Name of the student:		**Roll No.**	
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 most of 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: 2021-2022
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

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 strings created per word
           ↪ of given document
```

Code for Reducer:

Code for Driver Class:

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

8695 | YASH SANKPAL | BATCH-C

THEORY:

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

MapReduce consists of 2 steps:

- **Map Function – It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (Key-Value pair).**
- **Example – (Map function in Word Count)**

Input	Set of data	Bus, Car, bus, car, train, car, bus, car, train, bus, TRAIN,BUS, buS, caR, CAR, car, BUS, TRAIN
Output	Convert into another set of data (Key,Value)	(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1), (car,1), (BUS,1), (TRAIN,1)

- **Reduce Function – Takes the output from Map as an input and combines those data tuples into a smaller set of tuples.**
- **Example – (Reduce function in Word Count)**

Input (output of Map function)	Set of Tuples	(Bus,1), (Car,1), (bus,1), (car,1), (train,1), (car,1), (bus,1), (car,1), (train,1), (bus,1), (TRAIN,1),(BUS,1), (buS,1), (caR,1), (CAR,1),

		(car,1), (BUS,1), (TRAIN,1)
Output	Converts into smaller set of tuples	(BUS,7), (CAR,7), (TRAIN,4)

Work Flow of the Program

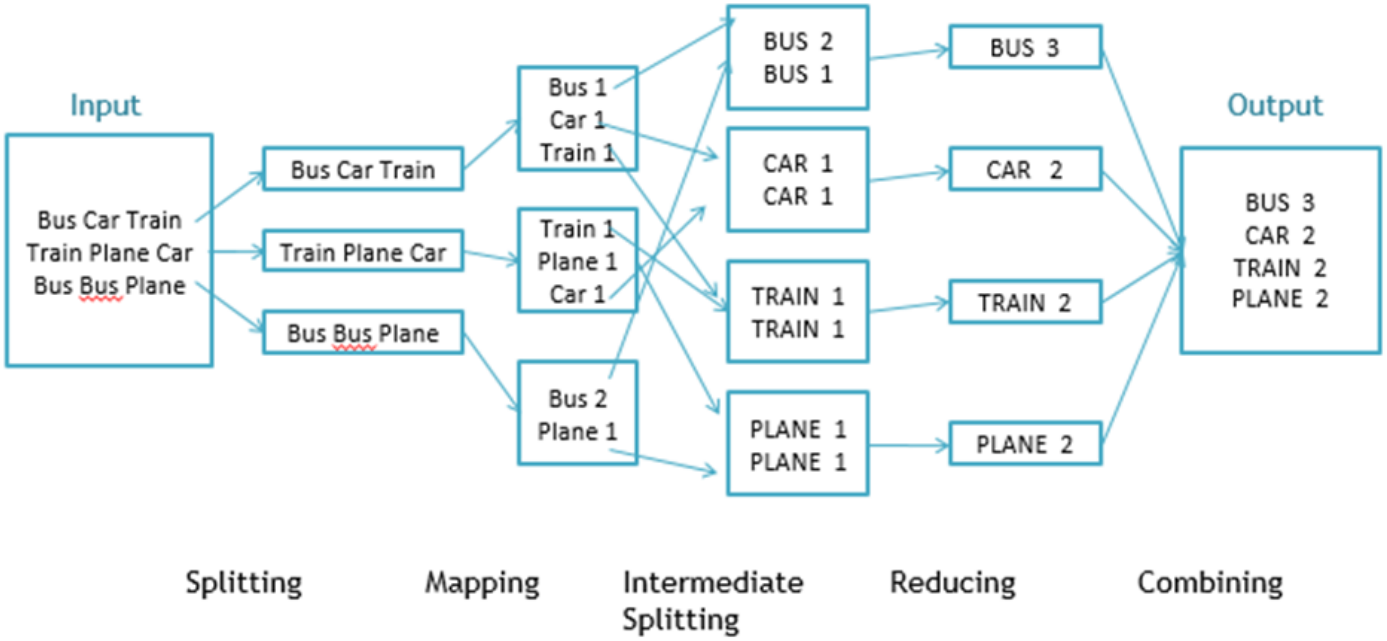


Fig. WorkFlow of MapReducing

Workflow of MapReduce consists of 5 steps:

1. **Splitting – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line ('\n').**
2. **Mapping – as explained above.**
3. **Intermediate splitting – the entire process in parallel on different clusters. In order to group them in "Reduce Phase" the similar KEY data should be on the same cluster.**
4. **Reduce – it is nothing but mostly group by phase.**
5. **Combining – The last phase where all the data (individual result set from each cluster) is combined together to form a result.**

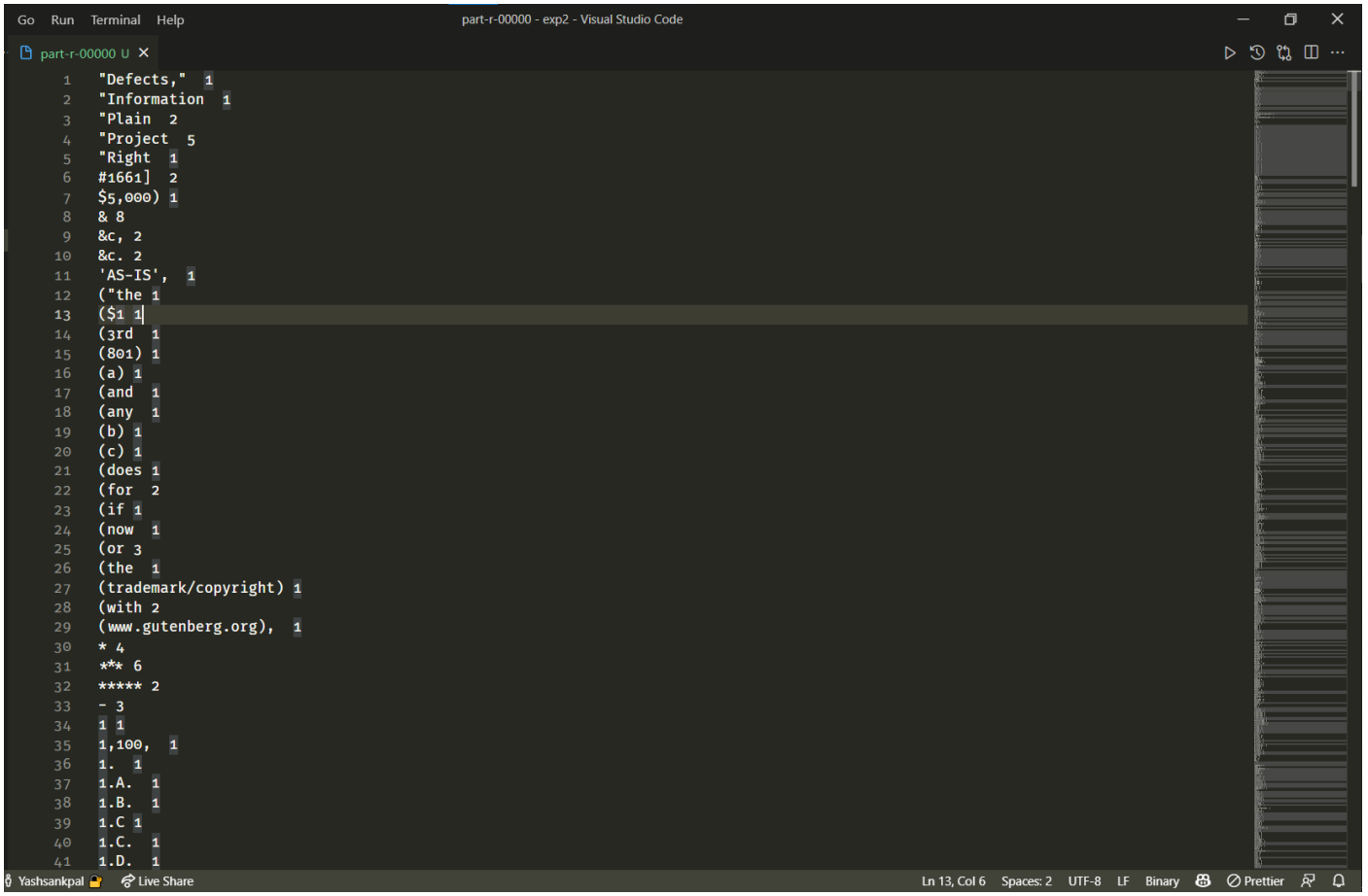
Code:

```java
package exp2;

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 strings created per word of given document
  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.
  public 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 var = ivalue.toString();
    //Convert this variable to tokens using StringTokenizer class as it separates out each word of the document
    StringTokenizer st = new StringTokenizer(var);
    //Until there are tokens in StringTokenizer,
    while(st.hasMoreTokens()) {
      // set the Text object created in first line of the code in WCMapper to next token in the StringTokenizer
      word.set(st.nextToken());
      context.write(word,one);

    }
    // Write this Text Variable and final static IntWritable Variable created to the context so that key-value pairs are generated.

    }
  }
```

```java
package exp2;

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 i:values) {
       sum += i.get();
     }
    //Write this count to the context.
     context.write(key,new IntWritable(sum));
  }
  }
```

```java
package exp2;



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;
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.setMapperClass(WCMapper.class);
     job.setReducerClass(WCReducer.class);

     job.setMapOutputKeyClass(Text.class);
     job.setMapOutputValueClass(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("hdfs://localhost:9000/input/wordcountdata.txt"));
     FileOutputFormat.setOutputPath(job, new Path("hdfs://localhost:9000/output/exp2"));

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

}

```

OUTPUT:



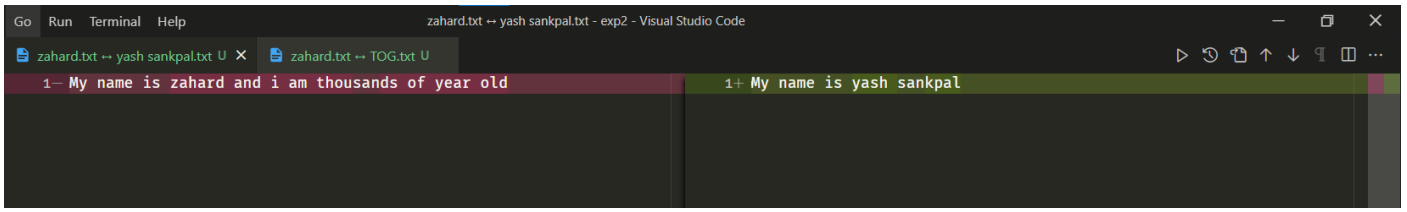
POSTLAB:

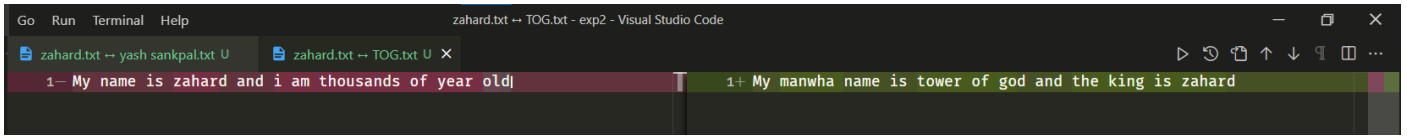
Inverted Index

```java
package exp2_postlab;

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

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

public class WCMapper extends Mapper<LongWritable, Text, Text, Text> {
  private Text nameKey = new Text();
  private Text fileNameValue = new Text();

  public void map(LongWritable ikey, Text ivalue, Context context) throws IOException, InterruptedException {
    String str = ivalue.toString();
    String[] st = str.split(" ");
    String filename = null;

    for(int i=0;i<st.length;i++) {
      filename = ((FileSplit) context.getInputSplit()).getPath().getName();
      nameKey.set(st[i]);
      fileNameValue.set("( "+filename+","+i+" )");
      context.write(nameKey,fileNameValue);
    }
  }

}
```

```java
package exp2_postlab;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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 WCDriver {

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "JobName");
    job.setJarByClass(WCDriver.class);
    // TODO: specify a mapper
    job.setMapperClass(WCMapper.class);
    // TODO: specify a reducer
    job.setReducerClass(WCReducer.class);

    // TODO: specify output types
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    // TODO: specify input and output DIRECTORIES (not files)
    FileInputFormat.setInputPaths(job, new Path("hdfs://localhost:9000/input/exp2-postlab"));
    FileOutputFormat.setOutputPath(job, new Path("hdfs://localhost:9000/exp2-postlab"));

    if (!job.waitForCompletion(true))
      return;
    System.out.println(job.waitForCompletion(true));
  }

}
```

```java
package exp2_postlab;

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, Text, Text, Text> {

  public void reduce(Text key, Iterable<Text> values, Context context)
      throws IOException, InterruptedException {
    // process values
    //initialize sum=0
    //Iterate over the collection of values to get count of each word i.e. key
    StringBuilder  sb = new StringBuilder();

    for(Text i:values) {
      sb.append(i+" ");
    }


    //Write this count to the context.
     context.write(key,new Text(sb.toString()));
  }
  }

```





OUTPUT:

```
My  ( yash sankpal.txt,0 ) ( TOG.txt,0 ) ( zahard.txt,0 )
am  ( zahard.txt,6 )
and ( zahard.txt,4 ) ( TOG.txt,7 )
god ( TOG.txt,6 )
i ( zahard.txt,5 )
is  ( zahard.txt,2 ) ( yash sankpal.txt,2 ) ( TOG.txt,10 ) ( TOG.txt,3 )
king  ( TOG.txt,9 )
manwha  ( TOG.txt,1 )
name  ( TOG.txt,2 ) ( zahard.txt,1 ) ( yash sankpal.txt,1 )
of  ( TOG.txt,5 ) ( zahard.txt,8 )
old ( zahard.txt,10 )
sankpal ( yash sankpal.txt,4 )
the ( TOG.txt,8 )
thousands ( zahard.txt,7 )
tower ( TOG.txt,4 )
yash  ( yash sankpal.txt,3 )
year  ( zahard.txt,9 )
zahard  ( zahard.txt,3 ) ( TOG.txt,11 )
```