Name of the student:	Tanmay Prashant Rane	**Roll No.**	8031
Practical Number:2		**Date of Practical:**	
Relevant CO's: ITC802.2	At the end of the course students will be able to use tools like hadoop and NoSQL to solve big data related problems.		
Sign here to indicate that you have read all the relevant material provided before attempting this practical		**Sign:**	

Practical grading using Rubrics

Indicator	**Very Poor**	**Poor**	**Average**	**Good**	**Excellent**
Timeline (2)	More than a session late (0)	NA	NA	NA	Early or on time (2)
Code design (2)	N/A	Very poor code design with no comments and indentation(0.5)	Poor code design with very comments and indentation (1)	Design with good coding standards (1.5)	Accurate design with better coding satndards (2)
Performance (4)	Unable to perform the experiment (0)	Able to partially perform the experiment (1)	Able to perform the experiment for certain use cases (2)	Able to perform the experiment considering mostof the use cases (3)	Able to perform the experiment considering all use cases (4)
Postlab (2)	No Execution(0)	N/A	Partially Executed (1)	N/A	Fully Executed (2)

Total Marks (10)	**Sign of instructor**

Practical

Course title: Big Data Analytics
Course term: 2019-2020
Instructor name: Saurabh Kulkarni

Problem Statement: Counting number of words in given text file using map reduce.

Theory:Explain the working of word count using map reduce with small example and diagrams

Hadoop WordCount operation occurs in 3 stages –

Mapper Phase

Shuffle Phase

Reducer Phase

Hadoop WordCount Example- Mapper Phase Execution

The text from the input text file is tokenized into words to form a key value pair with all the words present in the input text file. The key is the word from the input file and value is '1'.

For instance if you consider the sentence "An elephant is an animal". The mapper phase in the WordCount example will split the string into individual tokens i.e. words. In this case, the entire sentence will be split into 5 tokens (one for each word) with a value 1 as shown below –

Key-Value pairs from Hadoop Map Phase Execution-
(an,1)
(elephant,1)
(is,1)
(an,1)
(animal,1)

Hadoop WordCount Example- Shuffle Phase Execution

After the map phase execution is completed successfully, shuffle phase is executed automatically wherein the key-value pairs generated in the map phase are taken as input and then sorted in alphabetical order. After the shuffle phase is executed from the WordCount example code, the output will look like this -
(an,1)
(an,1)
(animal,1)
(elephant,1)
(is,1)

Hadoop WordCount Example- Reducer Phase Execution

In the reduce phase, all the keys are grouped together and the values for similar keys are added up to find the occurrences for a particular word. It is like an aggregation phase for the keys generated by the map phase. The reducer phase takes the output of shuffle phase as input and then reduces the key-value pairs to unique keys with values added up. In our example “An elephant is an animal.”

(an,2)
(animal,1)
(elephant,1)
(is,1)

This is how the MapReduce word count program executes and outputs the number of occurrences of a word in any given input file. An important point to note during the execution of the WordCount example is that the mapper class in the WordCount program will execute completely on the entire input file and not just a single sentence. Suppose if the input file has 15 lines then the mapper class will split the words of all the 15 lines and form initial key value pairs for the entire dataset. The reducer execution will begin only after the mapper phase is executed successfully.

Code:

code for mapper:

```
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.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Mapper.Context;

public class WCMapper extends Mapper<LongWritable, Text, Text,
    ,→ IntWritable>
 {
        // Create object of type Text to hold string created per word
           ,→ of given document
```

Code for Reducer:

Code for Driver Class:

```
//MAPPER
private Text word = new Text();

        // Create final static variable of type IntWritable with value equal to 1 as according to algorithm map
function puts 1 for each word encountered.
         private final static IntWritable one = new IntWritable(1);

        //write a map function here
        public void map(LongWritable ikey, Text ivalue, Context context)
                        throws IOException, InterruptedException {
                //Define a String type variable and assign value which is equal to string equivalent of Text value
passed to map function
                String line = ivalue.toString();
                //Convert this variable to tokens using StringTokenizer class as it separates out each word of the
document
                StringTokenizer tokenizer = new StringTokenizer(line);
                //Until there are tokens in StringTokenizer,
                while(tokenizer.hasMoreTokens())
                {
                // set the Text object created in first line of the code in WCMapper to next token in the
StringTokenizer
                        word.set(tokenizer.nextToken());
                 // Write this Text Variable and final static IntWritable Variable created to the context so that
key-value pairs are generated.
                        context.write(word, one);
                }
                }
        }
```

```
//REDUCER
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Reducer.Context;


public class WCReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

	public void reduce(Text key, Iterable<IntWritable> values, Context context)
			throws IOException, InterruptedException {
		// process values
		//initialize sum=0
		int sum = 0;

		//Iterate over the collection of values to get count of each word i.e. key
		 for(IntWritable val:values)
		 {
			sum+=val.get();
		 }
		 //Write this count to the context.
		context.write(key,new IntWritable(sum));

	}
}
//DRIVER
import java.io.IOException;
import java.util.Date;
import java.util.Formatter;
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.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.mapreduce.*;

public class WCDriver {

	public static void main(String[] args) throws Exception {
		Configuration conf = new Configuration();
		 GenericOptionsParser parser = new GenericOptionsParser(conf, args);
		 args = parser.getRemainingArgs();

		//Job job = new Job(conf, "wordcount");
		 Job job=new Job(conf,"wordcount");
		 job.setJarByClass(WCDriver.class);

		//job.setOutputKeyClass(Text.class);
		// job.setOutputValueClass(IntWritable.class);
```

```
job.setInputFormatClass(TextInputFormat.class);
                        job.setOutputFormatClass(TextOutputFormat.class);

                       Formatter formatter = new Formatter();
                        String outpath = "Out"
                        + formatter.format("%1$tm%1$td%1$tH%1$tM%1$tS", new Date());
                        FileInputFormat.setInputPaths(job, new
Path("/media/tanmay/Data/SEM-8/BDA/EXP2/testfiles"));
                        FileOutputFormat.setOutputPath(job, new
Path("/media/tanmay/Data/SEM-8/BDA/EXP2/output"));
                        job.setMapperClass(WCMapper.class);
                        job.setReducerClass(WCReducer.class);

                       System.out.println(job.waitForCompletion(true));
          }

}
```

PostLab:Find inverted index

In this assignment you have to implement a simple map reduce job that builds an inverted index on the set of input documents. An inverted index maps each word to a list of documents that contain the word, and additionally records the position of each occurrence of the word within the document.

For the purpose of this assignment, the position will be based on counting words, not characters.

Ex: Assume below are the input Documents.
file1="data is good."
file2="data is not good?"

Output:
data (file1,1)(file2,1)
good (file1,3)(file2,4)
is (file1,2)(file2,2)
not (file2,3)

For more details on inverted indices, you can check out the Wikipedia page on inverted indices.

Now in this assignment you need to implement above map-reduce job.
Input: A set of documents

Output:
Map: word1 (filename, position)
word2 (filename, position)
word1 (filename, position)
and so on for each occurrence of each word.
Reduce: word1 (filename, position)(filename,position)
word2 (filename, position)
and so on for each word.
Code for getting file name in Hadoop, which can be used in the Map function:

```
String filename = null;
filename = ((FileSplit)context.getInputSplit()).
,→getPath().getName();
```

Code for postlab question

```
//MAPPER
public class Map extends Mapper<LongWritable,Text,Text,Text> {
@Override
public void map(LongWritable key, Text value, Context context)
throws IOException,InterruptedException
{
/*Get the name of the file using context.getInputSplit()method*/
String fileName = ((FileSplit) context.getInputSplit()).getPath().getName();
String line=value.toString();
//Split the line in words
String words[]=line.split(" ");
for(String s:words){
//for each word emit word as key and file name as value
context.write(new Text(s), new Text(fileName));
}
}
}
```

//REDUCER

```
public static class Reduce extends
Reducer<Text, Text, Text, Text> {
@Override
public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
/*Declare the Hash Map to store File name as key to compute and store number of times the filename is occurred
for as value*/
HashMap m=new HashMap();
int count=0;
for(Text t:values){
String str=t.toString();
/*Check if file name is present in the HashMap ,if File name is not present then add the Filename to the HashMap
and increment the counter by one , This condition will be satisfied on first occurrence of that word*/
if(m!=null &&m.get(str)!=null){
count=(int)m.get(str);
m.put(str, ++count);
}else{
/*Else part will execute if file name is already added then just increase the count for that file name which is stored
as key in the hash map*/
m.put(str, 1);
}
}
/* Emit word and [file1→count of the word1 in file1 , file2→count of the word1 in file2 ……] as output*/
context.write(key, new Text(m.toString()));
}
}
```

//DRIVER

```
public static void main(String[] args) throws Exception {
Configuration conf= new Configuration();
Job job = new Job(conf,"UseCase1");
//Defining the output key and value class for the mapper
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(Text.class);
job.setJarByClass(InvertedIndex.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
//Defining the output value class for the mapper
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, outputPath);
//deleting the output path automatically from hdfs so that we don't have delete it explicitly
outputPath.getFileSystem(conf).delete(outputPath);
//exiting the job only if the flag value becomes false
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```