WordCount.java

```
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      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(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);
  }
}
```

Input: input.txt

WordCount example reads text files and counts how often words occur. The input is text files, and the output is text files, each line of which contains a word and the count of how often it occurred, separated by a tab.

Map Reduce Project that works on weather data and process it, the final outcome of the project can be processed further to find similarities on different weather stations

The Shadow was an American pulp magazine published by Street & Smith from 1931 to 1949. Each issue contained a novel about The Shadow, a mysterious crime-fighting figure who spoke the line "Who knows what evil lurks in the hearts of men? The Shadow knows" in radio broadcasts of stories from Street & Smith's Detective Story Magazine. For the first issue, dated April 1931, Walter Gibson wrote the lead novel,

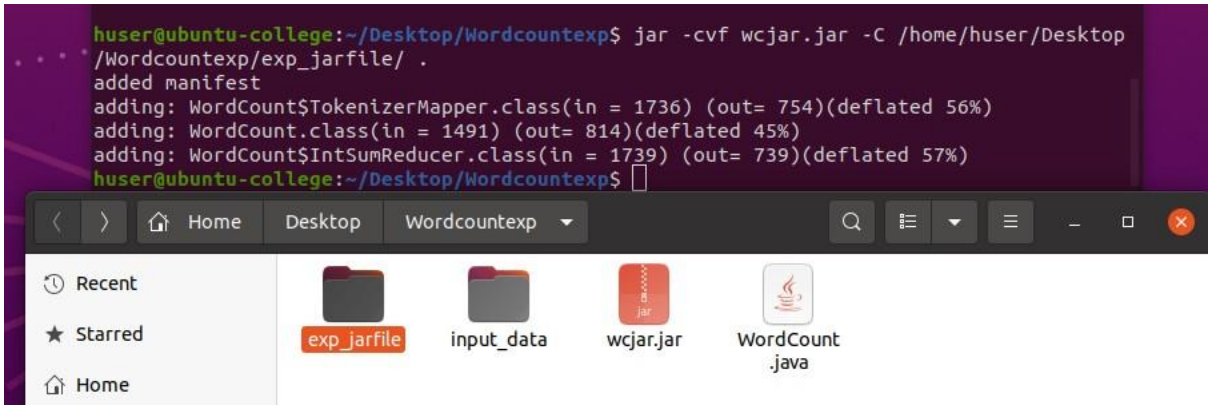
Output: output file (part-r-00000)

Wordcount Steps to run:

1. Starting Hadoop
   $ start-all.sh
2. Made A folder “wordcountexp” and write WordCount.java code.
3. Create new folder for input data.
4. Add input text file in the input data folder.
5. Create new folder to hold java class files.
6. Set HADOOP_CLASSPATH environment variable.
   $ export HADOOP_CLASSPATH=$(hadoop classpath)
7. Create a directory on HDFS
   $ hdfs dfs -mkdir /WordCountTut
   $ hdfs dfs -mkdir /WordCountTut/Input
8. Upload the input file (device) to that directory.
   $ hdfs dfs -put ‘/home/huser/Desktop/Wordcountexp/input_data/input.txt’ /WordCountTut/Input
9. Compile the java code:
   $ javac -classpath $(HADOOP_CLASSPATH) -d ‘/home/huser/Desktop/Wordcountexp/ exp_jarfile’ /home/huser/Desktop/Wordcountexp/ .*java

10.Creation .jar file of classes:

$ jar -cvf wcjar.jar -C '/home/huser/Desktop/Wordcountexp/exp_jarfile/ .



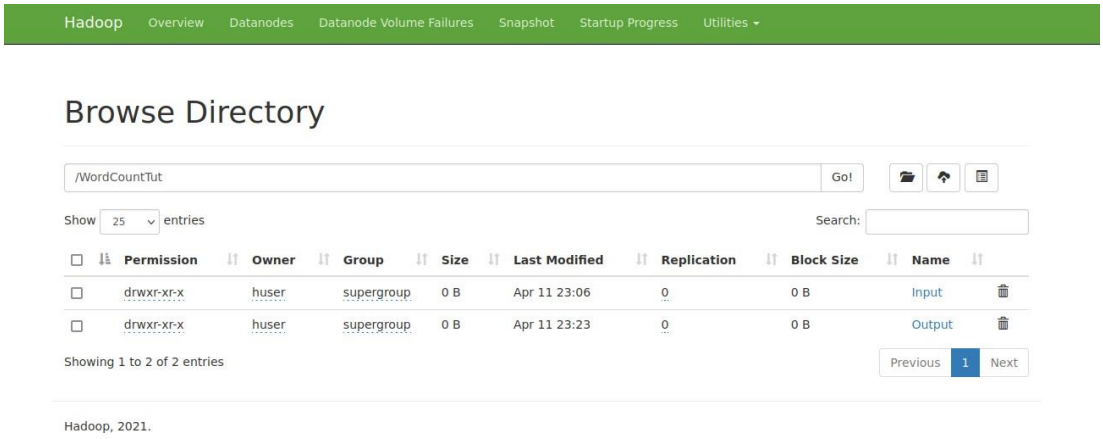
11.Running the jar file on Hadoop

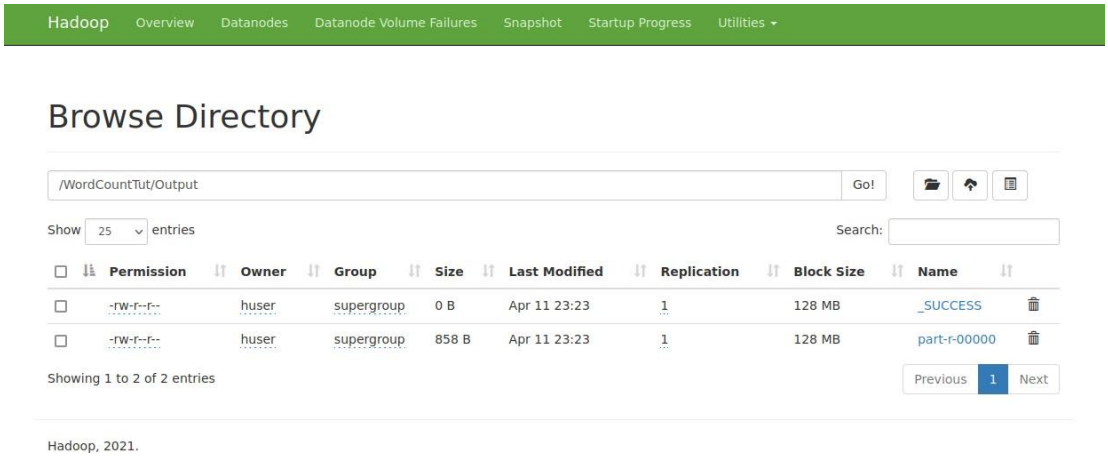
$ hadoop jar wcjar.jar WordCount /WordCountTut/Input /WordCountTut/Output

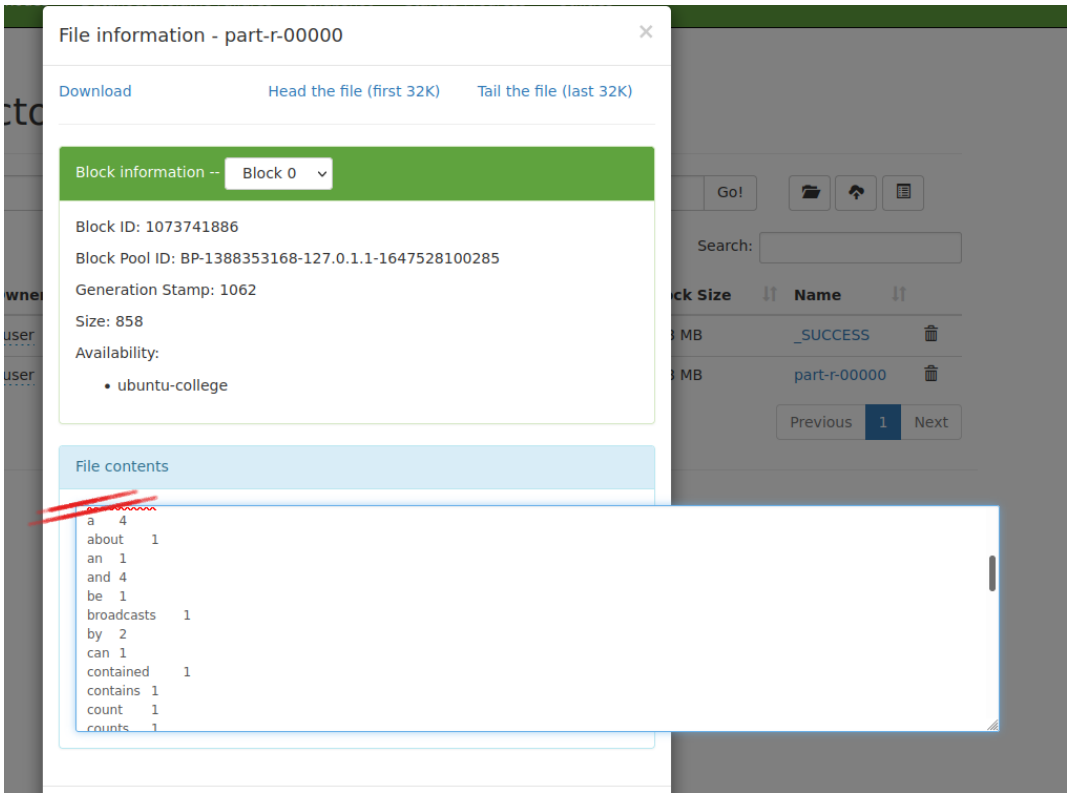
```
huser@ubuntu-college:~/Desktop/Wordcountexp$ hadoop jar wcjar.jar WordCount /WordCount
Tut/Input /WordCountTut/Output
2022-04-11 23:21:14,369 INFO client.RMProxy: Connecting to ResourceManager at /127.0.0
.1:8032
2022-04-11 23:21:17,291 WARN mapreduce.JobResourceUploader: Hadoop command-line option
 parsing not performed. Implement the Tool interface and execute your application with
 ToolRunner to remedy this.
2022-04-11 23:21:17,535 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding f
or path: /tmp/hadoop-yarn/staging/huser/.staging/job_1649697057322_0001
2022-04-11 23:21:18,170 INFO input.FileInputFormat: Total input files to process : 1
2022-04-11 23:21:18,286 INFO mapreduce.JobSubmitter: number of splits:1
2022-04-11 23:21:18,812 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_16
49697057322_0001
2022-04-11 23:21:18,813 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-04-11 23:21:19,296 INFO conf.Configuration: resource-types.xml not found
2022-04-11 23:21:19,296 INFO resource.ResourceUtils: Unable to find 'resource-types.xm
l'.
```

```
                Peak Reduce Physical memory (bytes)=167215104
                Peak Reduce Virtual memory (bytes)=2533072896
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=800
        File Output Format Counters
                Bytes Written=858
huser@ubuntu-college:~/Desktop/Wordcountexp$
```

12.Check output on localhost:9870 /localhost:50070







Conclusion: Thus, we successfully implement, WordCount application that counts the number of occurrence of each word in a given input set using the Hadoop MapReduce framework on local-standalone set-up.

Code:

1> LogFileMapper.java (Use for mapping the IP addresses from input csv file**)**

```
package LogFileCountry;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class LogFileMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text,
IntWritable> {
        private final static IntWritable one = new IntWritable(1);

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

                String valueString  =  value.toString();
                String[] SingleIpData = valueString.split("-");
                output.collect(new Text(SingleIpData[0]), one);
        }
}
```

2>LogFileReduce.java (Use for reducing data received from mapper process to final output**)**

```
package LogFileCountry;
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.io.IntWritable;
import   org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class LogFileReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text,
IntWritable> {
```

```
	public void reduce(Text t_key, Iterator<IntWritable> values, OutputCollector<Text,IntWritable>
output, Reporter reporter) throws IOException {
		Text key = t_key;
		int frequencyForIp = 0;
		while (values.hasNext()) {
			// replace type of value with the actual type of our value
			IntWritable value = (IntWritable) values.next();
			frequencyForIp += value.get();
		}
		output.collect(key, new IntWritable(frequencyForIp));
	}
}
```

3>LogFileCountryDriver.java (The driver code to run map-reduce on hdfs)

```
package LogFileCountry;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class LogFileCountryDriver {
	public static void main(String[] args) {
		JobClient my_client = new JobClient();
		// Create a configuration object for the job
		JobConf job_conf = new JobConf(LogFileCountryDriver.class);

		// Set a name of the Job
		job_conf.setJobName("LogFileIP");
		// Specify data type of output key and value
		job_conf.setOutputKeyClass(Text.class);
		job_conf.setOutputValueClass(IntWritable.class);
		// Specify names of Mapper and Reducer Class
		job_conf.setMapperClass(LogFileCountry.LogFileMapper.class);
		job_conf.setReducerClass(LogFileCountry.LogFileReducer.class);

		// Specify formats of the data type of Input and output
		job_conf.setInputFormat(TextInputFormat.class);
		job_conf.setOutputFormat(TextOutputFormat.class);

		// Set input and output directories using command line arguments,
		//arg[0] = name of input directory on HDFS, and arg[1] = name of output directory to be
created to store the output file.

		FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
		FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

		my_client.setConf(job_conf);
		try {  // Run the job
```

```
                JobClient.runJob(job_conf);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

4> **log_file.txt** (Input file sample)

```
0.223.157.186 - - [15/Jul/2009:20:50:32 -0700] "GET /assets/js/the-associates.js HTTP/1.1" 304 -
10.223.157.186 - - [15/Jul/2009:20:50:33 -0700] "GET /assets/img/home-logo.png HTTP/1.1" 304 -
10.223.157.186 - - [15/Jul/2009:20:50:33 -0700] "GET /assets/img/dummy/primary-news-2.jpg HTTP/1.1" 304 -
10.223.157.186 - - [15/Jul/2009:20:50:33 -0700] "GET /assets/img/dummy/primary-news-1.jpg HTTP1.1" 304 -
10.223.157.186 - - [15/Jul/2009:20:50:33 -0700] "GET /assets/img/home-media-block-placeholder.jpg HTTP/1.1" 304
-
10.223.157.186 - - [15/Jul/2009:20:50:33 -0700] "GET /assets/img/dummy/secondary-news-4.jpg HTTP/1.1" 304 -
10.223.157.186 - - [15/Jul/2009:20:50:33 -0700] "GET /assets/img/loading.gif HTTP/1.1" 304 -
10.223.157.186 - - [15/Jul/2009:20:50:33 -0700] "GET /assets/img/search-button.gif HTTP/1.1" 304 –
```

5> **Output** (part-00000.txt On Hadoop) (sample)

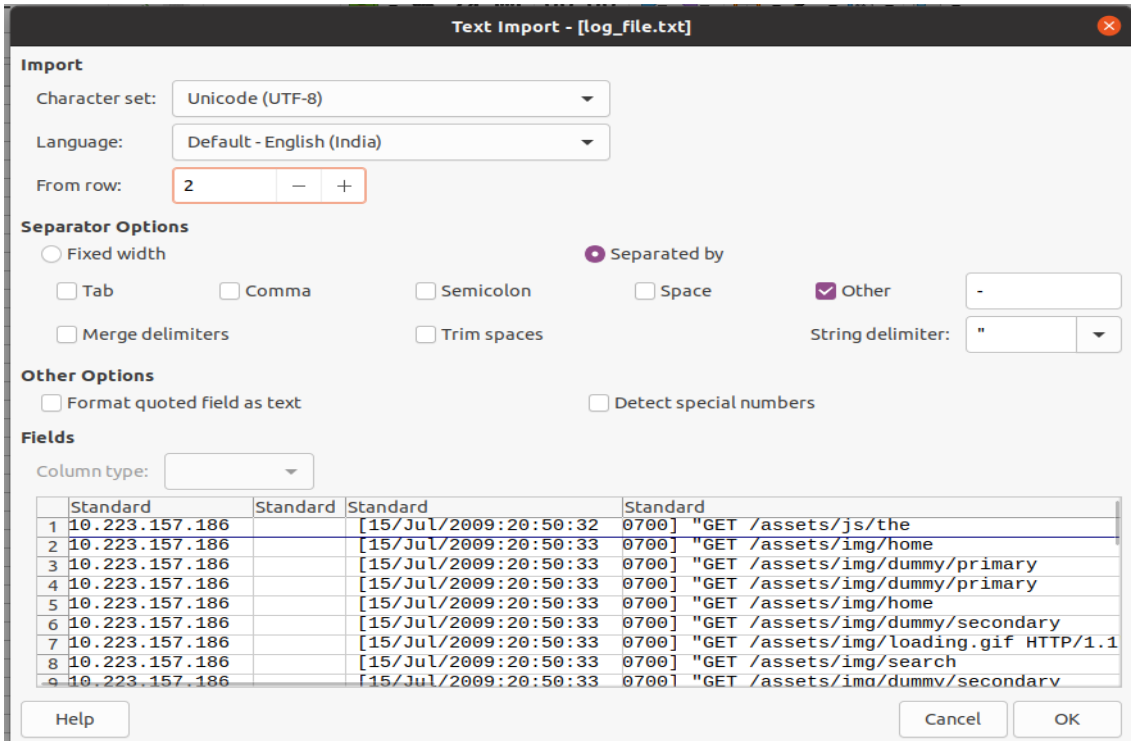
```
10.1.1.236      7
10.1.181.142    14
10.1.232.31     5
10.10.55.142    14
10.102.101.66   1
10.103.184.104  1
10.103.190.81   53
10.103.63.29    1
10.104.73.51    1
10.105.160.183  1
10.108.91.151   15
10.109.21.76    1
10.11.131.40    1
10.111.71.20    8
10.112.227.184  6
10.114.74.30    1
10.115.118.78   1
10.117.224.230  1
```

Step For Logs File Code:

13. Starting Hadoop and check if it is started.

    $ start-all.sh

14. Create folder “LogFileTut”. Copy the log_file.txt given and create the java files.
    i. LogFileMapper.java
    ii. LogFileReducer.java
    iii. LogFileCountryDriver.java
15. Convert the log_file.txt to .csv file. Open LibreOffice Calc-> Open -> log_file.txt. Save As .csv in the LogFileTut folder.



16. Give Read permission to all the files in directories.

    $ sudo chmod +r *.*

17. Set HADOOP_CLASSPATH environment variable.

    $ export HADOOP_CLASSPATH=$(hadoop classpath)

    or

    $ export CLASSPATH=
    "$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-core-3.2.2.jar:
    $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-common-3.2.2.jar:
    $HADOOP_HOME/share/hadoop/common/hadoop-common-3.2.2.jar: $HADOOP_HOME/lib/*:
    ~/home/huser/Desktop/LogFileTut/*"

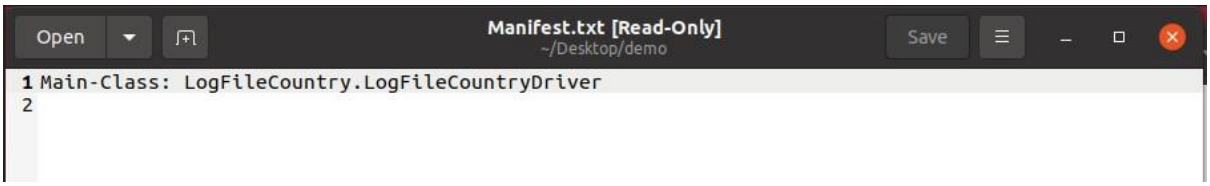
18. Compile the java code:

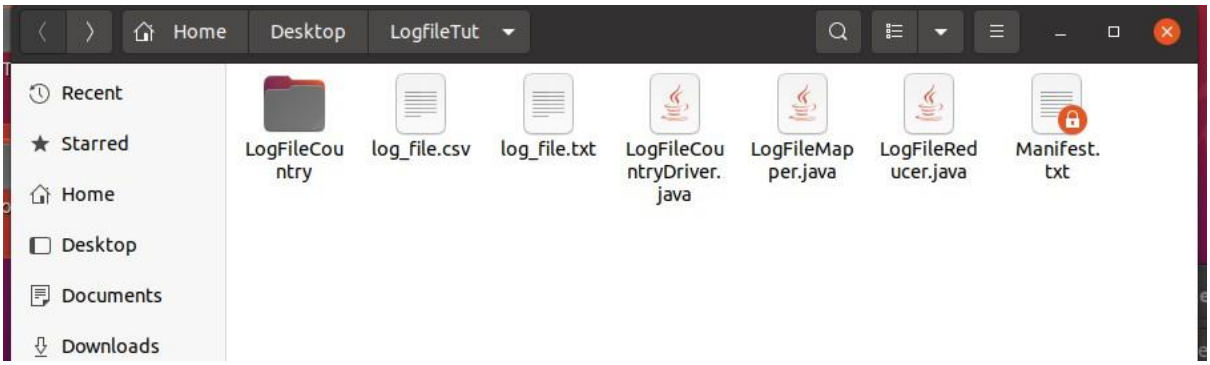
    $ javac -classpath $(HADOOP_CLASSPATH) -d ‘/home/huser/Desktop/LogFileTut/exp_classfile
    .*java

19. Create Manifest.txt file.

```
huser@ubuntu-college:~/Desktop/LogfileTut$ javac -d '/home/huser/Desktop/LogfileTut/exp_classfile' LogFileMa
pper.java LogFileReducer.java LogFileCountryDriver.java
huser@ubuntu-college:~/Desktop/LogfileTut$ sudo gedit Manifest.txt
[sudo] password for huser:

(gedit:59507): Tepl-WARNING **: 22:24:16.402: GVfs metadata is not supported. Fallback to TeplMetadataManage
r. Either GVfs is not correctly installed or GVfs metadata are not supported on this platform. In the latter
 case, you should configure Tepl with --disable-gvfs-metadata.
```

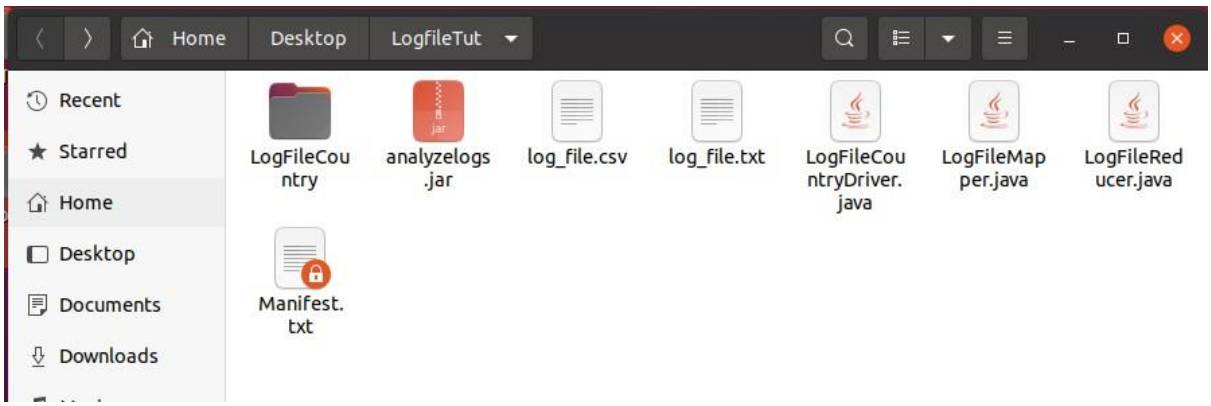




20. Creation .jar file of classes:

$ jar -cvfm analyzelogs.jar Manifest.txt LogFileCountry/*.class

```
huser@ubuntu-college:~/Desktop/LogfileTut$ jar -cvfm analyzelogs.jar Manifest.txt LogFileCountry/*.class
added manifest
adding: LogFileCountry/LogFileCountryDriver.class(in = 1677) (out= 825)(deflated 50%)
adding: LogFileCountry/LogFileMapper.class(in = 1713) (out= 645)(deflated 62%)
adding: LogFileCountry/LogFileReducer.class(in = 1580) (out= 635)(deflated 59%)
```



21. Create a directory on HDFS .And check on localhost:9870

$ hdfs dfs -mkdir / LogFileExp

$ hdfs dfs -mkdir / LogFileExp/Input

$ hdfs dfs -mkdir / LogFileExp/Output

22. Upload the log_file.csv in hadoop dir /LogFileExp/Input

$ hdfs dfs -put '/home/huser/Desktop/LogFileTut/log_file.csv' /LogFileExp/Input

23. Running the jar file on Hadoop.

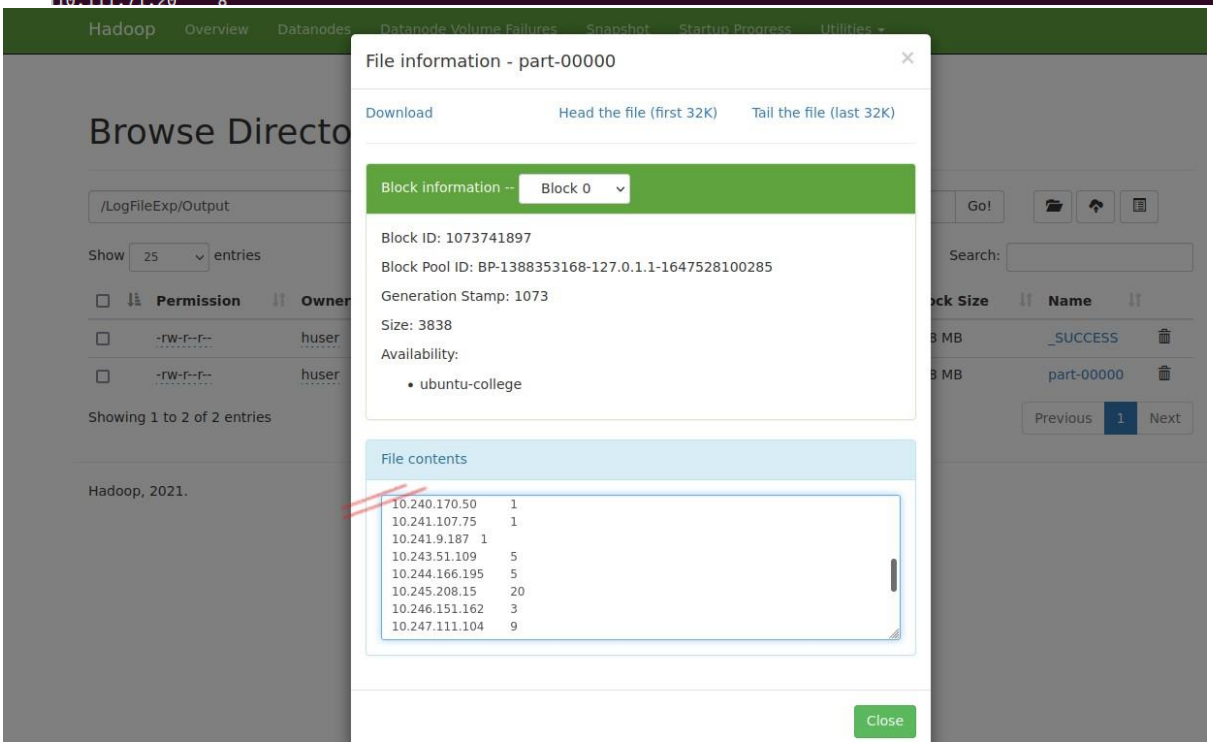
$ hadoop jar analyzelogs.jar /LogFileExp/Input /LogFileExp/Output

```
huser@ubuntu-college:~/Desktop/LogfileTut$ hadoop jar analyzelogs.jar /LogFileExp/Input /LogFileExp/Output
2022-04-12 22:51:25,988 INFO client.RMProxy: Connecting to ResourceManager at /127.0.0.1:8032
2022-04-12 22:51:27,403 INFO client.RMProxy: Connecting to ResourceManager at /127.0.0.1:8032
2022-04-12 22:51:35,208 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement th
e Tool interface and execute your application with ToolRunner to remedy this.
2022-04-12 22:51:36,276 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/hu
ser/.staging/job_1649777619248_0001
2022-04-12 22:51:39,085 INFO mapred.FileInputFormat: Total input files to process : 1
2022-04-12 22:51:40,281 INFO mapreduce.JobSubmitter: number of splits:2
2022-04-12 22:51:40,821 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1649777619248_0001
2022-04-12 22:51:40,823 INFO mapreduce.JobSubmitter: Executing with tokens: []
2022-04-12 22:51:42,337 INFO conf.Configuration: resource-types.xml not found
2022-04-12 22:51:42,337 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2022-04-12 22:52:55,956 INFO impl.YarnClientImpl: Submitted application application_1649777619248_0001
2022-04-12 22:52:58,540 INFO mapreduce.Job: The url to track the job: http://ubuntu-college:8088/proxy/application_1649777
619248_0001/
2022-04-12 22:52:58,584 INFO mapreduce.Job: Running job: job_1649777619248_0001
2022-04-12 22:57:02,946 INFO mapreduce.Job: Job job_1649777619248_0001 running in uber mode : false
2022-04-12 22:57:03,272 INFO mapreduce.Job:  map 0% reduce 0%
2022-04-12 23:00:09,974 INFO mapreduce.Job:  map 83% reduce 0%
2022-04-12 23:00:27,106 INFO mapreduce.Job:  map 100% reduce 0%
2022-04-12 23:01:05,301 INFO mapreduce.Job:  map 100% reduce 100%
2022-04-12 23:01:08,520 INFO mapreduce.Job: Job job_1649777619248_0001 completed successfully
2022-04-12 23:01:24,647 INFO mapreduce.Job: Counters: 54
```

24. Check the Output file.

$ hdfs dfs -cat /LogFileExp/Output/part-00000

```
huser@ubuntu-college:~/Desktop/LogfileTut$ hdfs dfs -cat /LogFileExp/Output/part-00000
10.1.1.236      7
10.1.181.142    14
10.1.232.31     5
10.10.55.142    14
10.102.101.66   1
10.103.184.104  1
10.103.190.81   53
10.103.63.29    1
10.104.73.51    1
10.105.160.183  1
10.108.91.151   1
10.109.21.76    1
10.11.131.40    1
10.111.71.20    8
```



25. Stop all processes :

$ stop-all.sh

Conclusion: Thus, we successfully implement, distributed application using MapReduce which processes a log file of a system.

GROUP B

EXPERIMENT: 3

AIM: Locate dataset (e.g., sample_weather.txt) for working on weather data which reads the text input files and finds average for temperature, dew point and wind speed.

PRE-REQUISITE:

- Java Installation -check whether the Java is installed or not using the following command.
  java -version
- Hadoop Installation -Check whether the Hadoop is installed or not using the following command.
  hadoop -version

THEORY:

Hadoop does distributed processing for huge data sets across the cluster of commodity servers and works on multiple machines simultaneously. To process any data, the client submits data and program to Hadoop. HDFS stores the data while MapReduce process the data and Yam divide the tasks.
Let's discuss in detail how Hadoop works –

i HDFS

- Hadoop Distributed File System has master-slave topology.
- It has got two daemons running, they are NameNode and DataNode.

NameNode

- NameNode is the daemon running of the master machine. It is the centerpiece of an HDFS file system.
- NameNode stores the directory tree of all files in the file system. It tracks where across the cluster the file data resides. It does not store the data contained in these files.
- When the client applications want to add/copy/move/delete a file, they interact with NameNode.
- The NameNode responds to the request from client by returning a list of relevant DataNode servers where the data lives.

DataNode

- DataNode daemon runs on the slave nodes. It stores data in the HadoopFileSystem. In functional file system data replicates across many DataNodes.
- On startup, a DataNode connects to the NameNode. It keeps on looking for the request from NameNode to access data. Once the NameNode provides the location of the data, client applications can talk directly to a DataNode, while replicating the data, DataNode instances can talk to each other.

Replica Placement

- The placement of replica decides HDFS reliability and performance. Optimization of replica placement makes HDFS apart from other distributed system. Huge HDFS instances run on a cluster of computers spreads across many racks. The communication between nodes on different racks has to go through the switches. Mostly the network bandwidth between nodes on the same rack is more than that between the machines on separate racks.

- The rack awareness algorithm determines the rack id of each DataNode. Under a simple policy, the replicas get placed on unique racks. This prevents data loss in the event of rack

failure. Also, it utilizes bandwidth from multiple racks while reading data. However, this method increases the cost of writes.

- Let us assume that the replication factor is three. Suppose HDFS's placement policy places one replica on a local rack and other two replicas on the remote but same rack. This policy cuts the inter-rack write traffic thereby improving the write performance. The chances of rack failure are less than that of node failure. Hence this policy does not affect data reliability and availability. But, it does reduce the aggregate network bandwidth used when reading data. This is because a block gets placed in only two unique racks rather than three.

ii. MapReduce

- The general idea of the MapReduce algorithm is to process the data in parallel on your distributed cluster. It subsequently combine it into the desired result or output.

Hadoop MapReduce includes several stages:

- In the first step, the program locates and reads the « input file » containing the raw data.
- As the file format is arbitrary, there is a need to convert data into something the program can process. The « InputFormat » and « RecordReader » (RR) does this job.
  InputFormat uses InputSplit function to split the file into smaller pieces
  Then the RecordReader transforms the raw data for processing by the map. It outputs a list of key-value pairs.
  Once the mapper process these key-value pairs the result goes to « OutputCollector ». There is another function called « Reporter » which intimates the user when the mapping task finishes.
- In the next step, the Reduce function performs its task on each key-value pair from the mapper.
- Finally, OutputFormat organizes the key-value pairs from Reducer for writing it on HDFS.
- Being the heart of the Hadoop system, Map-Reduce process the data in a highly resilient, fault-tolerant manner.

iii. Yarn

- Yarn divides the task on resource management and job scheduling/monitoring into separate daemons.
- There is one ResourceManager and per-application ApplicationMaster.
- An application can be either a job or a DAG of jobs.

The Resource Manager have two components – Scheduler and Application Manager
The scheduler is a pure scheduler i.e. it does not track the status of running application. It only allocates resources to various competing applications. Also, it does not restart the job after failure due to hardware or application failure. The scheduler allocates the resources based on an abstract notion of a container. A container is nothing but a fraction of resources like CPU, memory, disk, network etc.

Following are the tasks of Application Manager:-

- Accepts submission of jobs by client.
- Negotiates first container for specific Application Master.
- Restarts the container after application failure.

Weather.java

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.*;
/**
 * This is an Hadoop Map/Reduce application for Working on weather data It reads
 * the text input files, breaks each line into stations weather data and finds
 * average for temperature, dew point , wind speed. The output is a locally
 * sorted list of stations and its 12 attribute vector of average temp , dew ,
 * wind speed of 4 sections for each month.
*/
public class Weather extends Configured implements Tool {
        final long DEFAULT_SPLIT_SIZE = 128 * 1024 * 1024;
        /**
         * Map Class for Job 1
         * For each line of input, emits key value pair with
         * station_yearmonth_sectionno as key and 3 attribute vector with
         * temperature , dew point , wind speed as value.Map method will strip the
         * day and hour from field and replace it with section no (
         * <b>station_yearmonth_sectionno</b>, <b><temperature,dew point , wind
```

```
 * speed></b>).
 */
public static class MapClass extends MapReduceBase
                implements Mapper<LongWritable, Text, Text, Text> {
        private Text word = new Text();
        private Text values = new Text();
        public void map(LongWritable key, Text value,
                                OutputCollector<Text, Text> output,
                                Reporter reporter) throws IOException {
                String line = value.toString();
                StringTokenizer itr = new StringTokenizer(line);
                int counter = 0;
                String key_out = null;
                String value_str = null;
                boolean skip = false;
                loop:while (itr.hasMoreTokens() && counter<13) {
                        String str = itr.nextToken();
                        switch (counter) {
                        case 0:
                                key_out = str;
                                if(str.contains("STN")){//Ignoring rows where stationid is all 9
                                        skip = true;
                                        break loop;
                                }else{
                                        break;
                                }
                        case 2:
                                int hour = Integer.valueOf(str.substring(str.lastIndexOf("_")+1, str.length()));
                                str = str.substring(4,str.lastIndexOf("_")-2);
                                if(hour>4 && hour<=10){
                                        str = str.concat("_section1");
                                }else if(hour>10 && hour<=16){
                                        str = str.concat("_section2");
                                }else if(hour>16 && hour<=22){
                                        str = str.concat("_section3");
                                } else{ str = str.concat("_section4");
                                }

                                key_out = key_out.concat("_").concat(str);
                                break;
                        case 3://Temperature
                                if(str.equals("9999.9")){//Ignoring rows temperature is all 9
                                        skip = true;
                                        break loop;
                                }else{
                                        value_str = str.concat(" ");
                                        break;
                                }
                        case 4://Dew point
                                if(str.equals("9999.9")){//Ignoring rows where dewpoint all 9
                                        skip = true;
                                        break loop;
```

```
                                        }else{
                                                value_str = value_str.concat(str).concat(" ");
                                                break;
                                        }
                                case 12://Wind speed
                                        if(str.equals("999.9")){//Ignoring rows wind speed is all 9
                                                skip = true;
                                                break loop;
                                        }else{ value_str = value_str.concat(str).concat(" ");
                                                break;
                                        }
                                default: break;
                                }
                                counter++;
                        }
                        if(!skip){
                                word.set(key_out);
                                values.set(value_str);
                                output.collect(word, values);
                        }
                }
        }
        /**
         * Reducer Class for Job 1
         * A reducer class that just emits 3 attribute vector with average
         * temperature , dew point , wind speed for each of the section of the month for each input
         */
        public static class Reduce extends MapReduceBase
                        implements Reducer<Text, Text, Text, Text> {
                private Text value_out_text = new Text();
                public void reduce(Text key, Iterator<Text> values,
                                OutputCollector<Text, Text> output, Reporter reporter) throws IOException {
                        double sum_temp = 0;
                        double sum_dew = 0;
                        double sum_wind = 0;
                        int count = 0;

                        while (values.hasNext()) {
                                String str = values.next().toString();
                                StringTokenizer itr = new StringTokenizer(str);
                                int count_vector = 0;
                                while (itr.hasMoreTokens()) {
                                        String nextToken = itr.nextToken(" ");
                                        if(count_vector==0){
                                                sum_temp += Double.valueOf(nextToken);
                                        }
                                        if(count_vector==1){
                                                sum_dew += Double.valueOf(nextToken);
                                        }
                                        if(count_vector==2){
                                                sum_wind += Double.valueOf(nextToken);
                                        }
```

```
                                count_vector++;
                        }
                        count++;
                }
                double avg_tmp = sum_temp / count;
                double avg_dew = sum_dew / count;
                double avg_wind = sum_wind / count;
                System.out.println(key.toString()+" count is "+count+" sum of temp is "+sum_temp+" sum of
dew is "+sum_dew+" sum of wind is "+sum_wind+"\n");
                String value_out = String.valueOf(avg_tmp).concat("
").concat(String.valueOf(avg_dew)).concat(" ").concat(String.valueOf(avg_wind));
                value_out_text.set(value_out);
                output.collect(key, value_out_text);
        }
    }
    static int printUsage() {
        System.out.println("weather [-m <maps>] [-r <reduces>] <job_1 input> <job_1 output> <job_2
output>");
        ToolRunner.printGenericCommandUsage(System.out);
        return -1;
    }
    /**
     * The main driver for weather map/reduce program.
     * Invoke this method to submit the map/reduce job.
     * @throws IOException When there is communication problems with the job tracker.
     */
    public int run(String[] args) throws Exception {
        Configuration config = getConf();
        // We need to lower input block size by factor of two
        JobConf conf = new JobConf(config, Weather.class);
        conf.setJobName("Weather Job1");

        // the keys are words (strings)
        conf.setOutputKeyClass(Text.class);
        // the values are counts (ints)
        conf.setOutputValueClass(Text.class);

        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(Text.class);
        conf.setMapperClass(MapClass.class);
        //conf.setCombinerClass(Combiner.class);
        conf.setReducerClass(Reduce.class);
        List<String> other_args = new ArrayList<String>();
        for(int i=0; i < args.length; ++i) {
            try {
                if ("-m".equals(args[i])) {
                    conf.setNumMapTasks(Integer.parseInt(args[++i]));
                } else if ("-r".equals(args[i])) {
                    conf.setNumReduceTasks(Integer.parseInt(args[++i]));
                } else {
                    other_args.add(args[i]);
                }
```

```
                } catch (NumberFormatException except) {
                        System.out.println("ERROR: Integer expected instead of " + args[i]);
                        return printUsage();
                } catch (ArrayIndexOutOfBoundsException except) {
                        System.out.println("ERROR: Required parameter missing from " +
                                args[i-1]);
                        return printUsage();
                }
        }
        // Make sure there are exactly 2 parameters left.
        FileInputFormat.setInputPaths(conf, other_args.get(0));
        FileOutputFormat.setOutputPath(conf, new Path(other_args.get(1)));
        JobClient.runJob(conf);
        return 0;
    }
    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new Weather(), args);
        System.exit(res);
    }
}
```

Input: sample_weather.txt (sample)

```
690190 13910 20060201_0 51.75   33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 000000
690190 13910 20060201_1 54.74   33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 00000
690190 13910 20060201_2 50.59   33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 00000
690190 13910 20060201_3 51.67   33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 000000
690190 13910 20060201_4 65.67   33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 000000
690190 13910 20060201_5 55.37   33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 000000
690190 13910 20060201_6 49.26   33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 000000
690190 13910 20060201_7 55.44   33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 000000
690190 13910 20060201_8 64.05   33.0 24 1006.3 24 943.9 24 15.0 24 10.7 24 22.0 28.9 0.00I 999.9 000000
```

Output: part-00000.txt (on Hadoop)

```
690190_02_section1  53.87166666666666 25.899999999999995 7.774999999999998
690190_02_section2  54.76125000000001 25.900000000000006 7.774999999999999
690190_02_section3  53.25041666666667 25.899999999999995 7.774999999999996
690190_02_section4  52.44708333333333 25.900000000000006 7.774999999999999
```

Weather Data Analysis Steps to run:

26. Starting Hadoop

    $ start-all.sh

27. Made A folder “WeatherAssi” and write Weather.java code.
28. Create new folder for input data.
29. Add input text file in the input data folder.
30. Create new folder to hold java class files.
31. Set HADOOP_CLASSPATH environment variable.

    $ export HADOOP_CLASSPATH=$(hadoop classpath)

32. Create a directory on HDFS

$ hdfs dfs -mkdir /WeatherTut

$ hdfs dfs -mkdir /WeatherTut/Input

33. Checking on localhost:9870

34. Upload the input file (device) to that directory.

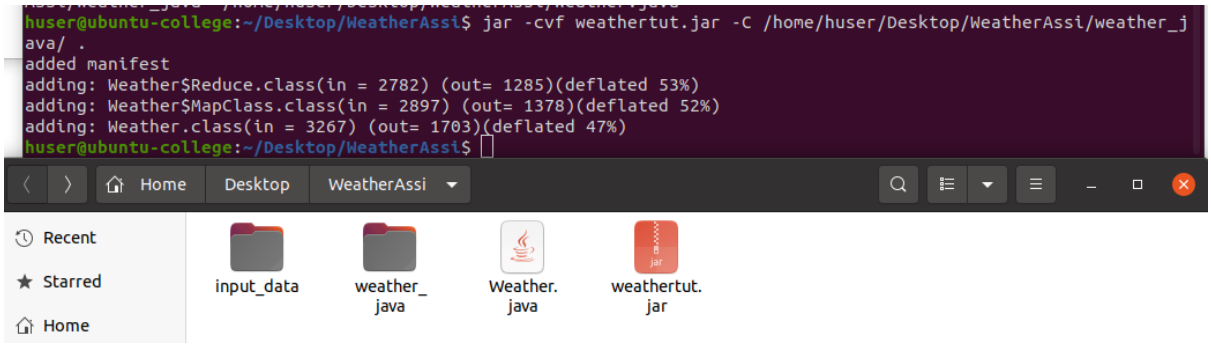
$ hdfs dfs -put input_data/sample_wheater.txt /WeatherTuT/Input

35. Compile the java code:

$ javac -classpath $(HADOOP_CLASSPATH) -d '/home/huser/Desktop/WeatherAssi/ weather_java' /home/huser/Desktop/WeatherAssi/Weather.java

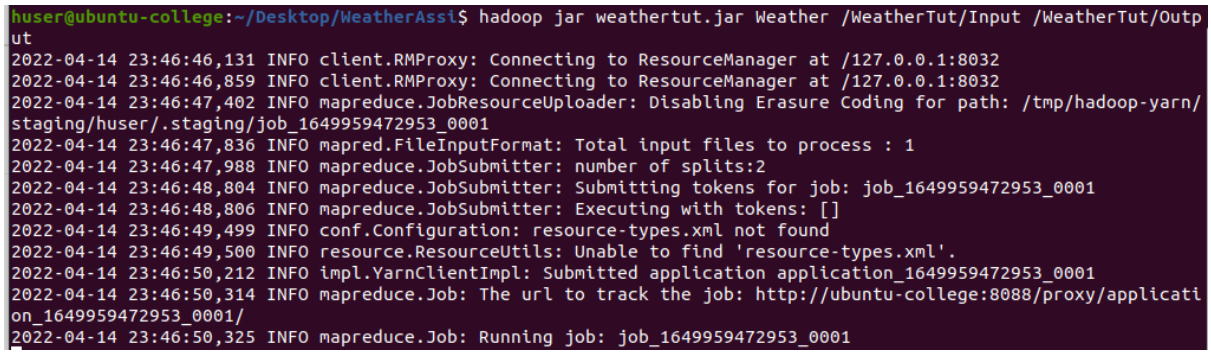
36. Creation .jar file of classes:

$ jar -cvf weathertut.jar -C /home/huser/Desktop/WeatherAssi/weather_java/ .

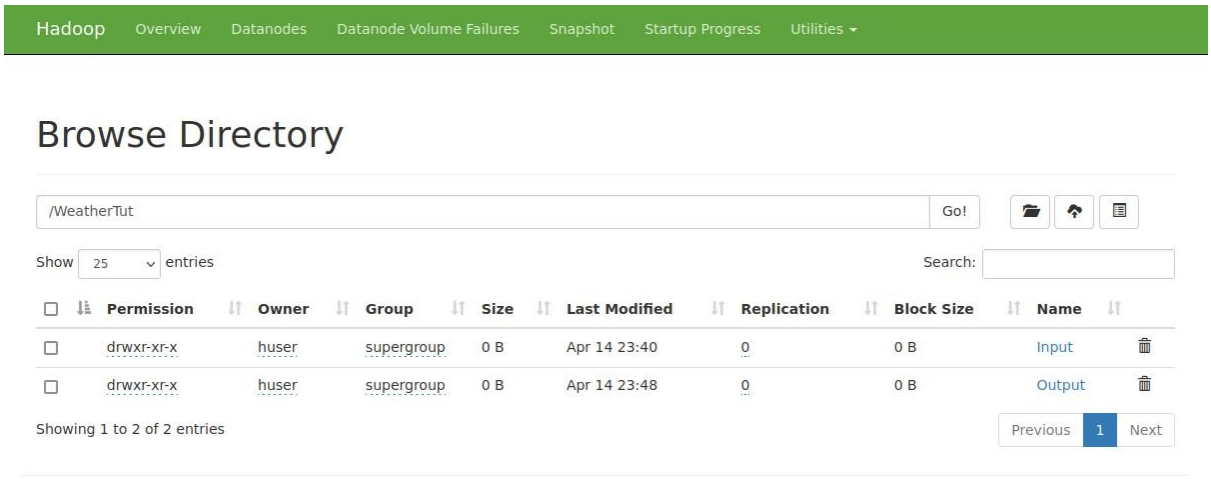


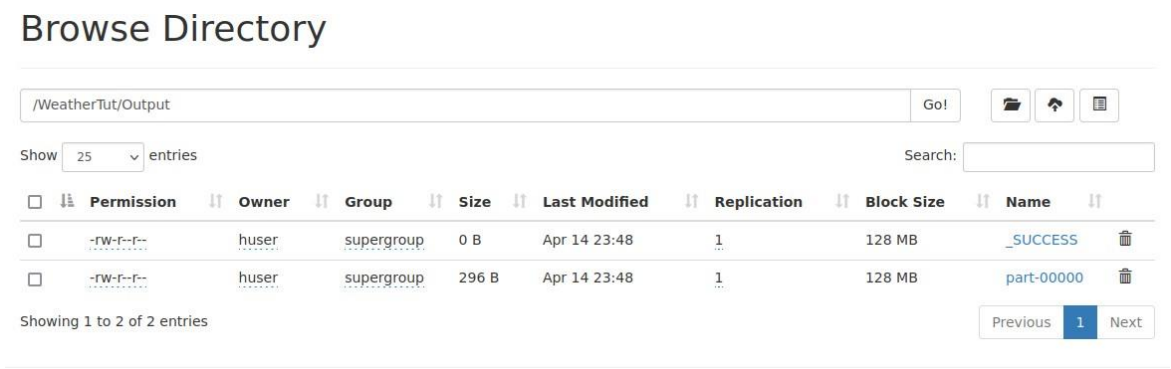
37. Running the jar file on Hadoop

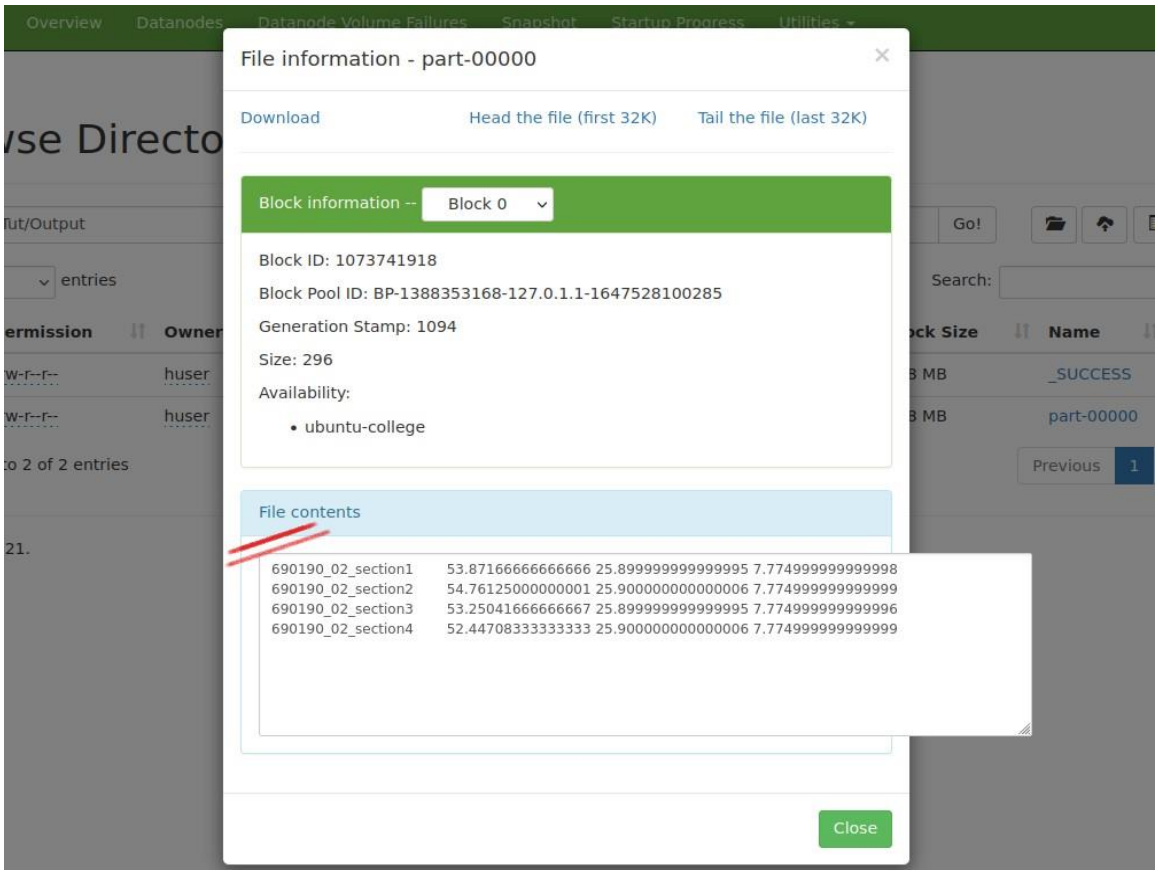
$ hadoop jar weather.jar Weather /WeatherTut/Input /WeatherTut/Ouput



38. Check output on localhost:9870 /localhost:50070







39.Stop Hadoop services:

$ stop-all.sh

Conclusion: Thus, we successfully, Locate dataset (e.g., sample_weather.txt) for working on weather data which reads the text input files and finds average for temperature, dew point and wind speed.