**EXPERIMENT 1**

**AIM:** Learn all the basic commands.

**COMMANDS:**

1. **start-all.sh –** This command is used for starting Hadoop services.
2. **jps –** This command is used to check if Hadoop services are up and running successfully.
3. **hdfs dfs -ls <path> -** This command is used to list all the files under hdfs filesystem.
4. **hdfs dfs -mkdir <folder name> -** This command is used to create a directory under hdfs filesystem.
5. **hdfs dfs -copyFromLocal <local file path> <dest (present on hdfs) > -** This command is used to copy files/folders from local file system to hdfs filesystem.
6. **hdfs dfs -cat <path> -** This command is used to print file contents onto the CLI window.
7. **hdfs dfs -copyToLocal <srcfile (on hdfs) > <local file dest> -** This command is used to copy files/folders from hdfs filesystem store to local file system.
8. **hdfs dfs -moveFromLocal <local src> <dest (on hdfs) > -** This command is used to move files from local to hdfs filesystem.
9. **hdfs dfs -cp <src (on hdfs) > <dest (on hdfs) > -** This command is used to copy files within hdfs filesystem.
10. **hdfs dfs -mv <src (on hdfs) > <src (on hdfs) > -** This command is used to move files within hdfs filesystem.
11. **hdfs dfs -rmr <filename/directoryName> -** This command is used to deletesa file from HDFS recursively.

**EXPERIMENT 2**

**AIM:** To take some census data, using census data compute the number of records for each state using Pig.

**PROCEDURE:**

1. Load data into Pig Storage format from hdfs (Hadoop Distributed File System)
2. Group data by state
3. For each grouped data generate count of population from data
4. Store the count data into output.txt in hdfs

**PROGRAM:**

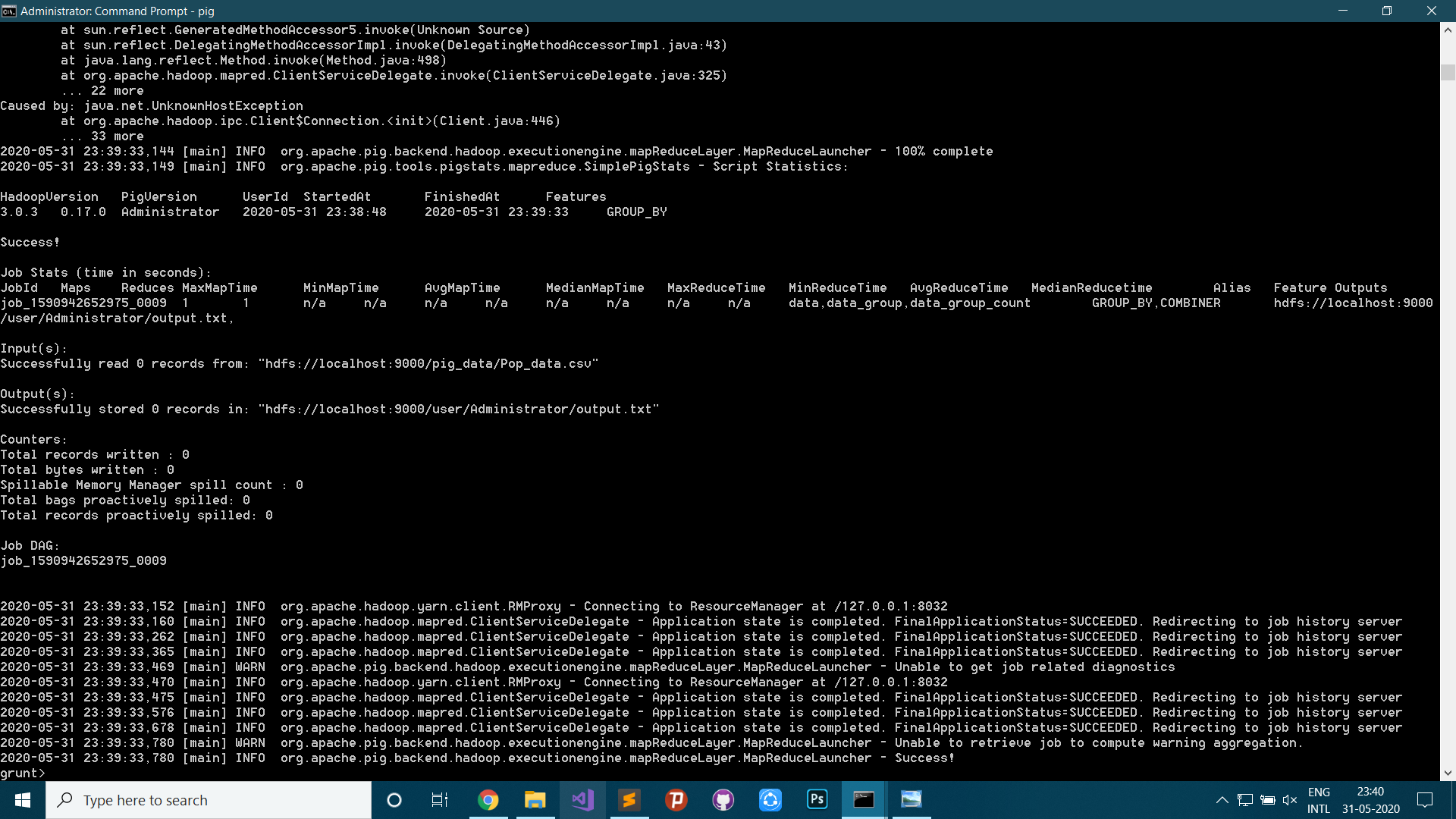
data = LOAD 'hdfs://localhost:9000/pig\_data/Pop\_data.csv' USING PigStorage(',') AS (sl:int, district:chararray, state:chararray, pop:chararray, growth:chararray, sex\_ratio:chararray, literacy:chararray);

data\_group = GROUP data BY state;

data\_group\_count = FOREACH data\_group GENERATE COUNT(data.pop);

STORE data\_group\_count INTO 'output.txt';

**OUTPUT:**



2

27

38

9

14

30

20

4

.

.

.

2

71

23

50

12

16

22

1

3

**EXPERIMENT 3**

**AIM:** Appy Natural Join between two tables using MapReduce.

**PROCEDURE:**

1. Load data in hdfs filesystem
2. Map cell number and customer name as key-value pair
3. Map cell number and delivery code as key-value pair
4. Reduce key-value pair as customer name and delivery status applying natural join
5. Store output to hdfs filesystem

**PROGRAM:**

UserFileMapper.java

package joinmapreduce;

import java.io.IOException;

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 UserFileMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, Text>

{

private String cellNumber, customerName, fileTag = "CD~";

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

{

String line = value.toString();

String splitarray[] = line.split(",");

cellNumber = splitarray[0].trim();

customerName = splitarray[1].trim();

output.collect(new Text(cellNumber), new Text(fileTag + customerName));

}

}

DeliveryFileMapper.java

package joinmapreduce;

import java.io.IOException;

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 DeliveryFileMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, Text>

{

private String cellNumber, deliveryCode, fileTag = "DR~";

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

{

String line = value.toString();

String splitarray[] = line.split(",");

cellNumber = splitarray[0].trim();

deliveryCode = splitarray[1].trim();

output.collect(new Text(cellNumber), new Text(fileTag + deliveryCode));

}

}

SmsReducer.java

package joinmapreduce;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.JobConf;

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 SmsReducer extends MapReduceBase implements Reducer<Text, Text, Text, Text> {

private String customerName, deliveryReport;

private static Map<String, String> DeliveryCodesMap = new HashMap<String, String>();

public void configure(JobConf job) {

loadDeliveryStatusCodes();

}

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

while (values.hasNext()) {

String currValue = values.next().toString();

String valueSplitted[] = currValue.split("~");

if(valueSplitted[0].equals("CD")) {

customerName = valueSplitted[1].trim();

}

else if(valueSplitted[0].equals("DR")) {

deliveryReport = DeliveryCodesMap.get(valueSplitted[1].trim());

}

}

if(customerName != null && deliveryReport != null)

output.collect(new Text(customerName), new Text(deliveryReport));

else if(customerName == null)

output.collect(new Text("customerName"), new Text(deliveryReport));

else if(deliveryReport == null)

output.collect(new Text(customerName), new Text("deliveryReport"));

}

private void loadDeliveryStatusCodes() {

String strRead;

try {

BufferedReader reader = new BufferedReader(new FileReader("C:\\Users\\Administrator\\eclipse-workspace\\JoinMapReduce\\DeliveryStatusCodes.txt"));

while ((strRead = reader.readLine() ) != null) {

String splitarray[] = strRead.split(",");

DeliveryCodesMap.put(splitarray[0].trim(), splitarray[1].trim());

}

reader.close();

}

catch (FileNotFoundException e) {

e.printStackTrace();

}

catch( IOException e ) {

e.printStackTrace();

}

}

}

SmsDriver.java

package joinmapreduce;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

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.lib.MultipleInputs;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class SmsDriver extends Configured implements Tool {

public int run(String[] args) throws Exception {

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

conf.setJobName("SMS Reports");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(Text.class);

conf.setReducerClass(SmsReducer.class);

MultipleInputs.addInputPath(conf, new Path(args[0]), TextInputFormat.class, UserFileMapper.class);

MultipleInputs.addInputPath(conf, new Path(args[1]), TextInputFormat.class, DeliveryFileMapper.class);

deleteDirectory(args[2], conf);

FileOutputFormat.setOutputPath(conf, new Path(args[2]));

JobClient.runJob(conf);

return 0;

}

private static void deleteDirectory(String args, Configuration conf) throws IOException {

Path p = new Path(args);

FileSystem fs = FileSystem.get(conf);

fs.exists(p);

fs.delete(p, true);

}

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

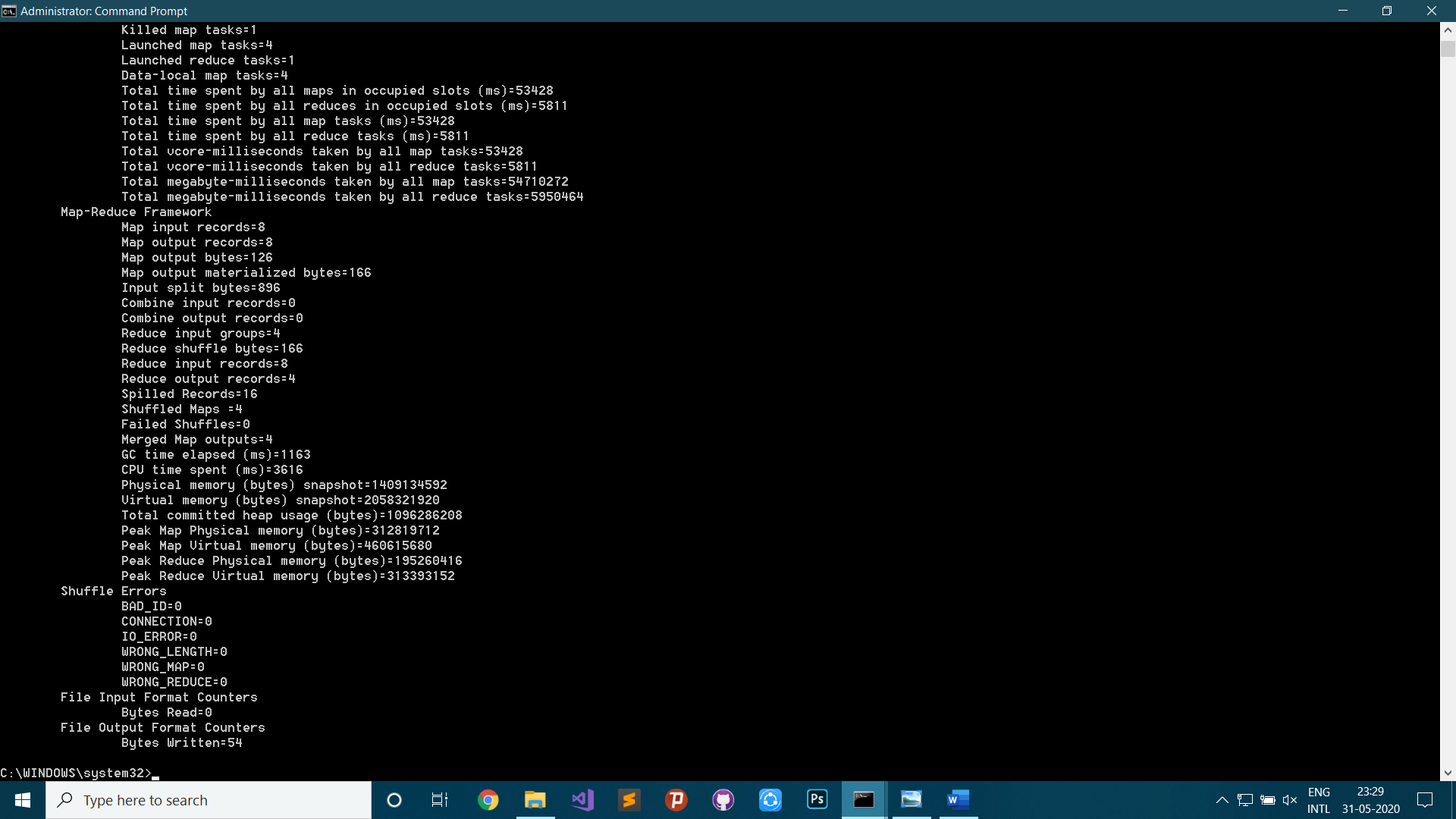
int res = ToolRunner.run(new Configuration(), new SmsDriver(), args);

System.exit(res);

}

}

**OUTPUT:**



**part-00000**

Jim Delivered

Tom Pending

Harry Failed

Richard Resend

**EXPERIMENT 4**

**AIM:** Integrate two text files and count the number of words available in the resultant file and also find the length of the word.

**PROCEDURE:**

1. Load data in hdfs filesystem
2. Start Job 1
3. Map “Total” and “1” as key-value pair
4. Reduce “Total” and sum of “1” s as key-value pair
5. Store output to hdfs filesystem
6. Start Job 2
7. Map token(word) and “1” as key-value pair
8. Reduce token(word) and length of tokens as key-value pair
9. Store output to hdfs filesystem

**PROGRAM:**

WCJMapper.java

package wordcountjob;

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.\*;

public class WCJMapper 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 {

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

while (itr.hasMoreTokens()) {

output.collect(new Text("Total"), one);

itr.nextToken();

}

}

}

WCJMapper1.java

package wordcountjob;

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.\*;

public class WCJMapper1 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 {

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

while (itr.hasMoreTokens()) {

output.collect(new Text(itr.nextToken()), one);

}

}

}

WCJReducer.java

package wordcountjob;

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 WCJReducer 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 sum = 0;

while (values.hasNext()) {

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

sum += value.get();

}

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

}

}

WCJReducer1.java

package wordcountjob;

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 WCJReducer1 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;

output.collect(key, new IntWritable(t\_key.getLength()));

}

}

WCJDriver.java

package wordcountjob;

import java.io.IOException;

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

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

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

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

import org.apache.hadoop.mapreduce.Job;

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

public class WCJDriver extends Configured implements Tool {

public int run(String[] args) {

JobConf job\_conf = new JobConf(WCJDriver.class);

job\_conf.setJobName("Total Length");

job\_conf.setOutputKeyClass(Text.class);

job\_conf.setOutputValueClass(IntWritable.class);

job\_conf.setMapperClass(wordcountjob.WCJMapper.class);

job\_conf.setReducerClass(wordcountjob.WCJReducer.class);

job\_conf.setInputFormat(TextInputFormat.class);

job\_conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(job\_conf, new Path(args[0]));

FileOutputFormat.setOutputPath(job\_conf, new Path(args[1]));

JobConf job\_conf1 = new JobConf(WCJDriver.class);

job\_conf1.setJobName("Word Count");

job\_conf1.setOutputKeyClass(Text.class);

job\_conf1.setOutputValueClass(IntWritable.class);

job\_conf1.setMapperClass(wordcountjob.WCJMapper1.class);

job\_conf1.setReducerClass(wordcountjob.WCJReducer1.class);

job\_conf1.setInputFormat(TextInputFormat.class);

job\_conf1.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(job\_conf1, new Path(args[0]));

FileOutputFormat.setOutputPath(job\_conf1, new Path(args[2]));

int result = -1;

try {

deleteDirectory(args[1], job\_conf);

Job job1 = Job.getInstance(job\_conf);

result = job1.waitForCompletion(true) ? 0 : 1;

} catch (Exception e) {

e.printStackTrace();

}

System.out.println("\nFirst Job Finished=============\n");

if(result == 0) {

try {

deleteDirectory(args[2], job\_conf1);

Job job2 = Job.getInstance(job\_conf1);

result = job2.waitForCompletion(true) ? 0 : 1;

} catch (Exception e) {

e.printStackTrace();

}

System.out.println("\nSecond Job Finished=============\n");

}

return result;

}

private static void deleteDirectory(String args, Configuration conf) throws IOException {

Path p = new Path(args);

FileSystem fs = FileSystem.get(conf);

fs.exists(p);

fs.delete(p, true);

}

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

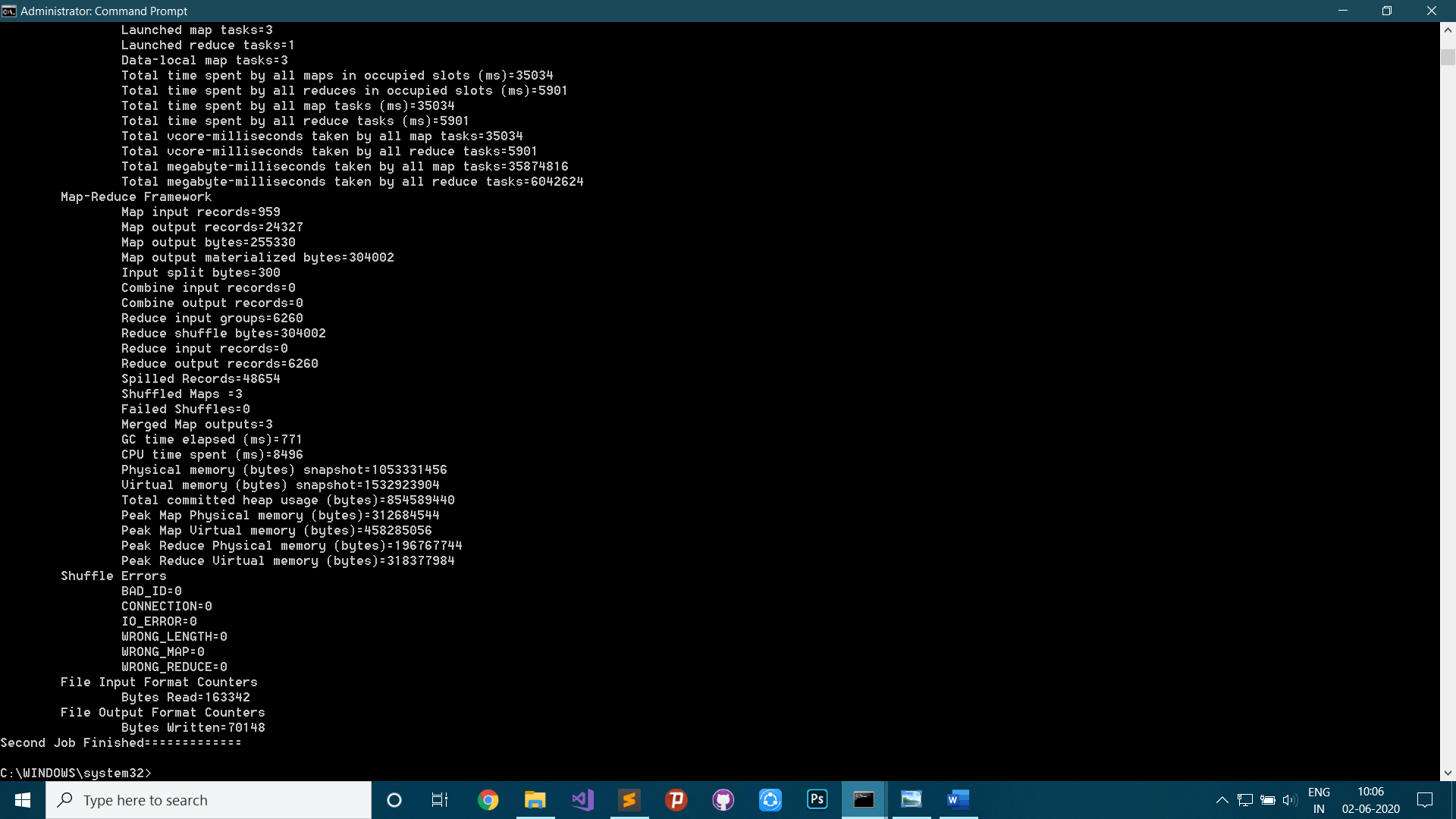
int res = ToolRunner.run(new Configuration(), new WCJDriver(), args);

System.exit(res);

}

}

**OUTPUT:**



**output\_dir\_1\_WCL/part-00000**

Total 24327

**output\_dir\_2\_WCL/part-00000**

"30% 4

"A 2

"Acorn 6

"Carrie 7

"DST 4

"Do 3

"Enter 6

"Error 6

"Fingerprint 12

"Five 5

"Fujianese" 11

.

.

.

youngsters,[493][52] 20

youth 5

youths 6

youths" 7

zone.[619][620] 15

~9000 5

┬º 2

ΓÇîα░╛) 7

ΓÇô 3

ΓÇó 3

σÅìΘÇüΣ╕¡ΘüèΦíî 15

σÆîτÉåΘ¥₧) 10

σ╗óΘ¥Æ; 7

**EXPERIMENT 5**

**AIM:** Write a MapReduce Job that read any text input and find the average length of all words that starts with each character.

**PROCEDURE:**

1. Load data in hdfs filesystem
2. Map first character of each token (word) and the length of that token
3. Reduce the character and average length of the words beginning with that character
4. Store output to hdfs filesystem

**PROGRAM:**

WordCountAverage.java

package org.apache.hadoop.examples;

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.DoubleWritable;

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 WordCountAverage {

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

private Text firstCharacter = new Text();

private final static IntWritable length = new IntWritable(1);

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

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

while (itr.hasMoreTokens()) {

String token = itr.nextToken();

firstCharacter.set(token.substring(0, 1));

length.set(token.length());

context.write(firstCharacter, length);

}

}

}

public static class WCAReducer extends Reducer<Text, IntWritable, Text, DoubleWritable> {

private DoubleWritable result = new DoubleWritable();

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

double sum = 0;

int total = 0;

for (IntWritable val : values) {

total ++;

sum += val.get();

}

double average = sum / total;

result.set(average);

context.write(key, result);

}

}

private static void deleteDirectory(String args, Configuration conf) throws IOException {

Path p = new Path(args);

FileSystem fs = FileSystem.get(conf);

fs.exists(p);

fs.delete(p, true);

}

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

Configuration conf = new Configuration();

Job job = new Job(conf, "Word Count Average");

job.setJarByClass(WordCountAverage.class);

job.setMapperClass(WCAMapper.class);

job.setReducerClass(WCAReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[1]));

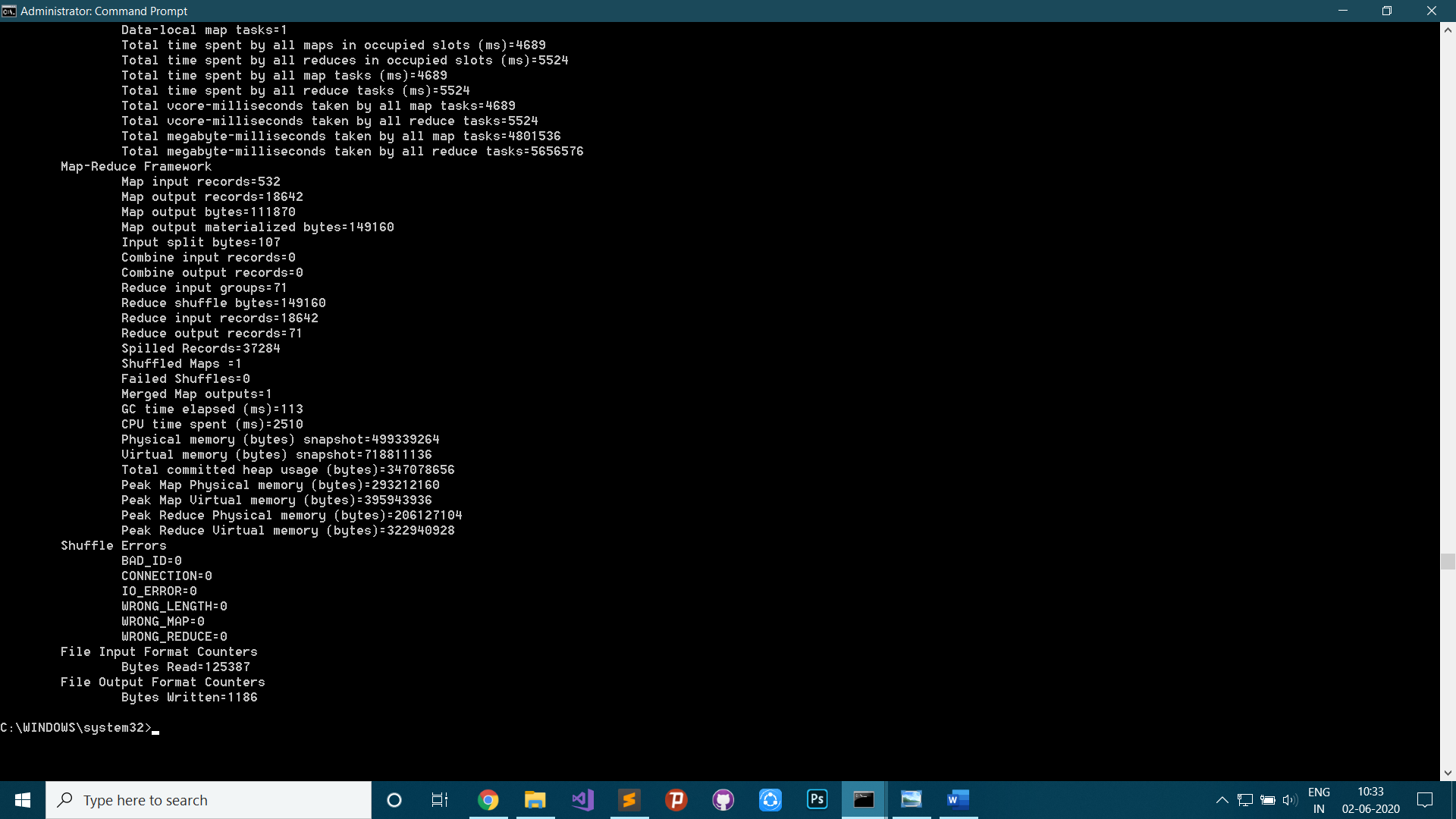
deleteDirectory(args[1], conf);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**OUTPUT:**



**Part-r-00000**

" 7.810457516339869

# 7.0

( 7.1891891891891895

- 1.0

0 5.6

1 2.910179640718563

2 4.852739726027397

3 3.0

4 1.6666666666666667

.

.

.

v 7.762962962962963

w 4.491048593350383

y 5.636363636363637

z 15.0

ΓÇó 1.0

σÅì 5.0

σÆî 4.0

σ╗ó 3.0

**EXPERIMENT 6**

**AIM:** Analyse the log file from a web server to count the no. of hits made from each unique IP address.

**PROCEDURE:**

1. Load the sample data in hdfs filesystem
2. Map ip address as key and one (1) as value
3. Reduce the IP address as key and the sum of similar IP addresses as value
4. Store output to hdfs filesystem

**PROGRAM:**

IPHits.java

package iphits;

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import iphits.IPHits;

import org.apache.hadoop.fs.FileSystem;

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 IPHits {

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

private Text ip = new Text();

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

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

StringTokenizer itr = new StringTokenizer(value.toString(), "\n");

//String token = itr.nextToken();

while (itr.hasMoreTokens()) {

String token = itr.nextToken();

if(token.substring(0, token.indexOf(',')).equalsIgnoreCase("ip") == true)

continue;

ip.set(token.substring(0, token.indexOf(',')));

context.write(ip, one);

}

}

}

public static class IPHitsReducer 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);

}

}

private static void deleteDirectory(String args, Configuration conf) throws IOException {

Path p = new Path(args);

FileSystem fs = FileSystem.get(conf);

fs.exists(p);

fs.delete(p, true);

}

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

Configuration conf = new Configuration();

Job job = new Job(conf, "Word Count Average");

job.setJarByClass(IPHits.class);

job.setMapperClass(IPHitsMapper.class);

job.setReducerClass(IPHitsReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[1]));

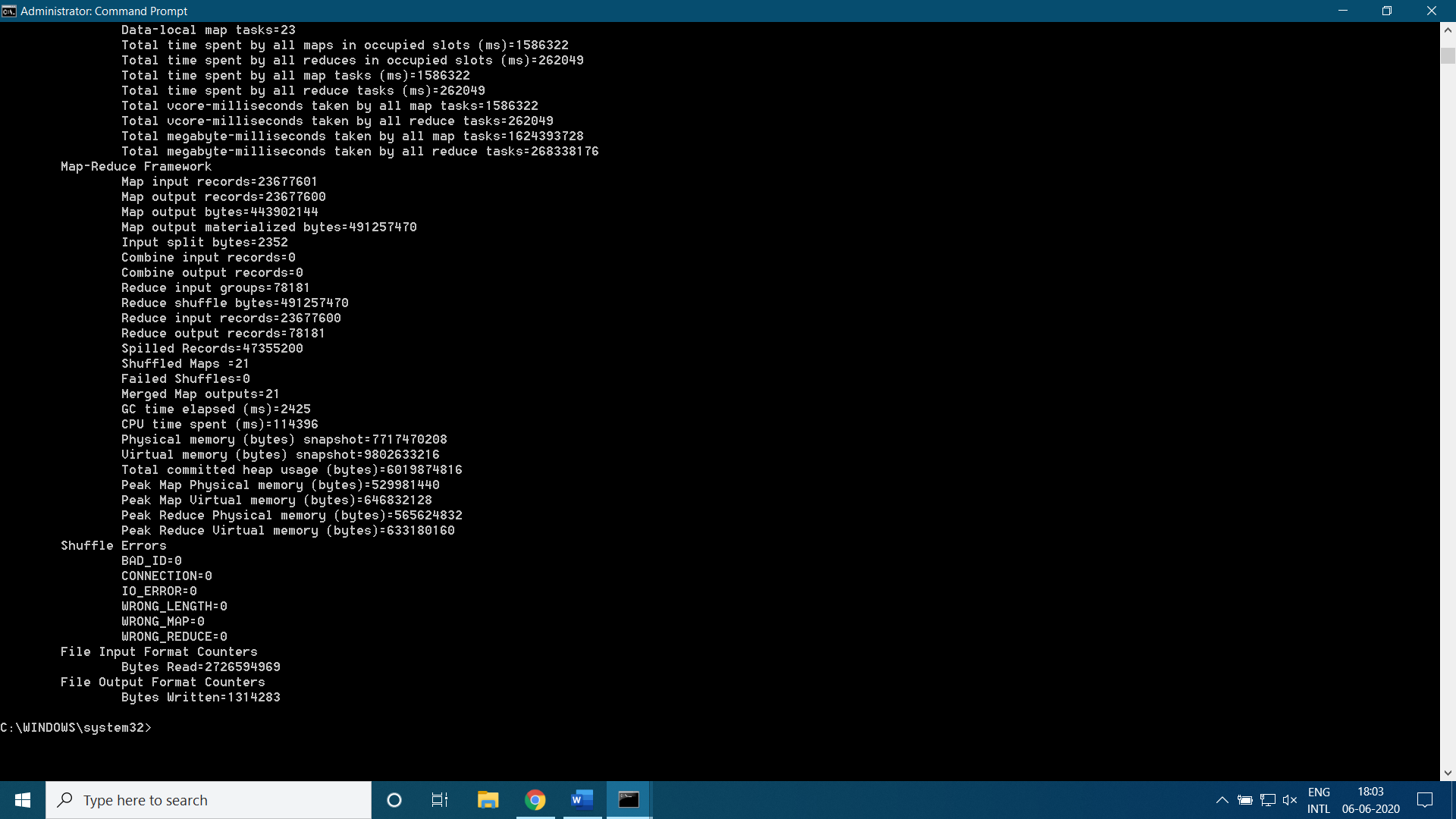
deleteDirectory(args[1], conf);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**OUTPUT:**



**Part-r-00000**

1.0.139.ceb 1

1.10.175.bch 1

1.10.205.fjb 1

1.119.129.eag 1

1.120.134.aca 1

1.124.48.acc 1

1.124.48.icd 2

1.127.48.fbb 8

.

.

.

99.96.207.bdf 1

99.97.29.ihf 2

99.98.103.cca 3

99.99.223.jda 4

99.99.56.def 2

**EXPERIMENT 7**

**AIM:** Write a MapReduce program that counts the no. of times words appear next to each other.

**PROCEDURE:**

1. Load the sample data in hdfs filesystem
2. Map a pair of words separated by a single space as key and one (1) as value
3. Reduce the pair of words as key and the total similar pairs as value
4. Store output to hdfs filesystem

**PROGRAM:**

IPHits.java

package wordpair;

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

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 WordPair {

public static class WPMapper 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(), " ");

if(itr.hasMoreTokens()) {

String str1 = itr.nextToken(), str2 = "";

while (itr.hasMoreTokens()) {

str2 = itr.nextToken();

word.set(str1.trim() + " " + str2.trim());

context.write(word, one);

str1 = str2;

}

}

}

}

public static class WPReducer 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);

}

}

private static void deleteDirectory(String args, Configuration conf) throws IOException {

Path p = new Path(args);

FileSystem fs = FileSystem.get(conf);

fs.exists(p);

fs.delete(p, true);

}

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Total words next to each other");

job.setJarByClass(WordPair.class);

job.setMapperClass(WPMapper.class);

job.setCombinerClass(WPReducer.class);

job.setReducerClass(WPReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

deleteDirectory(args[1], conf);

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

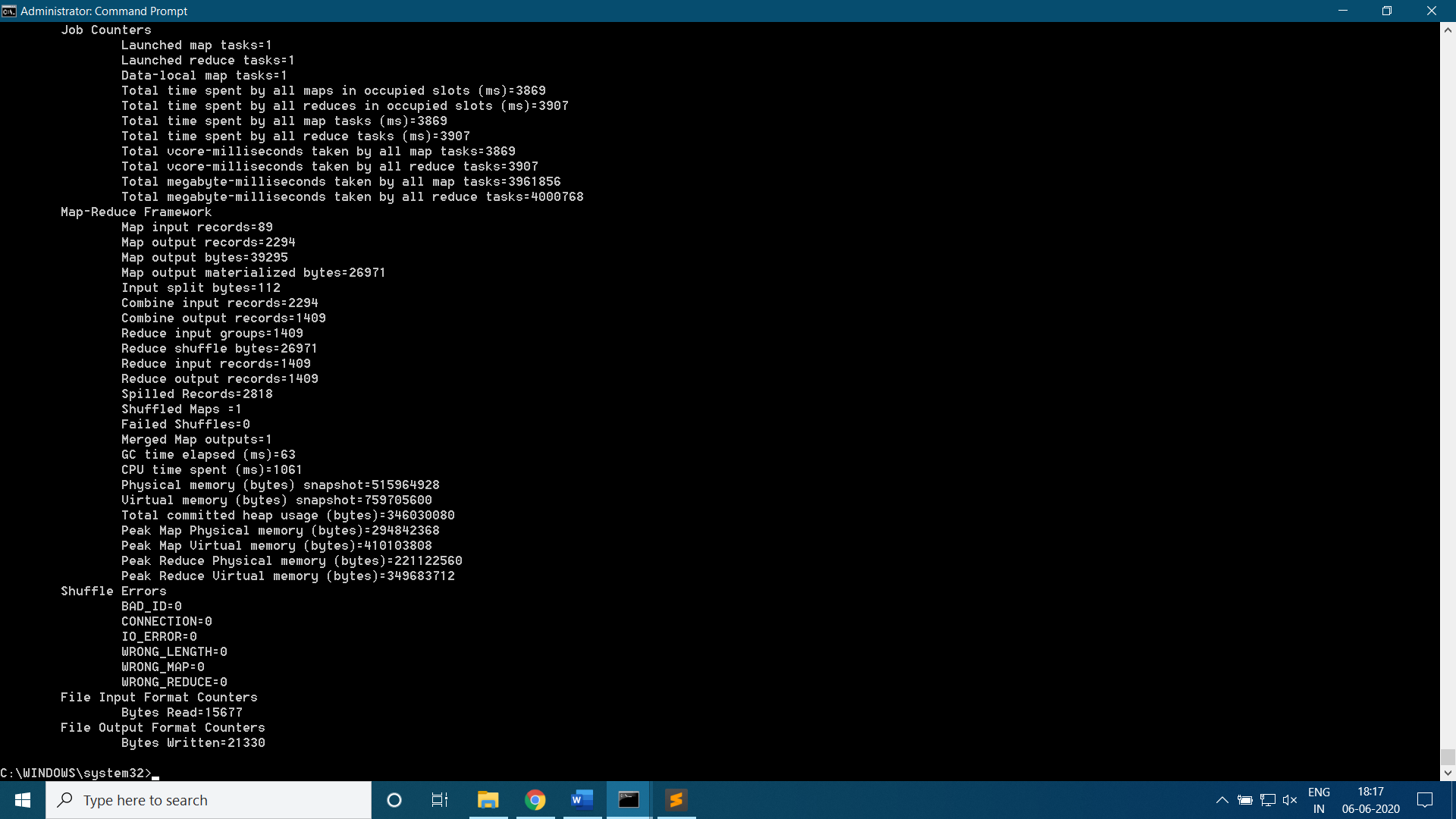
FileOutputFormat.setOutputPath(job, new Path(args[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**OUTPUT:**



**Part-r-00000**

(Try it 1

1 In eleifend 1

1, 2020 1

2 Cras fringilla 1

2006 review 1

2011, researchers 1

3 Aliquam erat 1

4 Fusce vitae 1

5 Etiam vehicula 1

.

.

.

ΓÇó Maecenas tincidunt 1

ΓÇó Mauris id 1

ΓÇó Morbi viverra 1

ΓÇó Nulla facilisi. 1