DWARAKA DOSS GOVERDHAN DOSS VAISHNAV COLLEGE

ARUMBAKKAM, CHENNAI-106

DEPARTMENT OF COMPUTER SCIENCE (UG & PG)

BIG DATA ANALYTICS LAB

Name :

Register Number :

Class : M.Sc. COMPUTER SCIENCE

Month & Year : NOVEMBER- 2024

Subject Code :

DWARAKA DOSS GOVERDHAN DOSS VAISHNAV COLLEGE

ARUMBAKKAM, CHENNAI-106

DEPARTMENT OF COMPUTER SCIENCE (UG & PG)

BIG DATA ANALYTICS LAB

Name :

Register Number :

Class : M.Sc. COMPUTER SCIENCE

Month & Year : NOVEMBER - 2024

Subject Code :

DWARAKA DOSS GOVERDHAN DOSS VAISHNAV COLLEGE

ARUMBAKKAM, CHENNAI-106

DEPARTMENT OF COMPUTER SCIENCE (UG & PG)

BONAFIDE CERTIFICATE

Certified that this is a Bonafide record of practical work done by

Mr/Ms…….....Reg No .....of

II M.Sc. (Computer Science) during the academic year 2023 - 2024.

FACULTY IN-CHARGE **HEAD OF THE DEPARTMENT**

Submitted for the Practical Examination held on _____
in the P.G. Department of Computer Science at Dwaraka Doss Goverdhan Doss Vaishnav College.

-

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

EX.No:1	PERFORMING SETUP AND INSTALLING HADOOP
DATE: 02/07/2024	

AIM:

To perform the setup and installation of hadoop.

SOURCE CODE:

A) 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

B) 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

C) 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

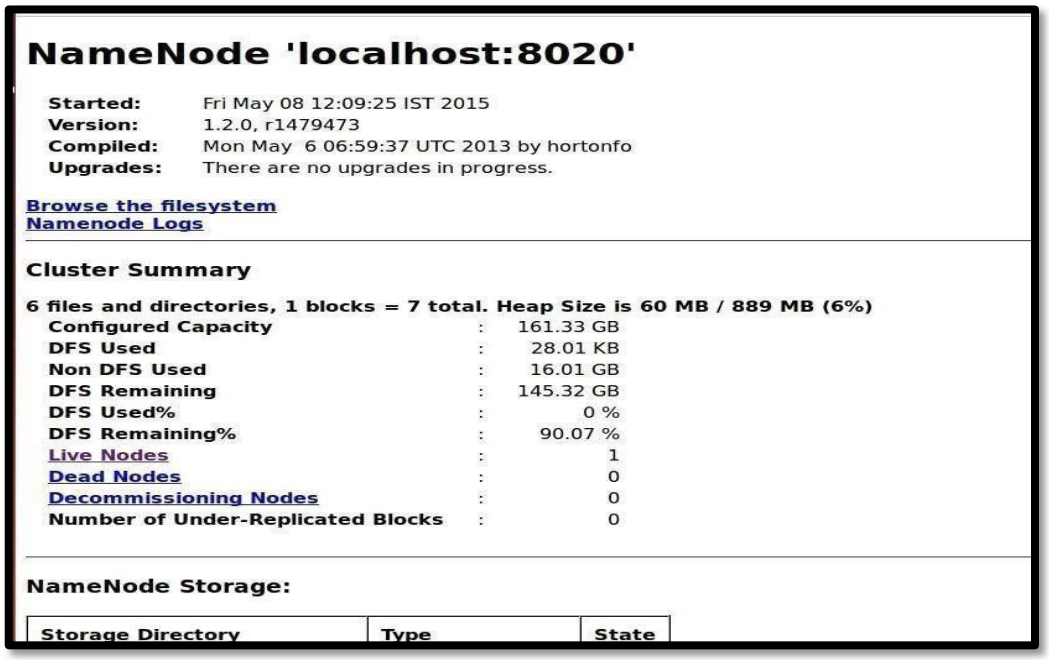
tasktracker

OUTPUT:

Using HDFS monitoring UI

HDFS Namenode on UI

http://locahost:50070/



NameNode 'localhost:8020'

Started: Fri May 08 12:09:25 IST 2015
Version: 1.2.0, r1479473
Compiled: Mon May 6 06:59:37 UTC 2013 by hortonfo
Upgrades: There are no upgrades in progress.

Browse the filesystem
Namenode Logs

Cluster Summary

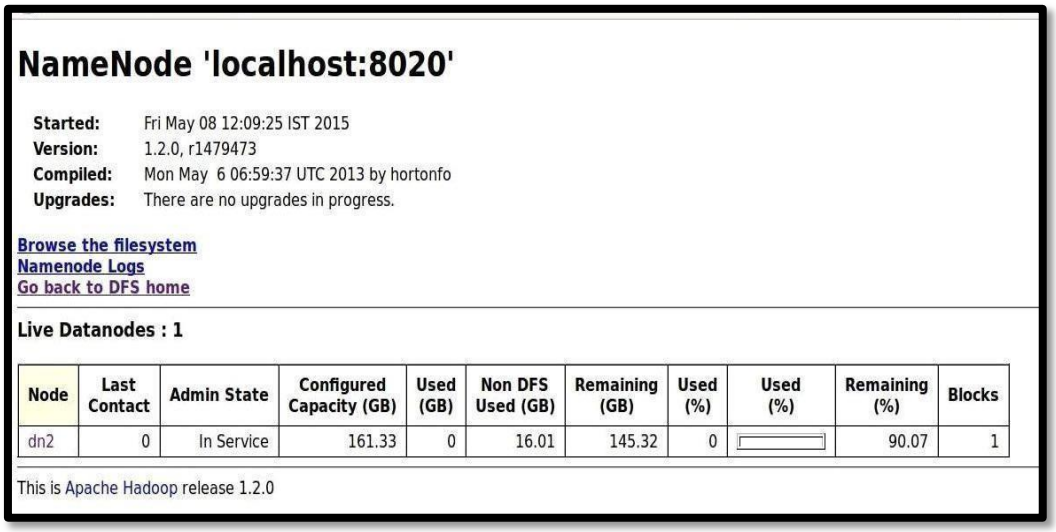
6 files and directories, 1 blocks = 7 total. Heap Size is 60 MB / 889 MB (6%)

Configured Capacity	:	161.33 GB
DFS Used	:	28.01 KB
Non DFS Used	:	16.01 GB
DFS Remaining	:	145.32 GB
DFS Used%	:	0 %
DFS Remaining%	:	90.07 %
Live Nodes	:	1
Dead Nodes	:	0
Decommissioning Nodes	:	0
Number of Under-Replicated Blocks	:	0

NameNode Storage:

Storage Directory	Type	State

HDFS Live Nodes list11



NameNode 'localhost:8020'

Started: Fri May 08 12:09:25 IST 2015
Version: 1.2.0, r1479473
Compiled: Mon May 6 06:59:37 UTC 2013 by hortonfo
Upgrades: There are no upgrades in progress.

Browse the filesystem
Namenode Logs
Go back to DFS home

Live Datanodes : 1

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
dn2	0	In Service	161.33	0	16.01	145.32	0		90.07	1

This is Apache Hadoop release 1.2.0

localhost Hadoop Machine List

Active Task Trackers

Task Trackers												
Name	Host	# running tasks	Max Map Tasks	Max Reduce Tasks	Task Failures	Directory Failures	Node Health Status	Seconds Since Node Last Healthy	Total Tasks Since Start	Succeeded Tasks Since Start	Total Tasks Last Day	Succeeded Tasks Last Day
tracker_dn2:localhost/127.0.0.1:49820	dn2	0	2	2	0	0	N/A	0	0	0	0	0

This is Apache Hadoop release 1.2.0

localhost Hadoop Map/Reduce Administration

State: RUNNING
Started: Fri May 08 12:09:33 IST 2015
Version: 1.2.0, r1479473
Compiled: Mon May 6 06:59:37 UTC 2013 by hortonfo
Identifier: 201505081209
SafeMode: OFF

Cluster Summary (Heap Size is 55.5 MB/889 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Exc N
0	0	0	1	0	0	0	0	2	2	4.00	0	0	0

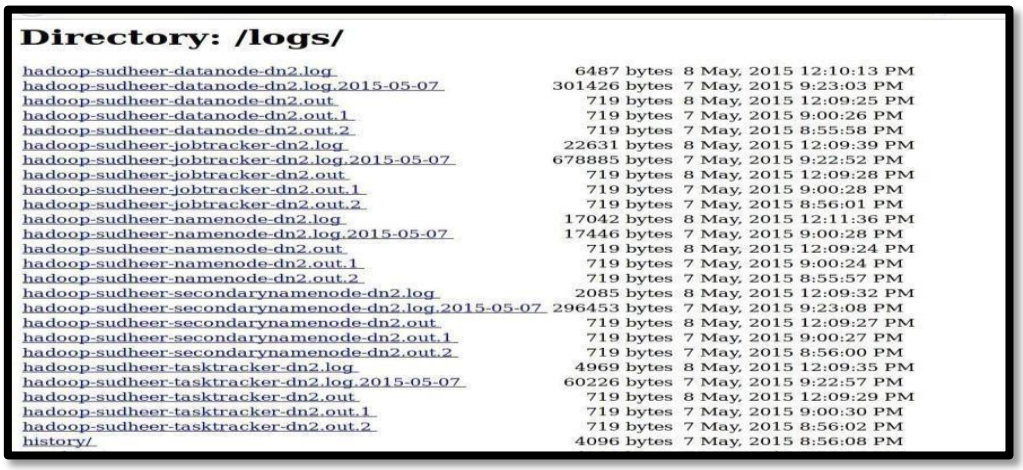
Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

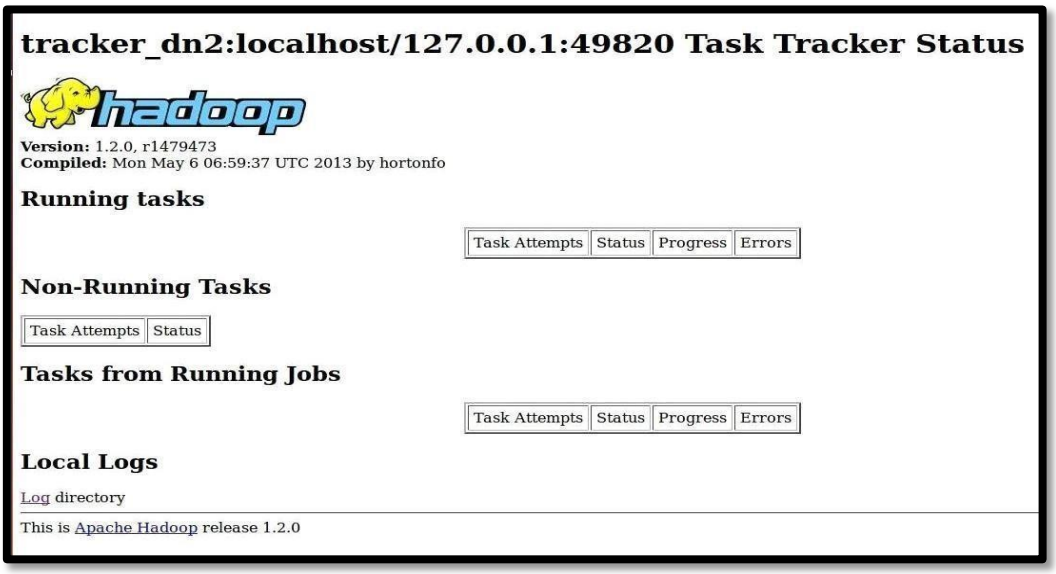
HDFS Logs



Directory: /logs/

hadoop-sudheer-datanode-dn2.log	6487 bytes	8 May, 2015	12:10:13 PM
hadoop-sudheer-datanode-dn2.log.2015-05-07	301426 bytes	7 May, 2015	9:23:03 PM
hadoop-sudheer-datanode-dn2.out	719 bytes	8 May, 2015	12:09:25 PM
hadoop-sudheer-datanode-dn2.out.1	719 bytes	7 May, 2015	9:00:26 PM
hadoop-sudheer-datanode-dn2.out.2	719 bytes	7 May, 2015	8:55:58 PM
hadoop-sudheer-jobtracker-dn2.log	22631 bytes	8 May, 2015	12:09:39 PM
hadoop-sudheer-jobtracker-dn2.log.2015-05-07	678885 bytes	7 May, 2015	9:22:52 PM
hadoop-sudheer-jobtracker-dn2.out	719 bytes	8 May, 2015	12:09:28 PM
hadoop-sudheer-jobtracker-dn2.out.1	719 bytes	7 May, 2015	9:00:28 PM
hadoop-sudheer-jobtracker-dn2.out.2	719 bytes	7 May, 2015	8:56:01 PM
hadoop-sudheer-namenode-dn2.log	17042 bytes	8 May, 2015	12:11:36 PM
hadoop-sudheer-namenode-dn2.log.2015-05-07	17446 bytes	7 May, 2015	9:00:28 PM
hadoop-sudheer-namenode-dn2.out	719 bytes	8 May, 2015	12:09:24 PM
hadoop-sudheer-namenode-dn2.out.1	719 bytes	7 May, 2015	9:00:24 PM
hadoop-sudheer-namenode-dn2.out.2	719 bytes	7 May, 2015	8:55:57 PM
hadoop-sudheer-secondarynamenode-dn2.log	2085 bytes	8 May, 2015	12:09:32 PM
hadoop-sudheer-secondarynamenode-dn2.log.2015-05-07	296453 bytes	7 May, 2015	9:23:08 PM
hadoop-sudheer-secondarynamenode-dn2.out	719 bytes	8 May, 2015	12:09:27 PM
hadoop-sudheer-secondarynamenode-dn2.out.1	719 bytes	7 May, 2015	9:00:27 PM
hadoop-sudheer-secondarynamenode-dn2.out.2	719 bytes	7 May, 2015	8:56:00 PM
hadoop-sudheer-tasktracker-dn2.log	4969 bytes	8 May, 2015	12:09:35 PM
hadoop-sudheer-tasktracker-dn2.log.2015-05-07	60226 bytes	7 May, 2015	9:22:57 PM
hadoop-sudheer-tasktracker-dn2.out	719 bytes	8 May, 2015	12:09:29 PM
hadoop-sudheer-tasktracker-dn2.out.1	719 bytes	7 May, 2015	9:00:30 PM
hadoop-sudheer-tasktracker-dn2.out.2	719 bytes	7 May, 2015	8:56:02 PM
history/	4096 bytes	7 May, 2015	8:56:08 PM

http://locahost:50070/logs/



tracker_dn2:localhost/127.0.0.1:49820 Task Tracker Status

Version: 1.2.0, r1479473
Compiled: Mon May 6 06:59:37 UTC 2013 by hortonfo

Running tasks

Task Attempts	Status	Progress	Errors

Non-Running Tasks

Task Attempts	Status

Tasks from Running Jobs

Task Attempts	Status	Progress	Errors

Local Logs

Log directory

This is Apache Hadoop release 1.2.0

RESULT:

Thus, the installation of the hadoop is done successfully.

EX.No:2	HDFS BASIC COMMAND LINE FILE OPERATIONS
DATE: 05/07/2024	

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>

3. **Upload and download a file in HDFS:**

   Upload: **Command:** hadoop fs -put <localsrc> <HDFS_dest_path>
   Download: **Command:** hadoop fs -get <HDFS_src> <localdst>

4. **See contents of a file:**
   Command: hadoop fs -cat <path[filename]>

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

6. Copy a file from/To Local file system to HDFS:
   Command: hadoop fs -copyFromLocal <localsrc> URI
   Command: hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localsrc>

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

8. **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>

9. **Display last few lines of a file:**

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

10. **Display the aggregate length of a file:**

    Command: hadoop fs -du <path>

11. **Getting help:**

    Command: hadoop fs -help

12. **Adding files and directories: Creating a directory:**

    Command: hadoop fs -mkdir input/

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

14. **Retrieving files:**

    Command: hadoop fs -get input/file01 localfs

15. 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: 10/07/2024	

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: 15/07/2024	

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

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

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

RESULT:

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

EX.No:5	Matrix Multiplication using Hadoop Map Reduce
DATE: 19/07/2024	

AIM:

To implement Matrix Multiplication with Hadoop Map Reduce.

SOURCE CODE:

```
import java.io.IOException;
import java.lang.InterruptedException;
import java.util.*;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
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;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class matrix extends Configured implements Tool {
   /*******************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();
   int m=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++)
    {
            outputKey.set(indicesAndValue[1] + "," +k);
            outputValue.set("A," + indicesAndValue[2] + "," + indicesAndValue[3]);
            context.write(outputKey, outputValue);
    }
    } else {
    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]));
    }
```

```
    }
    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 {
    int res = ToolRunner.run(new Configuration(), new matrix(), args);
    System.exit(res);
}
@Override
public int run(String[] args) throws Exception {
Configuration conf = this.getConf();
// A is an m-by-n matrix; B is an n-by-p matrix.
conf.set("m", "2");
conf.set("n", "2");
conf.set("p", "2");
Job job =Job.getInstance(conf,"matrix");
job.setJarByClass(matrix.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();
return job.waitForCompletion(true) ? 0 : 1;
}
}
```

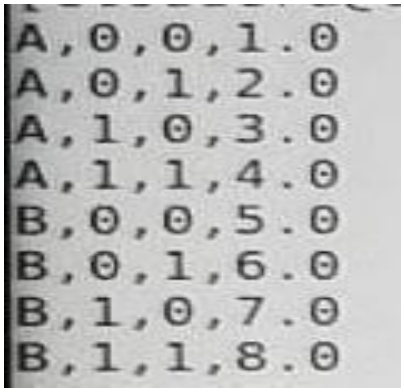
Create the temporary content file in the input directory:

Command: sudo mkdir input

Command: sudo gedit input/matrix.txt

Enter the 2x2 matrix on that file

Sample matrix 2x2 matrix dataset



```
A,0,0,1.0
A,0,1,2.0
A,1,0,3.0
A,1,1,4.0
B,0,0,5.0
B,0,1,6.0
B,1,0,7.0
B,1,1,8.0
```

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:

```
0,0,19.0
0,1,22.0
1,0,43.0
1,1,50.0
```

RESULT:

Thus matrix multiplication is successfully done with hadoop map reduce.

EX.No:6	PIG LATIN INSTALLATION
DATE: 23/07/2024	

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

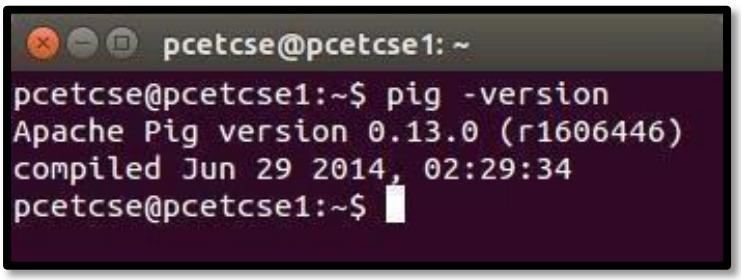
Set Path for pig:

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



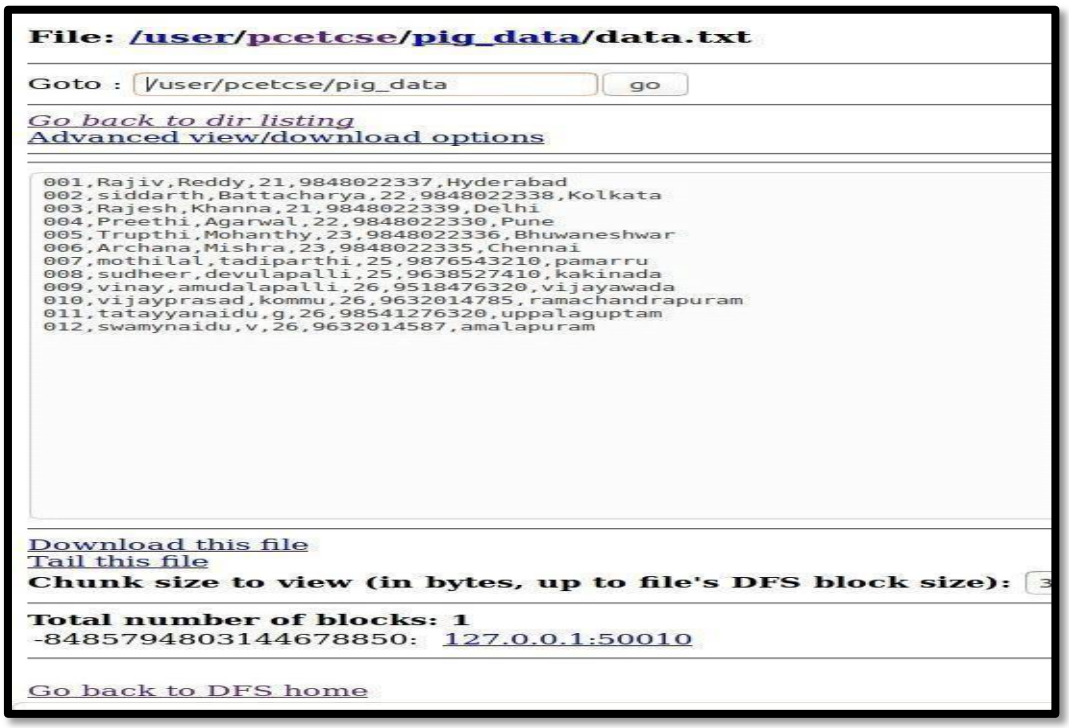
Local mode
Command:
$ **pig -x local**
15/09/28 10:13:03 INFO pig.Main: Logging error messages to:
/home/Hadoop/pig_1443415383991.log
2015-09-28 10:13:04,838 [main]
INFO org.apache.pig.backend.hadoop.execution
engine.HExecutionEngine - Connecting to hadoop file system at: file:///

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: 26/07/2024	

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-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-HYPERLINK "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-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"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
BIGINT	round(double a)	It returns the rounded
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	Krian	40000	Hr Admin	HR
1205	Kranthi	30000	Op Admin	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: 31/07/2024	

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: 13/08/2024	

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: 29/08/2024	

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, the commands has been executed successfully.