GROUP B

EXPERIMENT : 11

AIM: Write a code in JAVA for a simple WordCount application that counts the number of occurrences of each word in a given input set using the Hadoop MapReduce framework on local-standalone set-up.

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
  following
  hadoop version

THEORY:

Steps to execute MapReduce word count :

- Create a text file in your local machine and write some text into it.
  $ nano data.txt
- Check the text written in the data.txt file.
  $ cat data.txt

File: WC_Mapper.java

```
package com.javatpoint;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
```

```

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

    public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output,
           Reporter reporter) throws IOException{
        String line = value.toString();
        StringTokenizer  tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()){
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
}
```

File: WC_Reducer.java

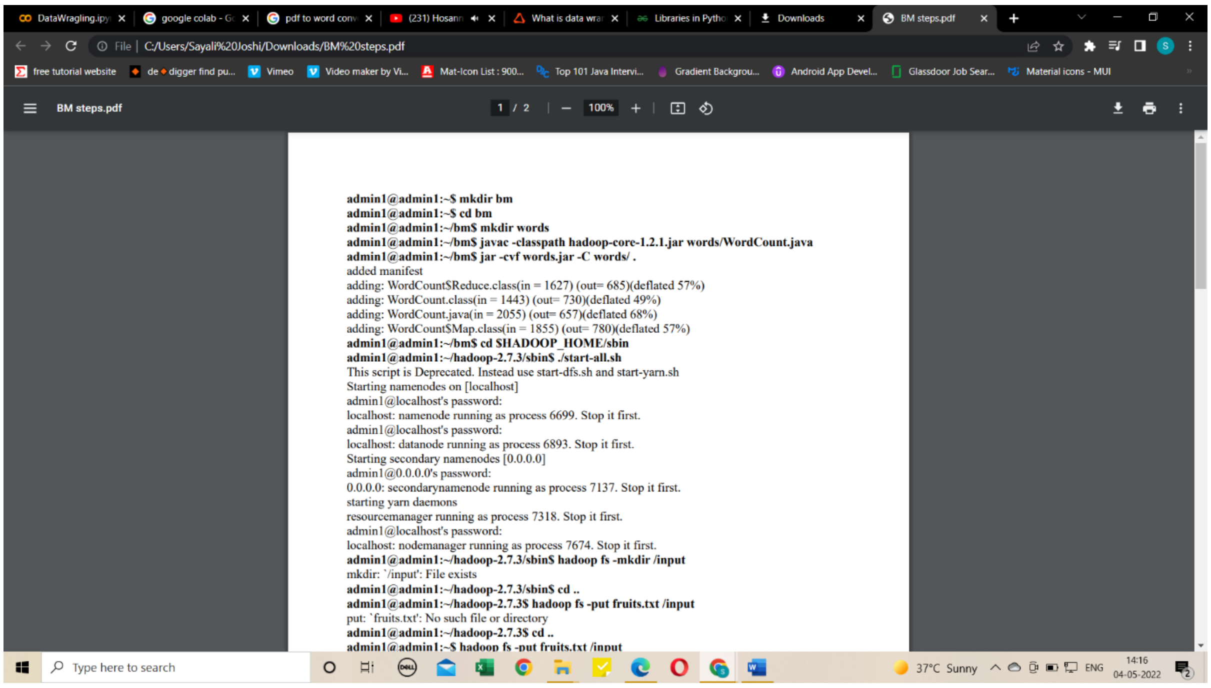
```
package com.javatpoint;
    import java.io.IOException;
    import java.util.Iterator;
    import org.apache.hadoop.io.IntWritable;
    import org.apache.hadoop.io.Text;
    import org.apache.hadoop.mapred.MapReduceBase;
    import org.apache.hadoop.mapred.OutputCollector;
    import org.apache.hadoop.mapred.Reducer;
    import org.apache.hadoop.mapred.Reporter;


    public class WC_Reducer  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 sum=0;
while (values.hasNext()) {
sum+=values.next().get();
}
output.collect(key,new IntWritable(sum));
}
}
```

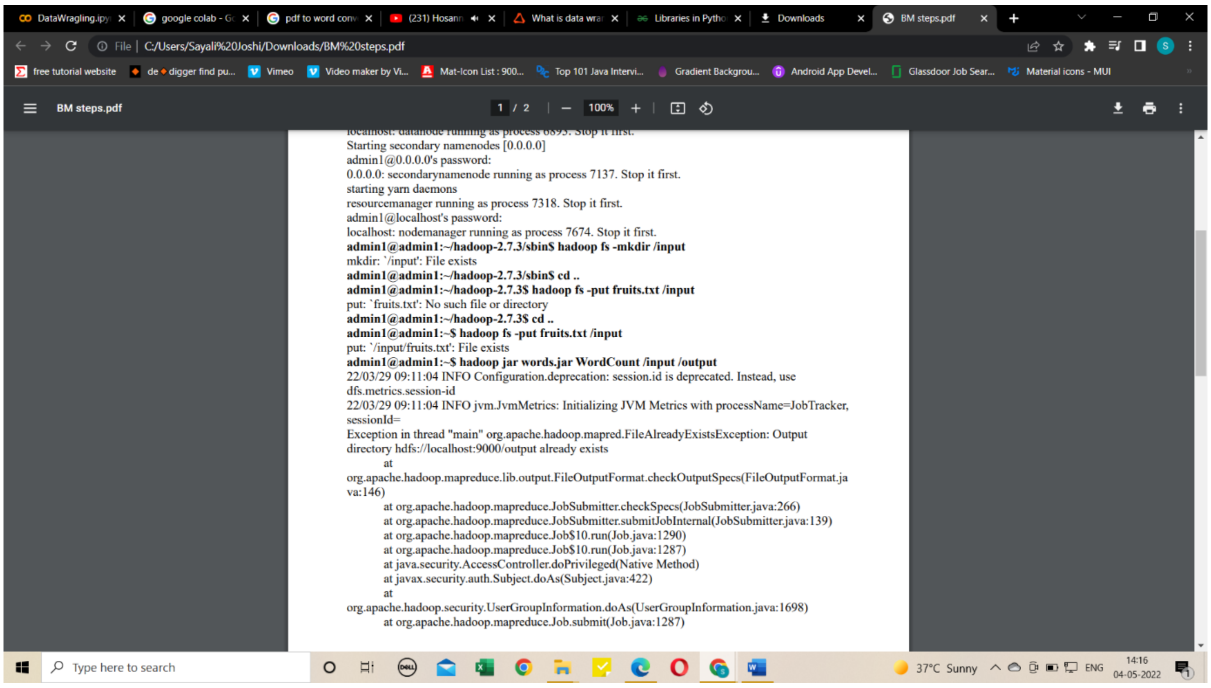
File: WC_Runner.java

```
package com.javatpoint;

import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
public class WC_Runner {
    public static void main(String[] args) throws IOException{
        JobConf conf = new JobConf(WC_Runner.class);
        conf.setJobName("WordCount");
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);
        conf.setMapperClass(WC_Mapper.class);
        conf.setCombinerClass(WC_Reducer.class);
        conf.setReducerClass(WC_Reducer.class);
        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);
       FileInputFormat.setInputPaths(conf,new Path(args[0]));
        FileOutputFormat.setOutputPath(conf,new Path(args[1]));
       JobClient.runJob(conf);
    }
}
```

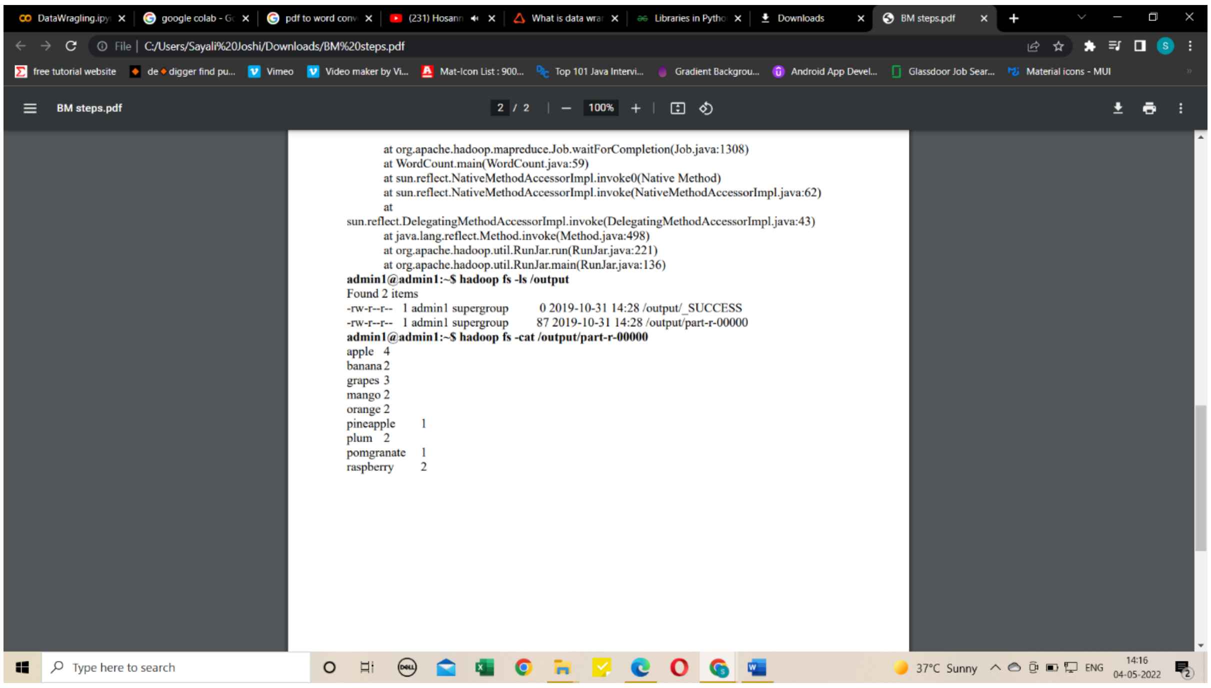
RESULTS:



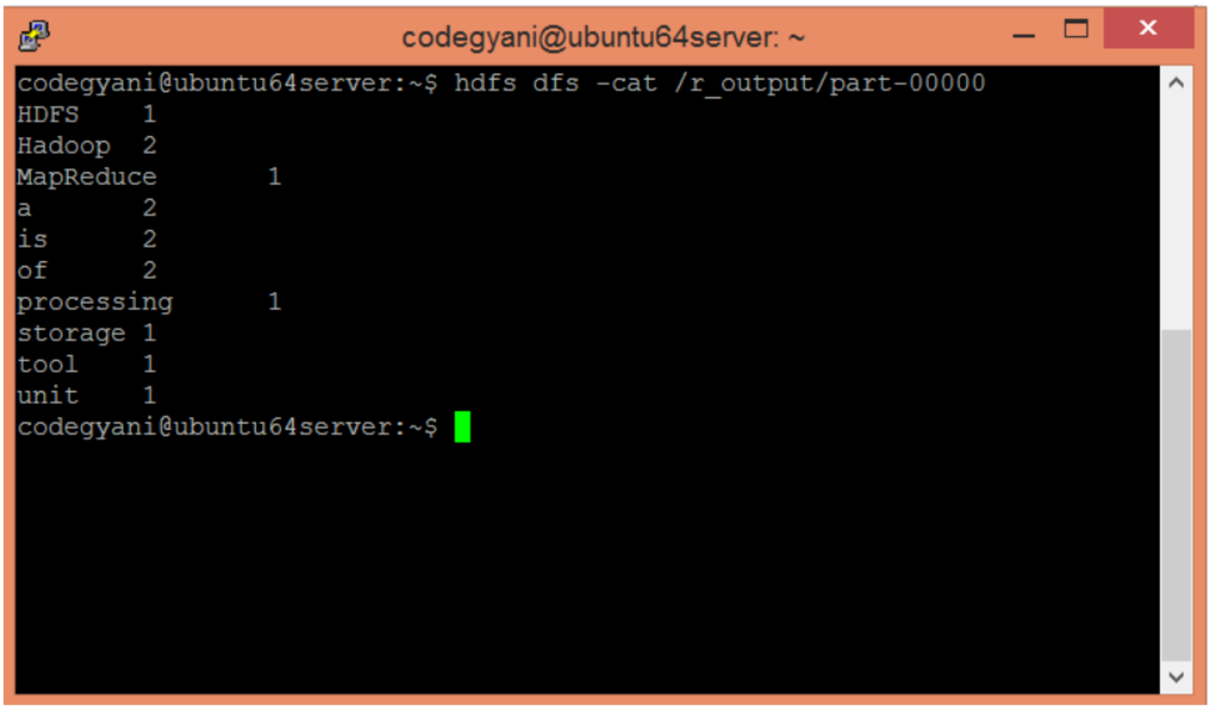
```
admin1@admin1:~$ mkdir bm
admin1@admin1:~$ cd bm
admin1@admin1:~/bm$ mkdir words
admin1@admin1:~/bm$ javac -classpath hadoop-core-1.2.1.jar words/WordCount.java
admin1@admin1:~/bm$ jar -cvf words.jar -C words/ .
added manifest
adding: WordCount$Reduce.class(in = 1627) (out= 685)(deflated 57%)
adding: WordCount.class(in = 1443) (out= 730)(deflated 49%)
adding: WordCount.java(in = 2055) (out= 657)(deflated 68%)
adding: WordCount$Map.class(in = 1855) (out= 780)(deflated 57%)
admin1@admin1:~/bm$ cd $HADOOP_HOME/sbin
admin1@admin1:~/hadoop-2.7.3/sbin$ ./start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [localhost]
admin1@localhost's password:
localhost: namenode running as process 6699. Stop it first.
admin1@localhost's password:
localhost: datanode running as process 6893. Stop it first.
Starting secondary namenodes [0.0.0.0]
admin1@0.0.0.0's password:
0.0.0.0: secondarynamenode running as process 7137. Stop it first.
starting yarn daemons
resourcemanager running as process 7318. Stop it first.
admin1@localhost's password:
localhost: nodemanager running as process 7674. Stop it first.
admin1@admin1:~/hadoop-2.7.3/sbin$ hadoop fs -mkdir /input
mkdir: `/input': File exists
admin1@admin1:~/hadoop-2.7.3/sbin$ cd ..
admin1@admin1:~/hadoop-2.7.3$ hadoop fs -put fruits.txt /input
put: `fruits.txt': No such file or directory
admin1@admin1:~/hadoop-2.7.3$ cd ..
admin1@admin1:~$ hadoop fs -put fruits.txt /input
```



```
localhost: datanode running as process 6895. Stop it first.
Starting secondary namenodes [0.0.0.0]
admin1@0.0.0.0's password:
0.0.0.0: secondarynamenode running as process 7137. Stop it first.
starting yarn daemons
resourcemanager running as process 7318. Stop it first.
admin1@localhost's password:
localhost: nodemanager running as process 7674. Stop it first.
admin1@admin1:~/hadoop-2.7.3/sbin$ hadoop fs -mkdir /input
mkdir: `/input': File exists
admin1@admin1:~/hadoop-2.7.3/sbin$ cd ..
admin1@admin1:~/hadoop-2.7.3$ hadoop fs -put fruits.txt /input
put: `fruits.txt': No such file or directory
admin1@admin1:~/hadoop-2.7.3$ cd ..
admin1@admin1:~$ hadoop fs -put fruits.txt /input
put: `/input/fruits.txt': File exists
admin1@admin1:~$ hadoop jar words.jar WordCount /input /output
22/03/29 09:11:04 INFO Configuration.deprecation: session.id is deprecated. Instead, use
dfs.metrics.session-id
22/03/29 09:11:04 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker,
sessionId=
Exception in thread "main" org.apache.hadoop.mapred.FileAlreadyExistsException: Output
directory hdfs://localhost:9000/output already exists
        at
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat.checkOutputSpecs(FileOutputFormat.ja
va:146)
        at org.apache.hadoop.mapreduce.JobSubmitter.checkSpecs(JobSubmitter.java:266)
        at org.apache.hadoop.mapreduce.JobSubmitter.submitJobInternal(JobSubmitter.java:139)
        at org.apache.hadoop.mapreduce.Job$10.run(Job.java:1290)
        at org.apache.hadoop.mapreduce.Job$10.run(Job.java:1287)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at
org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1698)
        at org.apache.hadoop.mapreduce.Job.submit(Job.java:1287)
```



```
        at org.apache.hadoop.mapreduce.Job.waitForCompletion(Job.java:1308)
        at WordCount.main(WordCount.java:59)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.hadoop.util.RunJar.run(RunJar.java:221)
        at org.apache.hadoop.util.RunJar.main(RunJar.java:136)
admin1@admin1:~$ hadoop fs -ls /output
Found 2 items
-rw-r--r--   1 admin1 supergroup          0 2019-10-31 14:28 /output/_SUCCESS
-rw-r--r--   1 admin1 supergroup         87 2019-10-31 14:28 /output/part-r-00000
admin1@admin1:~$ hadoop fs -cat /output/part-r-00000
apple   4
banana  2
grapes  3
mango   2
orange  2
pineapple       1
plum    2
pomgranate      1
raspberry       2
```



```
codegyani@ubuntu64server:~$ hdfs dfs -cat /r_output/part-00000
HDFS    1
Hadoop  2
MapReduce       1
a       2
is      2
of      2
processing      1
storage 1
tool    1
unit    1
codegyani@ubuntu64server:~$
```

CONCLUSION: Hence we have successfully completed hadoop installation and word count program .

EXPERIMENT : 12

AIM: Design a distributed application using MapReduce which processes a log file of a system.

OBJECTIVES:

THEORY:

CODE:

SalesMapper.java

```
package SalesCountry;


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 SalesMapper 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[] SingleCountryData = valueString.split("-");

            output.collect(new Text(SingleCountryData[0]), one);

      }

}
```

SalesCountryReducer.java

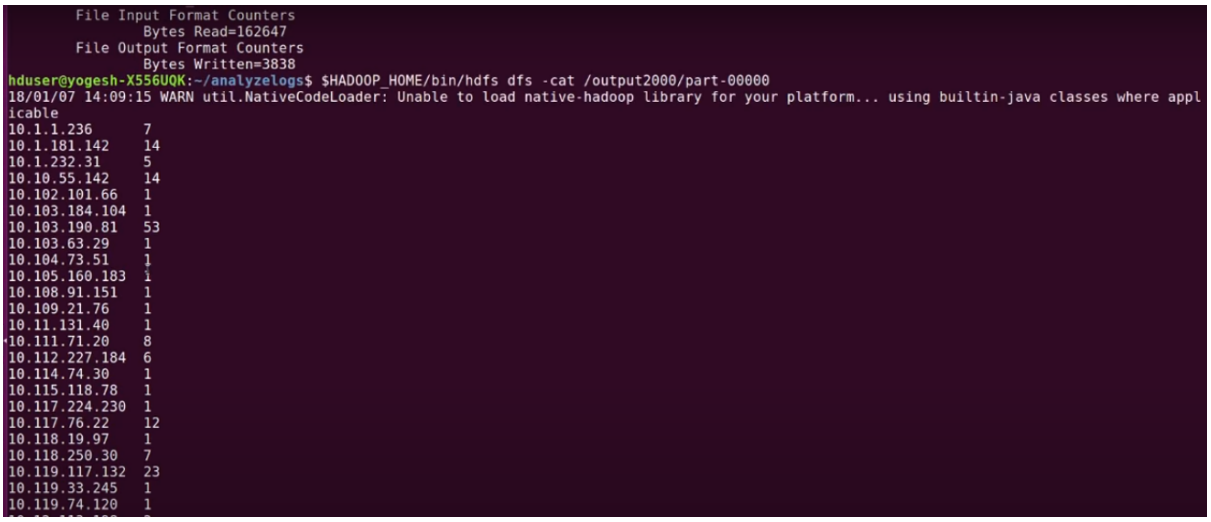
```
package SalesCountry;


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 SalesCountryReducer 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 frequencyForCountry = 0;

            while (values.hasNext()) {

                  // replace type of value with the actual type of our 
value

                  IntWritable value = (IntWritable) values.next();

                  frequencyForCountry += value.get();


            }

            output.collect(key, new IntWritable(frequencyForCountry));

      }

}
```

SalesCountryDriver.java

```
package SalesCountry;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

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

		// Set a name of the Job
		job_conf.setJobName("SalePerCountry");

		// 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(SalesCountry.SalesMapper.class);

      job_conf.setReducerClass(SalesCountry.SalesCountryReducer.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();

            }

      }

}
```

OUTPUT:



```
        File Input Format Counters
                Bytes Read=162647
        File Output Format Counters
                Bytes Written=3838
hduser@yogesh-X556UQK:~/analyzelogs$ $HADOOP_HOME/bin/hdfs dfs -cat /output2000/part-00000
18/01/07 14:09:15 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where appl
icable
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
10.112.227.184  6
10.114.74.30    1
10.115.118.78   1
10.117.224.230  1
10.117.76.22    12
10.118.19.97    1
10.118.250.30   7
10.119.117.132  23
10.119.33.245   1
10.119.74.120   1
```

CONCLUSION:

Hence ,we have successfully performed practical of distributed application using MapReduce