# LAB RECORD

**ON**

**BIG DATA ANALYTICS (16ITC35)**

Apurva Karne

160117737067

4/VII Sem/IT H-2



**DEPARTMENT OF INFORMATION TECHNOLOGY CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)**

**(Affiliated to Osmania University; Accredited by NBA (AICTE) and NAAC (UGC), ISO Certified 9001:2015), Kokapet (V), GANDIPET(M), HYDERABAD– 500 075**

**Website:** [**www.cbit.ac.in**](http://www.cbit.ac.in/)





**CERTIFICATE**

This is to certify that this is a bonafide record of the work done by Miss Apurva Karne bearing roll number 1601-17-737-067 of BE- VII Sem in the Big Data Analytics lab during the academic year 2020-21.

**Staff Member In Charge Ms. Kratika Sharma**

Asst. Professor, Dept. of IT, CBIT, Hyderabad.

# List of Experiments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SNo.** | **Program name** | **Page no.** | **Date** | **Remarks** |
| 1 | Understanding and using basic HDFS  Commands |  |  |  |
| 2 | Word count application using Mapper Reducer on  single node cluster |  |  |  |
| 3 | Analysis of weather data set using Mapper  Reducer on single node cluster |  |  |  |
| 4 | Web log analysis using Mapper Reducer on single  node cluster |  |  |  |
| 5 | Pig Daily show  data analysis |  |  |  |
| 6 | Pig Aviation data  Analysis |  |  |  |
| 7 | Pig Sentiment  Analysis |  |  |  |
| 8 | Pig User Defined  Function |  |  |  |
| 10 | Hive user Defined  Function |  |  |  |
| 11 | Hive partitioning  and bucketing |  |  |  |
| 12 | Spark Scala installation steps and word count  Program |  |  |  |

**PROGRAM 1**

**AIM:** **Understanding and using basic HDFS commands**

**Hadoop HDFS** is a distributed file system which provides redundant storage space for files having huge sizes. It is used for storing files which are in the range of terabytes to petabytes.

Command Line is one of the simplest interface to HDFS. Below are the basic HDFS File System Commands which are similar to Linux file system commands. Once the hadoop daemons are started running, HDFS file system is ready and file system operations like creating directories, moving files, deleting files, reading files and listing directories.

1. **mkdir:** To create a directory in hadoop file system.

Syntax:hdfs fs -mkdir <folder-name>

Example: hdfs fs -mkdir /hello

1. **ls:** This command is used to list all the files and directories.

Syntax: hdfs fs -ls <path>

Example: hdfs dfs -ls /

1. **put:**  To copy files/folders from local file system to hdfs store.

Syntax: hdfs fs -put <local file path> <dest present on hdfs>

Example: hdfs fs -put ../Desktop/file.txt /hello

1. **get:** To copy files/folders from hdfs store to local file system.

Syntax: hdfs fs -get <srcfile on hdfs> <local file dest>

Example: hdfs fs -get /hello/file.txt ..Desktop/folder

1. **cat:** To print file contents.

Syntax: hdfs fs -put <path>

Example: hdfs fs -cat /hello/file.txt

1. **cp:** This command is used to copy files within hdfs.

Syntax: hdfs fs -cp <src on hdfs> <dest on hdfs>

Example: hdfs -cp /hello /hello1

1. **mv:** This command is used to move files within hdfs.

Syntax: hdfs fs -mv <src on hdfs> <dest on hdfs>

Example: hdfs -mv /hello/myfile.txt /hello1

1. **rm**: This command removes the file or empty directory identified by path.

Syntax: hdfs fs -rm <path>

Example: hdfs fs -rm /hello/file.txt

1. **getmerge:** This command retrieves all files that match the path src in HDFS, and copies them to a single, merged file in the local file system identified by localDest.

Syntax: hdfs fs -getmerge <src> <localDest>

Example:

1. **setrep**: This command is used to change the replication factor of a file/directory in HDFS. By default it is 3 for anything which is stored in HDFS (as set in hdfs core-site.xml).

Syntax: hadoop fs -setrep <rep\_factor> <path>

Example: hadoop fs -setrep 6 hello/sample

1. **touchz:** It creates an empty file.

Syntax: hdfs fs -touchz <file-path>

Example: hdfs fs -touchz /hello/myfile.txt

1. **test:** Returns 1 if path exists; has zero length; or is a directory or 0 otherwise.

Options Description

-d Check whether the path given by the user is a directory or not, return 0 if it is a directory.

-e Check whether the path given by the user exists or not, return 0 if the path exists.

-f Check whether the path given by the user is a file or not, return 0 if it is a file.

-s Check if the path is not empty, return 0 if a path is not empty.

-r return 0 if the path exists and read permission is granted

-w return 0 if the path exists and write permission is granted

-z Checks whether the file size is 0 byte or not, return 0 if the file is of 0 bytes.

Syntax: hdfs fs -test -[ezd] <path>

Example: hdfs fs -test -d /sample

1. **appendToFile:** The HDFS fs shell command appendToFile appends the content of single or multiple local files specified in the localsrc to the provided destination file on the HDFS.

Syntax: hadoop fs -appendToFile <localsrc> <dest>

Example: hadoop fs -appendToFile /hello/file /hello1/sample

1. **df:** The Hadoop fs shell command df shows the capacity, size, and free space available on the HDFS file system.The -h option formats the file size in the human-readable format.

Syntax: hadoop fs -df [-h] <path>

Example: hadoop fs -df -h

1. **du**: This Hadoop fs shell command du prints a summary of the amount of disk usage of all files/directories in the path.

Syntax: hdfs fs -du <dirName>

Example: hdfs fs -du /hello

1. **count:** The Hadoop fs shell command count counts the number of files, directories, and bytes under the paths that matches the specified file pattern.

Options:

-q – shows quotas(quota is the hard limit on the number of names and amount of space used for individual directories)

-u – it limits output to show quotas and usage only

-h – shows sizes in a human-readable format

-v – shows header line

Syntax: hadoop fs -count [options] <path>

Example: hadoop fs -count -v /

1. **chgrp:** The Hadoop fs shell command chgrp changes the group of the file specified in the path. The user must be the owner of the file or superuser.Sets group recursively if -R is specified.

Syntax: hdfs fs -chgrp [-R] group <path>...

Example: hdfs fs -chgrp newgroup /hello

1. **chmod:** Changes the file permissions associated with one or more objects identified by path. Performs changes recursively with -R. Mode is a 3-digit octal mode, or {augo}+/-{rwxX}. Assumes if no scope is specified and does not apply an umask.

Syntax: hdfs fs -chmod [-R] mode,mode,.. <path>

Example: hdfs fs -chmod 744 /sample/file.txt

1. **chown**: Sets the owning user and/or group for files or directories identified by path.... Sets owner recursively if -R is specified.

Syntax: hdfs fs -chown [-R] [owner][:[group]] <path>

Example: hadoop fs -chown newsample /hello

**PROGRAM 2**

**AIM:** **Word count application using Mapper-Reducer on single node cluster.**

In Hadoop, MapReduce is a computation that decomposes large manipulation jobs into individual tasks that can be executed in parallel cross a cluster of servers. The results of tasks can be joined together to compute final results.

MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.

**Steps for execution in java:**

**Step 1.** Open Eclipse> File > New > Java Project >( Name it – WordCountProject) > Finish **Step 2.** Right Click > New > Package ( Name it - wordcount) > Finish

**Step 3.** Right Click on Package > New > Class (Name it - WordCount)

**Step 4.** Add Following Reference Libraries – Right Click on Project > Build Path> Add External Archivals (Here you add ALLJAR)

**Step 5.** Type following Program

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

public static class TokenizerMapper extends Mapper{

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 {

private IntWritable result = new IntWritable();

public void reduce(Text key, Iterable 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(WordCount.class);

job.setMapperClass(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1])); System.exit(job.waitForCompletion(true) ? 0 : 1);

} }

**Explanation:** The program consists of 3 classes:

• Driver class (Public void static main- the entry point)

• Map class which extends public class Mapper and implements the Map function.

• Reduce class which extends public class Reducer and implements the Reduce function. **Step 6:** Make Jar File

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

**Step 7:** Take a text file and move it in HDFS

Before moving it into HDFS Create a directory in HDFS

$ hadoop fs -mkdir /inputdata

$ hadoop fs -put File.txt /inputdata

**Step 8:** Run Jar file

(Hadoop jar jarpath/jarfilename.jar packageName.ClassName PathToInputTextFile PathToOutputDirectry)

$ hadoop jar /home/hduser/Desktop/test/test.jar wordcount.WordCount /inputdata/File.txt /outputdata

**Step 9:** Open Result

You can see the result on terminal

$ hadoop fs -ls /outputdata

$ hadoop fs -cat /outputdata/part-r-00000 Or you can see your result in Hadoop Web Interface <http://localhost:50070/>

Goto utilities> Browse file System> /outputdata

**Steps for execution in python:**

**Step 1:** Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

$ hadoop fs -mkdir /wordcount

$hadoop fs -mkdir /wordput/inp

Move the text file onto the hadoop store.

$ hadoop fs -put /home/hduser/Desktop/bda\_lab/wordcount\_i2/word.txt /wordcount/inp

**Step 2:** Type in the mapper code and reducer code.

**my\_mapper.py**

#!/usr/bin/python

import sys;

for line in sys.stdin:

line = line.strip()

words = line.split()

for word in words:

print ('%s-%s' %(word,1))

**my\_reducer.py**

#!/usr/bin/python

import sys;

from operator import itemgetter;

prev\_word = None

prev\_count = 0

word = None

for line in sys.stdin:

line = line.strip()

word,count = line.split('-',1)

count=int(count)

if word == prev\_word:

prev\_count+=1;

else:

if prev\_word:

print('%s-%s' %(prev\_word,prev\_count))

prev\_word = word;

prev\_count = count;

print('%s-%s' %(prev\_word,prev\_count))

**Step 3:** Run the following command.

$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar **-file** /home/hduser/Desktop/bda\_lab/wordcount\_i2/my\_mapper.py **-mapper** “python my\_mapper.py” **-file** /home/hduser/Desktop/bda\_lab/wordcount\_i2/my\_reducer.py **-reducer** “python my\_reducer.py**” -input** /wordcount/inp **-output** /wordcount/out

**Step 4:** Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/

Goto utilities> Browse file System> /wordcount/out\_2

**PROGRAM 3**

**AIM: Analysis of weather dataset using Mapper-Reducer on single node cluster.**

**Execution in java:**

**//Reducer code**

import java.io.IOException;

import java.util.Iterator;

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

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

public class Treducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable>

{

public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException

{

int max\_temp = 0;

;

while (values.hasNext())

{

int current=values.next().get();

if ( max\_temp < current)

max\_temp = current;

}

output.collect(key, new IntWritable(max\_temp/10));

}

}

**//Mapper code**

import java.io.IOException;

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

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

public class Tmapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

public static final int MISSING = 9999;

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter)

throws IOException {

String line = value.toString();

String year = line.substring(15, 19);

int temperature;

if (line.charAt(87) == '+')

temperature = Integer.parseInt(line.substring(88, 92));

else

temperature = Integer.parseInt(line.substring(87, 92));

String quality = line.substring(92, 93);

if (temperature != MISSING && quality.matches("[01459]"))

output.collect(new Text(year), new IntWritable(temperature));

}

}

**//Driver code**

import org.apache.hadoop.fs.Path;

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

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

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

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

public class Tdriver extends Configured implements Tool{

public int run(String[] args) throws Exception

{

JobConf conf = new JobConf(getConf(), Tdriver.class);

conf.setJobName("HighestDriver");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(Tmapper.class);

conf.setReducerClass(Treducer.class);

Path inp = new Path(args[0]);

Path out = new Path(args[1]);

FileInputFormat.addInputPath(conf, inp);

FileOutputFormat.setOutputPath(conf, out);

JobClient.runJob(conf);

return 0;

}

public static void main(String[] args) throws Exception

{

int res = ToolRunner.run(new Configuration(), new Tdriver(),args);

System.exit(res);

}

}

**Steps for execution in python:**

**Step 1:** Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

$ hadoop fs -mkdir /weather\_data

$hadoop fs -mkdir /weather\_data/input

Move the text file onto the hadoop store.

$ hadoop fs -put /home/hduser/Desktop/bda\_lab/weather\_data/merge /weather\_data/input

**Step 2:** Type in the mapper code and reducer code.

**my\_mapper.py**

#!/usr/bin/python

import sys

l=list()

for line in sys.stdin:

line = line.strip()

year=int(line[15:19])

temp=int(line[87:92])

l.append([year,temp])

for record in l:

print('%s %s' %(record[0],record[1]))

**my\_reducer.py**

#!/usr/bin/python

import sys

dmax=dict()

y\_list=list()

dmin=dict()

for line in sys.stdin:

line = line.strip()

year,temp = map(int,line.split())

if year not in y\_list:

y\_list.append(year)

dmax[year]=temp

dmin[year]=temp

else:

if dmax[year]<temp:

dmax[year]=temp

if dmin[year]>temp:

dmin[year]=temp

print('--------------------------------')

print('year max\_temp min\_temp')

print('--------------------------------')

for i in dmax.keys():

print('%s %s %s' %(i,dmax[i],dmin[i]))

**Step 3:** Run the following command.

$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar **-file** /home/hduser/Desktop/bda\_lab/weather\_data/my\_mapper.py **-mapper** “python my\_mapper.py” **-file** /home/hduser/Desktop/bda\_lab/weather\_data/my\_reducer.py **-reducer** “python my\_reducer.py” **-input** /weather\_data/input/merge **-output** /weather\_data/output1

**Step 4**: Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/ **or** Goto utilities> Browse file System> /weather\_data/output1

**PROGRAM 4**

**AIM: Web Log Analysis using Mapper-Reducer on single node cluster.**

Web logs are data that is generated by web servers for requests they receive. There are various web servers such as Apache, Nginx, Tomcat, and so on. Each web server logs data in a specific format. The server access log records all requests processed by the server. Of course, storing the information in the access log is only the start of log management. The next step is to analyze this information to produce useful statistics. We can store log data in many different format but here we are going to use data from the Apache Web Server, which is in combined access logs.

**Combined Log Format**

A typical configuration for the access log might look as follows.

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{Useragent}i\"" combined CustomLog log/acces\_log combined

**Example:** 127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache\_pb.gif HTTP/1.0" 200 2326 "http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"

**127.0.0.1 (%h)**

This is the IP address of the client (remote host) which made the request to the server.

**- (%l)**

The "hyphen" in the output indicates that the requested piece of information is not available

**frank (%u)**

This is the userid of the person requesting the document as determined by HTTP authentication.

**[10/Oct/2000:13:55:36 -0700] (%t)**

The time that the server finished processing the request.

The format is:

[day/month/year:hour:minute:second zone]

day = 2\*digit

month = 3\*letter

year = 4\*digit

hour = 2\*digit

minute = 2\*digit

second = 2\*digit

zone = (`+' | `-') 4\*digit

**"GET /apache\_pb.gif HTTP/1.0" (\"%r\")**

The request line from the client is given in double quotes. The request line contains a great deal of useful information. First, the method used by the client is GET. Second, the client requested the resource /apache\_pb.gif, and third, the client used the protocol HTTP/1.0.

**200 (%>s)**

This is the status code that the server sends back to the client. This information is very valuable, because it reveals whether the request resulted in a successful response (codes beginning in 2), a redirection (codes beginning in 3), an error caused by the client (codes beginning in 4), or an error in the server (codes beginning in 5).

**2326 (%b)**

The last entry indicates the size of the object returned to the client (in bytes), not including the response headers.

**"http://www.example.com/start.html" (\"%{Referer}i\")**

The "Referer" (sic) HTTP request header. This gives the site that the client reports having been referred from. (This should be the page that links to or includes /apache\_pb.gif). This is the page that is linked to this URL.

**"Mozilla/4.08 [en] (Win98; I ;Nav)" (\"%{User-agent}i\")**

This is the browser identification string.

**QUESTION 1**

Write a map reduce program that reads a weblog file to give URL address and their counts. (URL, Total Count)

**Steps for execution in python:**

**Step 1:** Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

$ hadoop fs -mkdir /web\_log

$hadoop fs -mkdir /web\_log/inputdata

Move the text file onto the hadoop store.

$ hadoop fs -put /home/hduser/Desktop/bda\_lab/web\_log/web\_log.txt /web\_log/inputdata

**Step 2:** Type in the mapper code and reducer code.

**my\_mapper.py**

#!/usr/bin/python

import sys;

import re;

for line in sys.stdin:

line = line.strip()

words=line.split()

x=re.findall("http://",words[10])

if (x):

print('%s %s' %(words[10],1))

**my\_reducer.py**

#!/usr/bin/python

import sys;

from operator import itemgetter;

prev\_word = None

prev\_count = 0

word = None

for line in sys.stdin:

line = line.strip()

word,count = line.split('\t',1)

count=int(count)

if word == prev\_word:

prev\_count+=1;

else:

if prev\_word:

print('%s %s' %(prev\_word,prev\_count))

prev\_word = word;

prev\_count = count;

print('%s %s' %(prev\_word,prev\_count))

**Step 3:** Run the following command.

$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar **-file** /home/hduser/Desktop/bda\_lab/web\_log/my\_mapper.py **-mapper** “python my\_mapper.py” **-file** /home/hduser/Desktop/bda\_lab/web\_log/my\_reducer.py **-reducer** “python my\_reducer.py” **-input** /web\_log/inputdata **-output** /web\_log/outputdata2

**Step 4:** Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/

Goto utilities> Browse file System> /web\_log/outputdata2

**QUESTION 2**

Write a map reduce program that reads a weblog file to output the most number of referral sites.

**Steps for execution in python:**

**Step 1:** Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

$ hadoop fs -mkdir /web\_log

$hadoop fs -mkdir /web\_log/inputdata

Move the text file onto the hadoop store.

$ hadoop fs -put /home/hduser/Desktop/bda\_lab/web\_log/web\_log.txt /web\_log/inputdata

**Step 2:** Type in the mapper code and reducer code.

**my\_mapper.py**

#!/usr/bin/python

import sys;

import re;

for line in sys.stdin:

line = line.strip()

words=line.split()

x=re.findall("http://",words[10])

if (x):

print('%s %s' %(words[10],1))

**most\_referred\_reducer.py**

#!/usr/bin/python

import sys;

from operator import itemgetter;

prev\_word = None

prev\_count = 0

word = None

m=0

for line in sys.stdin:

line = line.strip()

word,count = line.split('\t',1)

count=int(count)

if word == prev\_word:

prev\_count+=1;

else:

if prev\_word and m<prev\_count:

most=prev\_word

m=prev\_count

prev\_word = word;

prev\_count = count;

if m<prev\_count:

most=prev\_word

m=prev\_count

print('%s %s' %(most,m))

**Step 3:** Run the following command.

$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar **-file** /home/hduser/Desktop/bda\_lab/web\_log/my\_mapper.py **-mapper** “python my\_mapper.py” **-file** /home/hduser/Desktop/bda\_lab/web\_log/most\_referred\_reducer.py **-reducer** “python most\_referred\_reducer.py” **-input** /web\_log/inputdata **-output** /web\_log/ most\_referred\_\_url

**Step 4:** Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/

Goto utilities> Browse file System> /web\_log/ most\_referred\_\_url

**QUESTION 3**

Write a map reduce program that reads a weblog file to output the number of client errors (with the Http status of 4XX)

**Steps for execution in python:**

**Step 1:** Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

$ hadoop fs -mkdir /web\_log

$hadoop fs -mkdir /web\_log/inputdata

Move the text file onto the hadoop store.

$ hadoop fs -put /home/hduser/Desktop/bda\_lab/web\_log/web\_log.txt /web\_log/inputdata

**Step 2:** Type in the mapper code and reducer code.

**no\_4xx\_errors\_mapper.py**

#!/usr/bin/python

import sys;

import re;

for line in sys.stdin:

line = line.strip()

words=line.split()

x=re.findall("http://",words[10])

if (x):

print('%s %s' %(words[10],1))

**my\_reducer.py**

#!/usr/bin/python

import sys;

from operator import itemgetter;

prev\_word = None

prev\_count = 0

word = None

for line in sys.stdin:

line = line.strip()

word,count = line.split('\t',1)

count=int(count)

if word == prev\_word:

prev\_count+=1;

else:

if prev\_word:

print('%s %s' %(prev\_word,prev\_count))

prev\_word = word;

prev\_count = count;

print('%s %s' %(prev\_word,prev\_count))

**Step 3:** Run the following command.

$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar **-file** /home/hduser/Desktop/bda\_lab/web\_log/no\_4xx\_errors\_mapper.py **-mapper** “python no\_4xx\_errors\_mapper.py” **-file** /home/hduser/Desktop/bda\_lab/web\_log/my\_reducer.py **-reducer** “python my\_reducer.py” **-input** /web\_log/inputdata **-output** /web\_log/o2

**Step 4:** Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/

Goto utilities> Browse file System> /web\_log/ o2