -98.0 |
| Cold Day 20150107 | -98.0 |
| Cold Day 20150108 | -98.0 |
| Cold Day 20150109 | -98.0 |
| Cold Day 20150110 | -98.0 |
| Cold Day 20150111 | -98.0 |
| Cold Day 20150112 | -98.0 |
| Cold Day 20150113 | -98.0 |
| Cold Day 20150114 | -98.0 |
| Cold Day 20150115 | -98.0 |
| Cold Day 20150116 | -98.0 |
| Cold Day 20150117 | -98.0 |
| Cold Day 20150118 | -98.0 |
| Cold Day 20150119 | -98.0 |
| Cold Day 20150120 | -98.0 |
| Cold Day 20150121 | -98.0 |
| Cold Day 20150122 | -98.0 |
| Cold Day 20150123 | -98.0 |
| Cold Day 20150124 | -98.0 |
| Cold Day 20150125 | -98.0 |
| Cold Day 20150126 | -98.0 |
| Cold Day 20150127 | -98.0 |
| Cold Day 20150128 | -98.0 |
| Cold Day 20150129 | -98.0 |
| Cold Day 20150130 | -98.0 |
| Cold Day 20150131 | -98.0 |
| Cold Day 20150201 | -98.0 |
| Cold Day 20150202 | -98.0 |
| Cold Day 20150203 | -98.0 |

|  |  |
| --- | --- |
| Cold Day 20150204 | -98.0 |
| Cold Day 20150205 | -98.0 |
| Cold Day 20150206 | -98.0 |
| Cold Day 20150207 | -98.0 |
| Cold Day 20150208 | -98.0 |
| Cold Day 20150209 | -98.0 |
| Cold Day 20150210 | -98.0 |
| Cold Day 20150211 | -98.0 |
| Cold Day 20150212 | -98.0 |
| Cold Day 20150213 | -98.0 |
| Cold Day 20150214 | -98.0 |
| Cold Day 20150215 | -98.0 |
| Cold Day 20150216 | -98.0 |
| Cold Day 20150217 | -98.0 |
| Cold Day 20150218 | -98.0 |
| Cold Day 20150219 | -98.0 |
| Cold Day 20150220 | -98.0 |
| Cold Day 20150221 | -98.0 |
| Cold Day 20150222 | -98.0 |
| Cold Day 20150223 | -98.0 |

## RESULT:

Thus, the map reduce is successfully done in the given sample.

|  |  |
| --- | --- |
| **EX.No:5** | **Matrix Multiplication** |
| **DATE:** |

## AIM:

Implement Matrix Multiplication with Hadoop Map Reduce.

## SOURCE CODE:

import java.io.IOException; import java.util.\*;

import org.apache.hadoop.fs.Path; import org.apache.hadoop.conf.\*; import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapreduce.\*;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; importorg.apache.hadoop.mapreduce.lib.input.TextInputFormat; importorg.apache.hadoop.mapreduce.lib.output.FileOutputFormat; importorg.apache.hadoop.mapreduce.lib.output.TextOutputFor mat; public class MatrixMul {

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Mapper class\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ public static class Map extends Mapper<LongWritable, Text, Text, Text> {

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

Configuration conf = context.getConfiguration(); intm=Integer.parseInt(conf.get("m"));

int p =Integer.parseInt(conf.get("p"));

String line = value.toString();

String[] indicesAndValue = line.split(","); Text outputKey = new Text();

Text outputValue = new Text();

if (indicesAndValue[0].equals("A")) { for (int k = 0; k < p; k++)

{

}

} else { outputKey.set(indicesAndValue[1] + "," + k);

outputValue.set("A," + indicesAndValue[2] + "," + indicesAndValue[3]);

context.write(outputKey, outputValue); for (int i = 0; i < m; i++) {

outputKey.set(i + "," + indicesAndValue[2]);

outputValue.set("B," + indicesAndValue[1] + "," + indicesAndValue[3]); context.write(outputKey, outputValue);

}

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Reducer Class\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ public static class Reduce extends Reducer<Text, Text, Text, Text> {

public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

String[]value;

HashMap<Integer, Float> hashA = new HashMap<Integer, Float>(); HashMap<Integer, Float> hashB = new HashMap<Integer, Float>(); for (Text val : values) {

value = val.toString().split(",");

if (value[0].equals("A")) {

hashA.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));

} else {

hashB.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));

}

}

double[] myList = new double[10];

int n = Integer.parseInt(context.getConfiguration().get("n")); float result = 0.0f;

float a\_ij; float b\_jk;

for (int j = 0; j < n; j++) {

a\_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;

b`\_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f; result += a\_ij \* b\_jk;

}

if (result != 0.0f) {

context.write(null, new Text(key.toString() + "," + Float.toString(result)));

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Driver(main) function\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

// A is an m-by-n matrix; B is an n-by-p matrix. conf.set("m", "8");

conf.set("n", "8");

conf.set("p", "8");

Job job =Job.getInstance(conf,"MatrixMultiplication"); job.setJarByClass(MatrixMul.class); job.setOutputKeyClass(Text.class); job.setOutputValueClass(Text.class); job.setMapperClass(Map.class); job.setReducerClass(Reduce.class); job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class); FileInputFormat.addInputPath(job, new Path(args[0])); FileOutputFormat.setOutputPath(job, new Path(args[1])); job.submit();

}

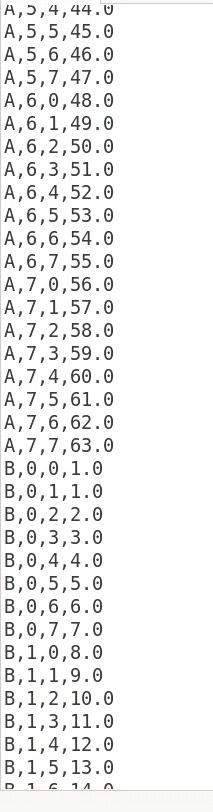
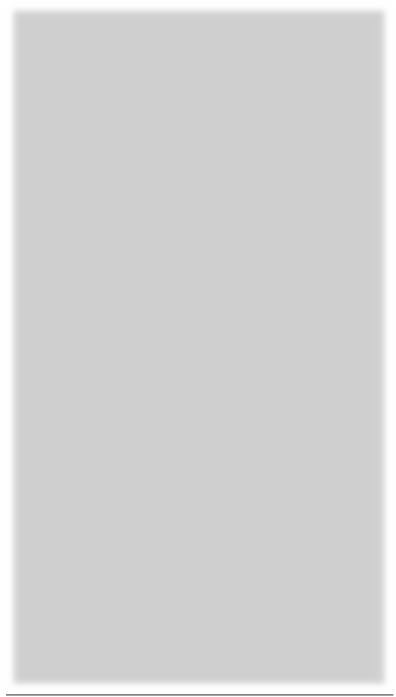
}

## Create the temporary content file in the input directory: Command: sudo mkdir input

**Command:** sudo gedit input/matrix.txt

## Enter the 8x8 matrix on that file

**Sample matrix 8x8 matrix dataset**



**Put the matrix input into HDFS:**

**Command:** hadoop fs -mkdir inputMatrix

**Command:** hadoop fs -put input/matrix.txt inputMatrix/

## Create jar file MatrixMultiplication Program:

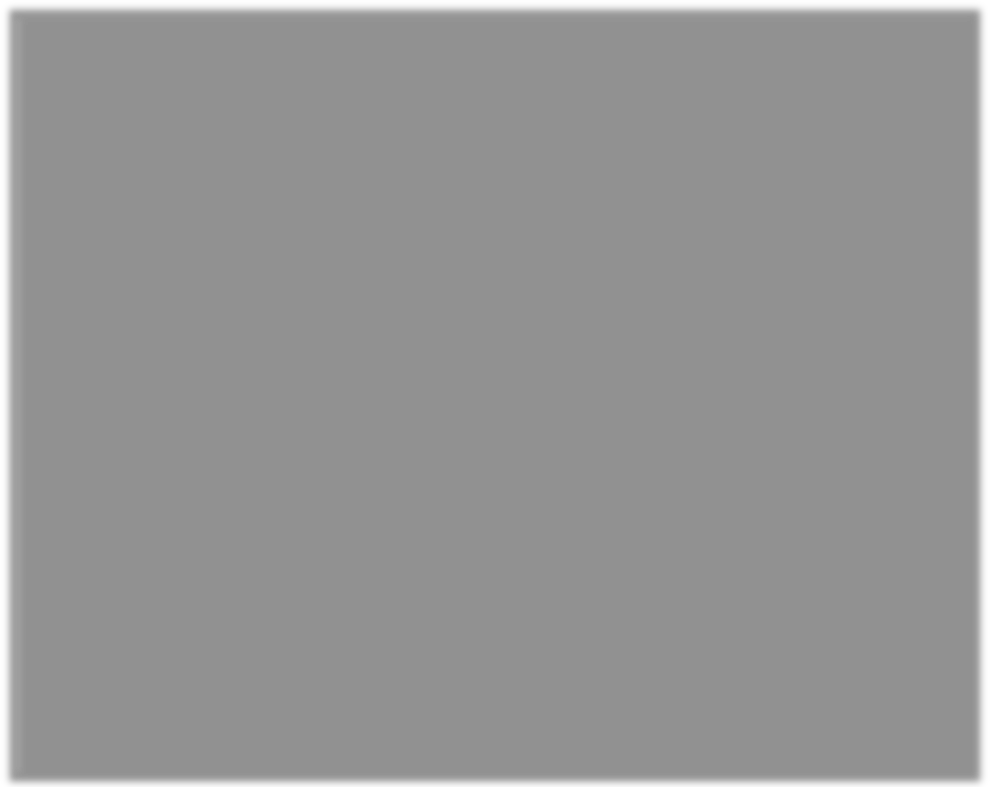
**Command:** hadoop com.sun.tools.javac.Main MatrixMul.java

**Command:** jar cvf mc.jar MatrixMul \*.class

## Run mc jar file on input directory:

**Command:** hadoop jar mc.jar MatrixMul inputMatrix/matrix.txt out1

## To see the output browse the file system:



**RESULT:**

Thus matrix multiplication is successfully done with hadoop map reduce.

|  |  |
| --- | --- |
| **EX.No:6** | **PIG LATIN INSTALLATION** |
| **DATE:** |

## Aim:

Install and Run Pig then write Pig Latin scripts to sort, group, join, project, and filter your data.

## Procedure:

**Download and extract pig-0.13.0.**

**Command:** wget <https://archive.apache.org/dist/pig/pig-0.13.0/pig-0.13.0.tar.gz>

**Command:** tar xvf pig-0.13.0.tar.gz

**Command:** sudo mv pig-0.13.0 /usr/lib/pig

## Set Path for pig:

**Command:**

sudo gedit $HOME/.bashrc export PIG\_HOME=/usr/lib/pig export PATH=$PATH:$PIG\_HOME/bin

export PIG\_CLASSPATH=$HADOOP\_COMMON\_HOME/conf pig.properties file

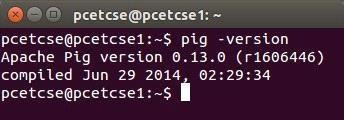
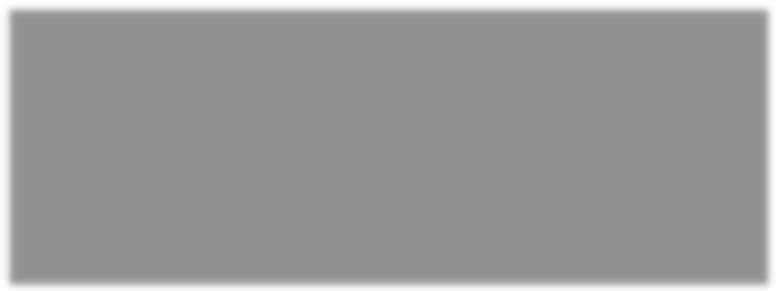
In the conf folder of Pig, we have a file named pig.properties. In the pig.properties file, you can set various parameters as given below.

pig -h properties

## 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.

**Command:** pig -version



## Local mode Command:

$ **pig -x local**

15/09/28 10:13:03 INFO pig.Main: Logging error messages to:

/home/Hadoop/pig\_1443415383991.log 2015-09-28 10:13:04,838 [main]

INFO org.apache.pig.backend.hadoop.execution engine.HExecutionEngine - Connecting to hadoop file system at: file:///

grunt>

## MapReduce mode Command:

$ **pig -x mapreduce**

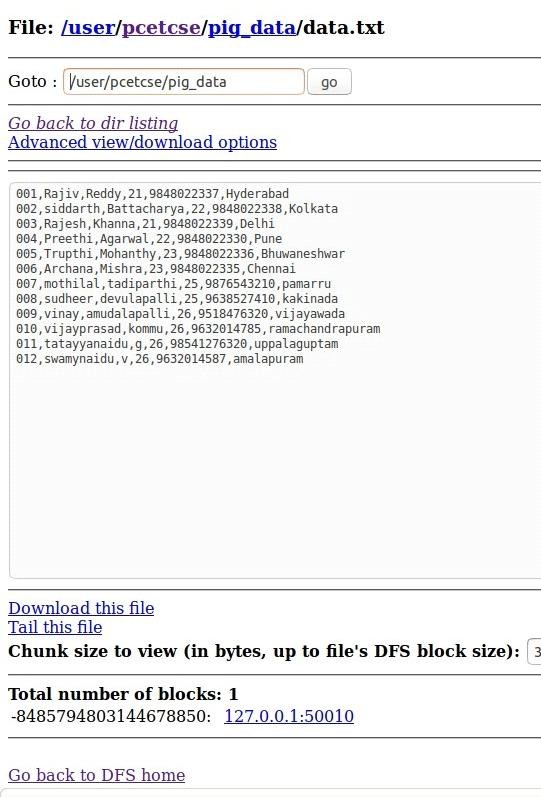
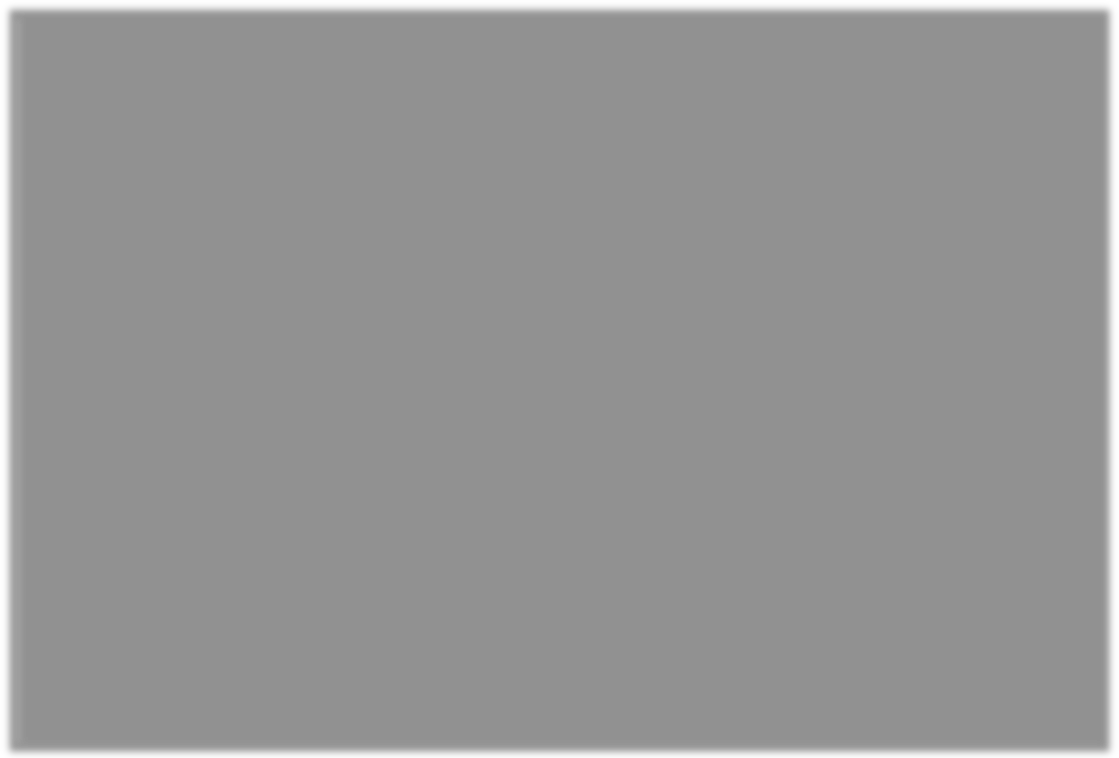
15/09/28 10:28:46 INFO pig.Main: Logging error messages to:/home/Hadoop/pig\_1443416326123.log

2015-09-28 10:28:46,427 [main] INFO org.apache.pig.backend.hadoop.execution engine.HExecutionEngine - Connecting to hadoop file system at: file:///

## Grouping Of Data:

**Put dataset into hadoop:**

**Command:** hadoop fs -put pig/input/data.txt pig\_data/



## Run pig script program of GROUP on hadoop mapreduce grunt>

student\_details = LOAD 'hdfs://localhost:8020/user/pcetcse/pig\_data/student\_details.txt' USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);

group\_data = GROUP student\_details by age; Dump group\_data;

## Joining Of Data:

Run pig script program of JOIN on hadoop mapreduce

## grunt>

customers = LOAD 'hdfs://localhost:8020/user/pcetcse/pig\_data/customers.txt' USING PigStorage(',')as (id:int, name:chararray, age:int, address:chararray, salary:int);

orders = LOAD 'hdfs://localhost:8020/user/pcetcse/pig\_data/orders.txt' USING PigStorage(',')as (oid:int, date:chararray, customer\_id:int, amount:int);

**grunt>** coustomer\_orders = JOIN customers BY id, orders BY customer\_id;

## Verification

Verify the relation **coustomer\_orders** using the **DUMP** operator as shown below.

**grunt>** Dump coustomer\_orders;

## Output:

You will get the following output that wills the contents of the relation named

## coustomer\_orders.

(2,Khilan,25,Delhi,1500,101,2009-11-20 00:00:00,2,1560)

(3,kaushik,23,Kota,2000,100,2009-10-08 00:00:00,3,1500)

(3,kaushik,23,Kota,2000,102,2009-10-08 00:00:00,3,3000)

(4,Chaitali,25,Mumbai,6500,103,2008-05-20 00:00:00,4,2060)

## Sorting of Data:

Run pig script program of SORT on hadoop mapreduce

Assume that we have a file named **student\_details.txt** in the HDFS directory **/pig\_data/**

as shown below.

**student\_details.txt** 001,Rajiv,Reddy,21,9848022337,Hyderabad 002,siddarth,Battacharya,22,9848022338,Kolkata 003,Rajesh,Khanna,22,9848022339,Delhi 004,Preethi,Agarwal,21,9848022330,Pune 005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar 006,Archana,Mishra,23,9848022335,Chennai 007,Komal,Nayak,24,9848022334,trivendram 008,Bharathi,Nambiayar,24,9848022333,Chennai

And we have loaded this file into Pig with the schema name **student\_details** as shown below.

## grunt>

student\_details = LOAD

„hdfs://localhost:8020/user/pcetcse/pig\_data/student\_details.txt' USING PigStorage(',')as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);

Let us now sort the relation in a descending order based on the age of the student and store it into another relation named **data** using the **ORDER BY** operator as shown below.

**grunt> order\_by\_data** = ORDER **student\_details** BY age DESC;

## Verification

Verify the relation **order\_by\_data** using the **DUMP** operator as shown below.

**grunt>** Dump **order\_by\_**data;

## Output:

It will produce the following output, displaying the contents of the relation **order\_by\_data** as follows.

(8,Bharathi,Nambiayar,24,9848022333,Chennai) (7,Komal,Nayak,24,9848022334,trivendram) (6,Archana,Mishra,23,9848022335,Chennai) (5,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar) (3,Rajesh,Khanna,22,9848022339,Delhi) (2,siddarth,Battacharya,22,9848022338,Kolkata) (4,Preethi,Agarwal,21,9848022330,Pune) (1,Rajiv,Reddy,21,9848022337,Hyderabad)

## Filtering of data:

Run pig script program of FILTER on hadoop mapreduce

Assume that we have a file named **student\_details.txt** in the HDFS directory **/pig\_data/**

as shown below.

**student\_details.txt** 001,Rajiv,Reddy,21,9848022337,Hyderabad 002,siddarth,Battacharya,22,9848022338,Kolkata 003,Rajesh,Khanna,22,9848022339,Delhi 004,Preethi,Agarwal,21,9848022330,Pune 005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar 006,Archana,Mishra,23,9848022335,Chennai 007,Komal,Nayak,24,9848022334,trivendram 008,Bharathi,Nambiayar,24,9848022333,Chennai

And we have loaded this file into Pig with the schema name **student\_details** as shown below.

## grunt>

student\_details = LOAD

„hdfs://localhost:8020/user/pcetcse/pig\_data/student\_details.txt' USING PigStorage(',')as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);

Let us now use the Filter operator to get the details of the students who belong to the city Chennai.

**grunt>** filter\_data = FILTER student\_details BY city == 'Chennai';

## Verification

Verify the relation **filter\_data** using the **DUMP** operator as shown below.

**grunt>** Dump filter\_data;

## Output:

It will produce the following output, displaying the contents of the relation **filter\_data** as follows.

(6,Archana,Mishra,23,9848022335,Chennai) (8,Bharathi,Nambiayar,24,9848022333,Chennai)

**RESULT:**

Thus, the pig Latin has been executed and installed successfully.

|  |  |
| --- | --- |
| **EX.NO: 7** | **HIVE INSTALLATION** |
| **DATE:** |

## AIM

Install and Run Hive then use Hive to create, alter, and drop databases, tables, views, functions, and indexes

## Download and extract Hive:

**Command:** wget [https://archive.apache.org/dist/hive/hive-0.14.0/apache-hive-0.14.0-](https://archive.apache.org/dist/hive/hive-0.14.0/apache-hive-0.14.0-bin.tar.gz) [bin.tar.gz](https://archive.apache.org/dist/hive/hive-0.14.0/apache-hive-0.14.0-bin.tar.gz)

**Command:** tar zxvf apache-hive-0.14.0-bin.tar.gz **Command:** sudo mv apache-hive-0.13.1-bin /usr/lib/hive **Command:**

sudo gedit $HOME/.bashrc

export HIVE\_HOME=/usr/lib/hive

export PATH=$PATH:$HIVE\_HOME/bin

export CLASSPATH=$CLASSPATH:/usr/lib/hadoop/lib/\*.jar export CLASSPATH=$CLASSPATH:/usr/lib/hive/lib/\*.jar

**Command:** sudo cd $HIVE\_HOME/conf

**Command:** sudo cp hive-env.sh.template hive-env.sh export HADOOP\_HOME=/usr/lib/hadoop

## Downloading Apache Derby

The following command is used to download Apache Derby. It takes some time to download.

**Command:** wget [http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-](http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.2.0-bin.tar.gz) [HYPERLINK "http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-](http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.2.0-bin.tar.gz) [10.4.2.0-bin.tar.gz" HYPERLINK "http://archive.apache.org/dist/db/derby/db-derby-](http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.2.0-bin.tar.gz) [10.4.2.0/db-derby-10.4.2.0-bin.tar.gz" HYPERLINK](http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.2.0-bin.tar.gz) ["http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.2.0-](http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.2.0-bin.tar.gz) [bin.tar.gz"10.4.2.0-bin.tar.gz](http://archive.apache.org/dist/db/derby/db-derby-10.4.2.0/db-derby-10.4.2.0-bin.tar.gz)

**Command:** tar zxvf db-derby-10.4.2.0-bin.tar.gz

**Command:** sudo mv db-derby-10.4.2.0-bin

/usr/lib/derby

## Command:

sudo gedit $HOME/.bashrc

export DERBY\_HOME=/usr/local/derby export PATH=$PATH:$DERBY\_HOME/bin export CLASSPATH=$CLASSPATH:$DERBY\_HOME/lib/der

by.jar:$DERBY\_HOME/lib/ derbytools.jar:$DERBY\_HOME/lib/derbyclient.jar **Command:**

sudo mkdir $DERBY\_HOME/data

## Command:

sudo cd $HIVE\_HOME/conf

## Command:

sudo cp hive-default.xml.template hive- site.xml

## Command:

Sudo gedit $HOVE\_HOME/conf/hive-site.xml

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value> jdbc:derby://localhost:1527/metastore\_db;create=true

</value>

<description>JDBC connect string for a JDBC metastore </description>

</property>

## Create a file named jpox.properties and add the following lines into it:

javax.jdo.PersistenceManagerFactoryClass = org.jpox.PersistenceManagerFactoryImpl org.jpox.autoCreateSchema = false

org.jpox.validateTables = false org.jpox.validateColumns = false org.jpox.validateConstraints = false org.jpox.storeManagerType = rdbms org.jpox.autoCreateSchema = true org.jpox.autoStartMechanismMode=checked org.jpox.transactionIsolation = read\_committed javax.jdo.option.DetachAllOnCommit = true javax.jdo.option.NontransactionalRead = true

javax.jdo.option.ConnectionDriverName = org.apache.derby.jdbc.ClientDriver javax.jdo.option.ConnectionURL = jdbc:derby://hadoop1:1527/metastore\_db;create = true

javax.jdo.option.ConnectionUserName = APP javax.jdo.option.ConnectionPassword = mine

**Command:** HADOOP\_HOME/bin/hadoop fs -mkdir /tmp

**Command:** HADOOP\_HOME/bin/hadoop fs -mkdir /user/hive/warehouse

**Command:** HADOOP\_HOME/bin/hadoop fs -chmod g+w /tmp

**Command:** HADOOP\_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse

## SOURCE CODE:

**TEXT FILE:**

101,rasi,10000

**HDFS:**

## Create a directory :

[cloudera@quickstart ~]$ **hadoop fs -mkdir hive\_dir**

## Put the file into directory:

[cloudera@quickstart ~]$ **hadoop fs -put /home/cloudera/Desktop/details hive\_dir**

[cloudera@quickstart ~]$ **hadoop fs -ls hive\_dir/**

Found 1 items

-rw-r--r-- 1 cloudera cloudera 15 2019-10-09 22:40 hive\_dir/details

## To see the contents of the file:

[cloudera@quickstart ~]$ **hadoop fs -cat hive\_dir/details**

101,rasi,10000

## To login hive environment:

[cloudera@quickstart ~]$ **hive**

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties WARNING: Hive CLI is deprecated and migration to Beeline is recommended.

## Database and table creation hive> CREATE DATABASE h; OK

Time taken: 0.463 seconds

hive> **use h;**

OK

Time taken: 0.027 seconds

## Creation of table:

hive> **CREATE TABLE DET(id int,name String,amount int)**

* **ROW FORMAT DELIMITED**
* **FIELDS TERMINATED BY ','**
* **STORED AS TEXTFILE;**

OK

Time taken: 0.36 seconds

## Loading of data:

hive> **LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/details'**

* **OVERWRITE INTO TABLE DET**;

Loading data to table h.det

Table h.det stats: [numFiles=1, numRows=0, totalSize=15, rawDataSize=0] OK

Time taken: 0.56 seconds

## Displaying the data:

hive> **SELECT \* FROM DET;**

OK

## OUTPUT:

101 rasi 10000

Time taken: 0.475 seconds, Fetched: 1 row(s)

## Altering the name of table:

hive> **ALTER TABLE DET RENAME TO kdet;**

OK

Time taken: 0.118 seconds

hive> **SELECT \* FROM DET;**

FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'DET'

hive> **SELECT \* FROM kdet;**

OK

101 rasi 10000

Time taken: 0.075 seconds, Fetched: 1 row(s)

## Functions:

**Return type Signature Description**

round(double a) It returns the rounded

|  |  |  |
| --- | --- | --- |
| BIGINT |  | |
| BIGINT | floor(double a) | It returns the maximum BIGINT value that is equal or less than the double. |
| BIGINT | ceil(double a) | It returns the minimum BIGINT value that is equal or  greater than the double. |
| double | rand(), rand(int seed) | It returns a random number that changes from row to row. |
| string | concat(string A, string B,...) | It returns the string resulting from concatenating B after A |
| string | substr(string A, int start) | It returns the substring of A starting from start position till  the end of A |

string substr(string A, int start, int length)

hive> SELECT round(2.6) from temp; 2.0

## Views:

**Example**

Let us take an example for view. Assume employee table as given below, with

the fields Id, Name, Salary, Designation, and Dept. Generate a query to retrieve the employee details who earn a salary of more than Rs 30000. We store the result in a view named **emp\_30000**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| | **ID** | **| Name** | **| Salary | Designation** | **| Dept** | **|** |
| |1201 | | Gopal | | 45000 | Technical manager | | TP | | |
| |1202 | | Manisha | | 45000 | Proofreader | | PR | | |
| |1203 | | Masthanvali | | 40000 | Technical writer | | TP | | |
| |1204  |1205 | | Krian  | Kranthi | | 40000 | Hr Admin  | 30000 | Op Admin | | HR  | Admin | | | |

The following query retrieves the employee details using the above scenario: hive> CREATE VIEW emp\_30000 AS

* + SELECT \* FROM employee
  + WHERE salary>30000;

# RESULT:

Thus Hive has been installed and executed successsfully.

|  |  |
| --- | --- |
| **EX.NO: 8** | **PIG OPERATIONS ON CUSTOMER DATABASE** |
| **DATE:** |

## AIM

To create a database and Perform Pig aggregate Operations on Customer database(Group

and Filter).

## SOURCE CODE:

**Text file:**

**Customer\_det**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 101 | rasi | nokia | 111 | 10000 |
| 102 | Shylu | one plus | 222 | 5000 |
| 103 | Shyam | IPhone | 333 | 100000 |
| 104 | Balaji | Real me | 444 | 15000 |
| 105 | mahesh | IPhone | 555 | 100000 |
| 106 | Nivitha | celkon | 666 | 4000 |
| 107 | siddhu | moto | 777 | 7580 |

**Hdfs :**

**Create the temporary content file in the input directory:**

[cloudera@quickstart ~]$ **hadoop fs -mkdir rs Put the file.txt into hdfs:**

[cloudera@quickstart ~]$ **hadoop fs -put /home/cloudera/Desktop/customer\_det rs/**

[cloudera@quickstart ~]$ **hadoop fs -ls rs/**

Found 1 items

-rw-r--r-- 1 cloudera cloudera 193 2019-10-09 21:04 rs/customer\_det

# PIG Environment:

## pig -x local

**Loading the data :**

grunt> **cd = LOAD'/home/cloudera/Desktop/customer\_det' USING PigStorage(',') as (customerid:int, customername:chararray, productname:chararray, orderid:int, price:int);**

grunt> **group\_data = GROUP cd by orderid**

**Display the data:**

grunt> **DUMP group\_data;**

## OUTPUT:

(111,{(101,rasi,nokia,111,10000)})

(222,{(102,Shylu,one plus,222,5000)}) (333,{(103,Shyam,IPhone,333,100000)}) (444,{(104,Balaji,Real me,444,15000)}) (555,{(105,mahesh,IPhone,555,100000)})

(666,{(106,Nivitha,celkon,666,4000)}) (777,{(107,siddhu,moto,777,7580)})

(,{(,,,,)})

## Filtering the data by price greater than 10000:

grunt> **f = FILTER cd BY price > 10000;**

**Display:**

grunt> DUMP f;

## OUTPUT:

(103,Shyam,IPhone,333,100000)

(104,Balaji,Real me,444,15000) (105,mahesh,IPhone,555,100000)

# RESULT:

Thus commands has been executed successsfully.

|  |  |
| --- | --- |
| **EX.No:9** | **HIVE QUERY OPERATIONS ON ELECTRICITY BILL DATABASE** |
| **DATE:** |

## AIM

To perform hive operations on Electricity bill database.

## SOURCE CODE:

**File:electricity bill**

|  |  |  |  |
| --- | --- | --- | --- |
| 101 | Rasika | 21/9/2019 | 2000 |
| 102 | Shylaja | 5/8/2019 | 5200 |
| 103 | mahesh | 03/9/2019 | 2600 |
| 104 | Balaji | 9/10/2019 | 5000 |
| 105 | Mano | 15/8/2019 | 7200 |
| 106 | kavya | 5/9/2019 | 4560 |
| 107 | Shareen | 4/10/2019 | 4000 |
| 108 | maheswari | 2/10/2019 | 7420 |
| 109 | Sri | 7/9/2019 | 1100 |
| 110 | Raj | 8/9/2019 | 3000 |

**Hdfs:**

**Create the temporary content file in the input directory:**

[cloudera@quickstart ~]$ **hadoop fs -mkdir elect**

[cloudera@quickstart ~]$ hadoop fs -ls Found 22 items

drwxr-xr-x - cloudera cloudera 0 2019-01-04 00:04 Matrix

drwxr-xr-x - cloudera cloudera 0 2019-01-18 21:52 bigdata

-rw-r--r-- 1 cloudera cloudera 63 2019-08-01 02:09 copied.txt

drwxr-xr-x - cloudera cloudera 0 2019-07-29 23:42 cs

drwxr-xr-x - cloudera cloudera 0 2019-10-03 00:18 elect

drwxr-xr-x - cloudera cloudera 0 2019-01-18 21:54 exam

drwxr-xr-x - cloudera cloudera 0 2019-01-03 21:17 in

drwxr-xr-x - cloudera cloudera 0 2019-01-03 21:22 inputMatrix

drwxr-xr-x - cloudera cloudera 0 2019-09-12 21:46 mahes

drwxr-xr-x - cloudera cloudera 0 2019-09-12 21:51 mano

drwxr-xr-x - cloudera cloudera 0 2019-01-03 23:59 matrix

drwxr-xr-x - cloudera cloudera 0 2019-10-02 23:59 newrasi

drwxr-xr-x - cloudera cloudera 0 2019-01-18 22:09 one

drwxr-xr-x - cloudera cloudera 0 2019-01-08 21:18 program2

drwxr-xr-x - cloudera cloudera 0 2019-08-27 00:09 rasi

drwxr-xr-x - cloudera cloudera 0 2019-09-12 21:29 rasimano

drwxr-xr-x - cloudera cloudera 0 2019-01-04 00:11 result

|  |  |  |
| --- | --- | --- |
| drwxr-xr-x | - cloudera cloudera | 0 2019-01-18 22:11 sample |
| drwxr-xr-x | - cloudera cloudera | 0 2019-01-08 21:22 sample1 |
| drwxr-xr-x | - cloudera cloudera | 0 2019-01-08 21:15 sandhiya |

-rw-r--r-- 1 cloudera cloudera 41881 2019-01-02 21:14 wd.txt

-rw-r--r-- 1 cloudera cloudera 41881 2019-01-02 20:59 weather\_data.txt

## Put the file.txt into hdfs:

[cloudera@quickstart ~]$ **hadoop fs -put /home/cloudera/Desktop/electricity\_bill elect/**

[cloudera@quickstart ~]$ hadoop fs -ls elect/ Found 1 items

-rw-r--r-- 1 cloudera cloudera 255 2019-10-03 00:19 elect/electricity\_bill

## Using Hive:

[cloudera@quickstart ~]$ **hive**

## Creation of database:

hive> **CREATE DATABASE ELECT\_BILL;**

OK

hive> USE ELECT\_BILL; OK

## Creation of table:

hive> **CREATE TABLE ELEC1(id int,name String,Date String,Amount int)**

* **ROW FORMAT DELIMITED**
* **FIELDS TERMINATED BY ','**
* **STORED AS TEXTFILE;**

OK

## Loading of data:

hive> **LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/electricity\_bill'**

* **OVERWRITE INTO TABLE ELEC1;**

Loading data to table elect\_bill.elec1

Table elect\_bill.elec1 stats: [numFiles=1, numRows=0, totalSize=255, rawDataSize=0] OK

## Display:

hive> **SELECT \* FROM ELEC1;**

OK

## OUTPUT:

|  |  |  |  |
| --- | --- | --- | --- |
| 101 | Rasika | 21/9/2019 | 2000 |
| 102 | Shylaja | 05/8/2019 | 5200 |
| 103 | mahesh | 03/9/2019 | 2600 |

104 Balaji 9/10/2019 5000

105 Mano 15/8/2019 7200

106 kavya 5/9/2019 4560

107 Shareen 4/10/2019 4000

108 maheswari 2/10/2019 7420

109 Sri 7/9/2019 1100

110 Raj 8/9/2019 3000

**RESULT:**

Thus commands has been executed successsfully.

|  |  |
| --- | --- |
| **EX.No:10** | **STUDENT DATABASE USING HIVE** |
| **DATE:** |

## AIM

To perform hive operations on Student database.

## SOURCE CODE:

**File:Stud.txt**

|  |  |  |  |
| --- | --- | --- | --- |
| 101 | Rasika | M.SC CS | 99 |
| 102 | Shylaja | M.SC CS | 45 |
| 103 | mahesh | M.SC CS | 78 |
| 104 | Balaji | M.SC CS | 98 |
| 105 | Mano | M.SC IT | 75 |
| 106 | kavya | M.SC CS | 55 |
| 107 | Shareen | M.SC IT | 86 |
| 108 | maheswari | M.SC IT | 82 |
| 109 | Sri | M.SC IT | 97 |
| 110 | Raj | M.SC CS | 73 |

**Hdfs:**

**Create the temporary content file in the input directory:**

[cloudera@quickstart ~]$ **hadoop fs -mkdir s**

[cloudera@quickstart ~]$ hadoop fs -ls Found 22 items

drwxr-xr-x - cloudera cloudera 0 2019-01-04 00:04 Matrix

drwxr-xr-x - cloudera cloudera 0 2019-01-18 21:52 bigdata

-rw-r--r-- 1 cloudera cloudera 63 2019-08-01 02:09 copied.txt

drwxr-xr-x - cloudera cloudera 0 2019-07-29 23:42 s

drwxr-xr-x - cloudera cloudera 0 2019-10-03 00:18 elect

drwxr-xr-x - cloudera cloudera 0 2019-01-18 21:54 exam

drwxr-xr-x - cloudera cloudera 0 2019-01-03 21:17 in

drwxr-xr-x - cloudera cloudera 0 2019-01-03 21:22 inputMatrix

drwxr-xr-x - cloudera cloudera 0 2019-09-12 21:46 mahes

drwxr-xr-x - cloudera cloudera 0 2019-09-12 21:51 mano

drwxr-xr-x - cloudera cloudera 0 2019-01-03 23:59 matrix

drwxr-xr-x - cloudera cloudera 0 2019-10-02 23:59 newrasi

drwxr-xr-x - cloudera cloudera 0 2019-01-18 22:09 one

drwxr-xr-x - cloudera cloudera 0 2019-01-08 21:18 program2

drwxr-xr-x - cloudera cloudera 0 2019-08-27 00:09 rasi

drwxr-xr-x - cloudera cloudera 0 2019-09-12 21:29 rasimano

drwxr-xr-x - cloudera cloudera 0 2019-01-04 00:11 result

drwxr-xr-x - cloudera cloudera 0 2019-01-18 22:11 sample

drwxr-xr-x - cloudera cloudera 0 2019-01-08 21:22 sample1

drwxr-xr-x - cloudera cloudera 0 2019-01-08 21:15 sandhiya

-rw-r--r-- 1 cloudera cloudera 41881 2019-01-02 21:14 wd.txt

-rw-r--r-- 1 cloudera cloudera 41881 2019-01-02 20:59 weather\_data.txt

## Put the file.txt into hdfs:

[cloudera@quickstart ~]$ **hadoop fs -put /home/cloudera/Desktop/stud.txt s/**

[cloudera@quickstart ~]$ hadoop fs -ls s/ Found 1 items

-rw-r--r-- 1 cloudera cloudera 255 2019-10-03 00:19 s/stud

## Using Hive:

[cloudera@quickstart ~]$ **hive**

## Creation of database:

hive> **CREATE DATABASE stud1;**

OK

hive> USE stud1; OK

## Creation of table:

hive> **CREATE TABLE st(id int,name String,class String,total\_marks int)**

* **ROW FORMAT DELIMITED**
* **FIELDS TERMINATED BY ','**
* **STORED AS TEXTFILE;**

OK

## Loading of data:

hive> **LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/stud**

* **OVERWRITE INTO TABLE st;**

Loading data to table stud.st

Table stud.st stats: [numFiles=1, numRows=0, totalSize=255, rawDataSize=0] OK

## Display:

hive> **SELECT \* FROM st;**

OK

## OUTPUT:

|  |  |  |  |
| --- | --- | --- | --- |
| 101 | Rasika | M.SC CS | 99 |
| 102 | Shylaja | M.SC CS | 45 |
| 103 | mahesh | M.SC CS | 78 |
| 104 | Balaji | M.SC IT | 98 |

|  |  |  |  |
| --- | --- | --- | --- |
| 105 | Mano | M.SC CS | 75 |
| 106 | kavya | M.SC CS | 55 |
| 107 | Shareen | M.SC IT | 86 |
| 108 | maheswari | M.SC IT | 82 |
| 109 | Sri | M.SC IT | 97 |
| 110 | Raj | M.SC CS | 73 |

**RESULT:**

Thus commands has been executed successsfully.