Program 3: Writing an Inversed Indexing Program with MapReduce

Name: Pooja Nadagouda

Inverted Indexing Program:

This program takes keywords as user input and outputs count of keywords in each of the files provided.
It consists of 2 phases: Map and Reduce
In Map phase the keywords that are input from the user are read into HashSet. Each word from input files is read and compared if it is same as any of the keywords stored in HashSet. If it matches then the output collects it in the format Keyword : FileName.
In the Reduce phase we get input from the map phase, for each keyword there are set of files containing them. We use a HashMap to store the FileName (as key) and count of keywords (as value). When processing the values of reduce phase, based on the filename its corresponding count in incremented and stored in HashMap. The HashMap key-value pairs are combined using a StringBuilder and given as a single value to the output collector. Hence the output now has the keywords as key and filename along with count as value.

Screenshots for Inverted Indexing execution:

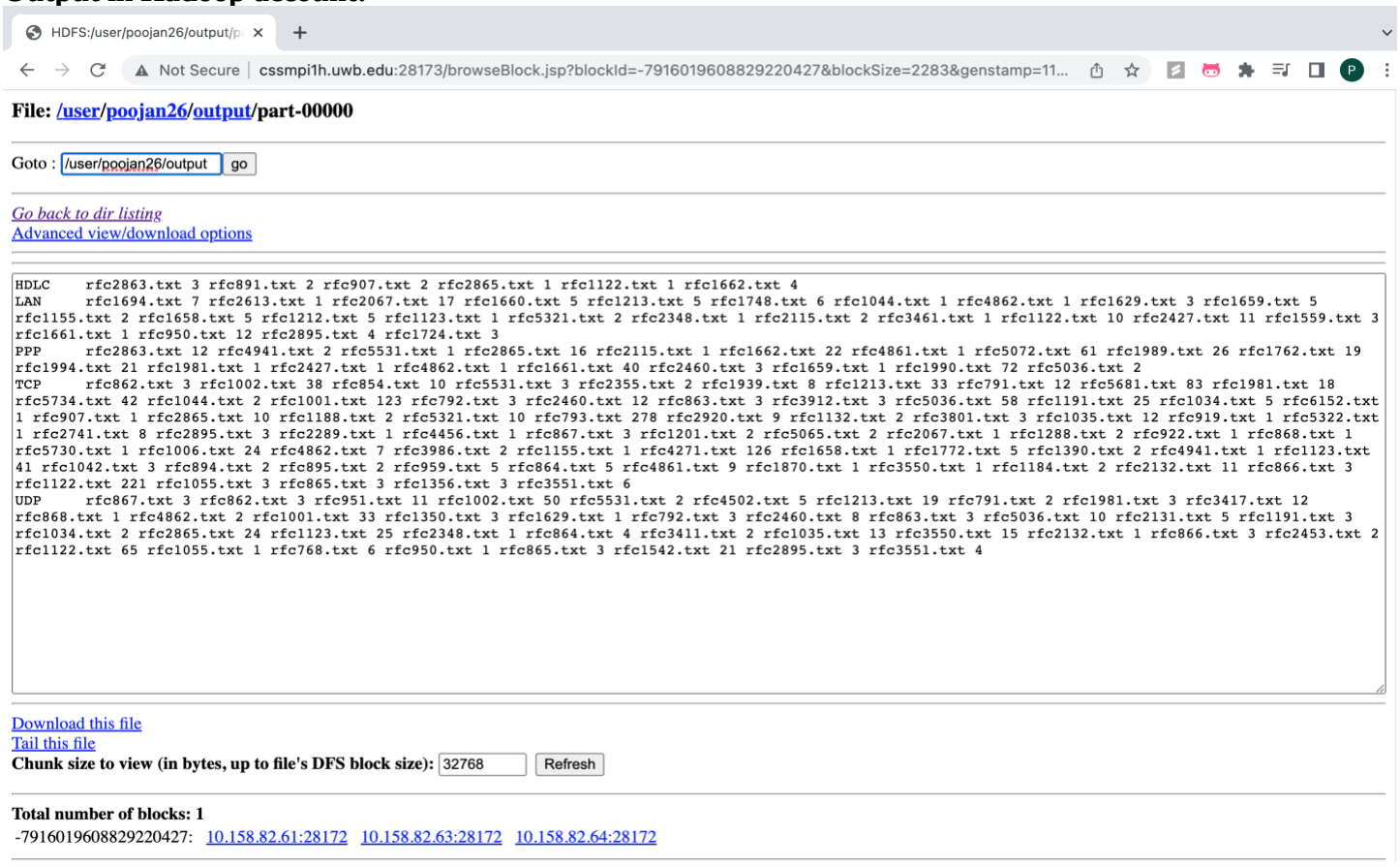
Using 4 nodes:

```
22/11/10 19:09:49 INFO mapred.JobClient:     Reduce shuffle bytes=39044
22/11/10 19:09:49 INFO mapred.JobClient:     Combine input records=0
22/11/10 19:09:49 INFO mapred.JobClient:     Map input bytes=11463932
22/11/10 19:09:49 INFO mapred.JobClient:     Reduce input groups=5
22/11/10 19:09:49 INFO mapred.JobClient:   FileSystemCounters
22/11/10 19:09:49 INFO mapred.JobClient:     HDFS_BYTES_READ=11463932
22/11/10 19:09:49 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=82704
22/11/10 19:09:49 INFO mapred.JobClient:     FILE_BYTES_READ=38144
22/11/10 19:09:49 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=2283
22/11/10 19:09:49 INFO mapred.JobClient:   Job Counters
22/11/10 19:09:49 INFO mapred.JobClient:     Launched map tasks=169
22/11/10 19:09:49 INFO mapred.JobClient:     Launched reduce tasks=1
22/11/10 19:09:49 INFO mapred.JobClient:     Rack-local map tasks=1
22/11/10 19:09:49 INFO mapred.JobClient:     Data-local map tasks=168
Time elapsed: 67201
[poojan26@cssmpi1h prog3]$ ~/hadoop-0.20.2/bin/hadoop fs -get /user/poojan26/output/part-00000 output
[poojan26@cssmpi1h prog3]$ cat output
HDLC    rfc2863.txt 3 rfc891.txt 2 rfc907.txt 2 rfc2865.txt 1 rfc1122.txt 1 rfc1662.txt 4
LAN     rfc1694.txt 7 rfc2613.txt 1 rfc2067.txt 17 rfc1660.txt 5 rfc1213.txt 5 rfc1748.txt 6 rfc1044.txt 1 rfc4862.txt 1 rfc1
629.txt 3 rfc1659.txt 5 rfc1155.txt 2 rfc1658.txt 5 rfc1212.txt 5 rfc1123.txt 1 rfc2348.txt 1 rfc5321.txt 2 rfc2115.txt 2 rfc
3461.txt 1 rfc1122.txt 10 rfc2427.txt 11 rfc1559.txt 3 rfc950.txt 12 rfc1661.txt 1 rfc2895.txt 4 rfc1724.txt 3
PPP     rfc2863.txt 12 rfc4941.txt 2 rfc5531.txt 1 rfc2865.txt 16 rfc2115.txt 1 rfc1662.txt 22 rfc4861.txt 1 rfc5072.txt 61 r
fc1989.txt 26 rfc1762.txt 19 rfc1994.txt 21 rfc1981.txt 1 rfc2427.txt 1 rfc4862.txt 1 rfc1661.txt 40 rfc1659.txt 1 rfc2460.tx
t 3 rfc1990.txt 72 rfc5036.txt 2
TCP     rfc862.txt 3 rfc854.txt 10 rfc1002.txt 38 rfc5531.txt 3 rfc2355.txt 2 rfc1939.txt 8 rfc1213.txt 33 rfc791.txt 12 rfc5
681.txt 83 rfc1981.txt 18 rfc5734.txt 42 rfc1044.txt 2 rfc1001.txt 123 rfc2460.txt 12 rfc792.txt 3 rfc863.txt 3 rfc3912.txt 3
 rfc5036.txt 58 rfc1191.txt 25 rfc1034.txt 5 rfc6152.txt 1 rfc907.txt 1 rfc2865.txt 10 rfc1188.txt 2 rfc5321.txt 10 rfc793.tx
t 278 rfc1132.txt 2 rfc2920.txt 9 rfc3801.txt 3 rfc1035.txt 12 rfc919.txt 1 rfc5322.txt 1 rfc2741.txt 8 rfc2895.txt 3 rfc2289
.txt 1 rfc4456.txt 1 rfc867.txt 3 rfc1201.txt 2 rfc5065.txt 2 rfc2067.txt 1 rfc1288.txt 2 rfc922.txt 1 rfc868.txt 1 rfc1006.t
xt 24 rfc5730.txt 1 rfc4862.txt 7 rfc3986.txt 2 rfc1155.txt 1 rfc4271.txt 126 rfc1658.txt 1 rfc1772.txt 5 rfc1390.txt 2 rfc49
41.txt 1 rfc1123.txt 41 rfc1042.txt 3 rfc894.txt 2 rfc895.txt 2 rfc959.txt 5 rfc4861.txt 9 rfc864.txt 5 rfc1870.txt 1 rfc3550
.txt 1 rfc1184.txt 2 rfc2132.txt 11 rfc866.txt 3 rfc1122.txt 221 rfc1055.txt 3 rfc865.txt 3 rfc1356.txt 3 rfc3551.txt 6
UDP     rfc862.txt 3 rfc867.txt 3 rfc951.txt 11 rfc1002.txt 50 rfc5531.txt 2 rfc4502.txt 5 rfc1213.txt 19 rfc791.txt 2 rfc341
7.txt 12 rfc1981.txt 3 rfc868.txt 1 rfc1001.txt 33 rfc4862.txt 2 rfc1629.txt 1 rfc1350.txt 3 rfc2460.txt 8 rfc792.txt 3 rfc86
3.txt 3 rfc5036.txt 10 rfc2131.txt 5 rfc1191.txt 3 rfc1034.txt 2 rfc2865.txt 24 rfc1123.txt 25 rfc2348.txt 1 rfc3411.txt 2 rf
c864.txt 4 rfc1035.txt 13 rfc3550.txt 15 rfc2132.txt 1 rfc866.txt 3 rfc2453.txt 2 rfc1122.txt 65 rfc1055.txt 1 rfc768.txt 6 r
fc950.txt 1 rfc865.txt 3 rfc1542.txt 21 rfc2895.txt 3 rfc3551.txt 4
[poojan26@cssmpi1h prog3]$
```

Using 1 node:

```
22/11/11 11:19:24 INFO mapred.JobClient:     Map output bytes=33858
22/11/11 11:19:24 INFO mapred.JobClient:     Reduce shuffle bytes=39044
22/11/11 11:19:24 INFO mapred.JobClient:     Combine input records=0
22/11/11 11:19:24 INFO mapred.JobClient:     Map input bytes=11463932
22/11/11 11:19:24 INFO mapred.JobClient:     Reduce input groups=5
22/11/11 11:19:24 INFO mapred.JobClient:   FileSystemCounters
22/11/11 11:19:24 INFO mapred.JobClient:     HDFS_BYTES_READ=11463932
22/11/11 11:19:24 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=82704
22/11/11 11:19:24 INFO mapred.JobClient:     FILE_BYTES_READ=38144
22/11/11 11:19:24 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=2283
22/11/11 11:19:24 INFO mapred.JobClient:   Job Counters
22/11/11 11:19:24 INFO mapred.JobClient:     Launched map tasks=169
22/11/11 11:19:24 INFO mapred.JobClient:     Launched reduce tasks=1
22/11/11 11:19:24 INFO mapred.JobClient:     Data-local map tasks=169
Time elapsed: 213058
[poojan26@cssmpi1h prog3]$ rm -rf output
[poojan26@cssmpi1h prog3]$ ~/hadoop-0.20.2/bin/hadoop fs -get /user/poojan26/output/part-00000 output
[poojan26@cssmpi1h prog3]$ cat output
HDLC    rfc2863.txt 3 rfc891.txt 2 rfc907.txt 2 rfc2865.txt 1 rfc1122.txt 1 rfc1662.txt 4
LAN     rfc1694.txt 7 rfc2613.txt 1 rfc2067.txt 17 rfc1660.txt 5 rfc1213.txt 5 rfc1748.txt 6 rfc1044.txt 1 rfc4862.txt 1 rfc1
629.txt 3 rfc1659.txt 5 rfc1155.txt 2 rfc1658.txt 5 rfc1212.txt 5 rfc1123.txt 1 rfc5321.txt 2 rfc2348.txt 1 rfc2115.txt 2 rfc
3461.txt 1 rfc1122.txt 10 rfc2427.txt 11 rfc1559.txt 3 rfc1661.txt 1 rfc950.txt 12 rfc2895.txt 4 rfc1724.txt 3
PPP     rfc2863.txt 12 rfc4941.txt 2 rfc5531.txt 1 rfc2865.txt 16 rfc2115.txt 1 rfc1662.txt 22 rfc4861.txt 1 rfc5072.txt 61 r
fc1989.txt 26 rfc1762.txt 19 rfc1994.txt 21 rfc1981.txt 1 rfc2427.txt 1 rfc4862.txt 1 rfc1661.txt 40 rfc1659.txt 1 rfc2460.tx
t 3 rfc1990.txt 72 rfc5036.txt 2
TCP     rfc862.txt 3 rfc1002.txt 38 rfc854.txt 10 rfc5531.txt 3 rfc2355.txt 2 rfc1939.txt 8 rfc1213.txt 33 rfc791.txt 12 rfc5
681.txt 83 rfc1981.txt 18 rfc5734.txt 42 rfc1044.txt 2 rfc1001.txt 123 rfc792.txt 3 rfc863.txt 3 rfc2460.txt 12 rfc3912.txt 3
 rfc5036.txt 58 rfc1191.txt 25 rfc1034.txt 5 rfc6152.txt 1 rfc907.txt 1 rfc1188.txt 2 rfc2865.txt 10 rfc5321.txt 10 rfc793.tx
t 278 rfc1132.txt 2 rfc2920.txt 9 rfc3801.txt 3 rfc1035.txt 12 rfc919.txt 1 rfc5322.txt 1 rfc2741.txt 8 rfc2895.txt 3 rfc2289
.txt 1 rfc4456.txt 1 rfc867.txt 3 rfc1201.txt 2 rfc5065.txt 2 rfc2067.txt 1 rfc1288.txt 2 rfc922.txt 1 rfc868.txt 1 rfc1006.t
xt 24 rfc5730.txt 1 rfc4862.txt 7 rfc3986.txt 2 rfc1155.txt 1 rfc4271.txt 126 rfc1658.txt 1 rfc1772.txt 5 rfc1390.txt 2 rfc49
41.txt 1 rfc1123.txt 41 rfc1042.txt 3 rfc894.txt 2 rfc895.txt 2 rfc864.txt 5 rfc4861.txt 9 rfc959.txt 5 rfc1870.txt 1 rfc3550
.txt 1 rfc1184.txt 2 rfc2132.txt 11 rfc866.txt 3 rfc1122.txt 221 rfc1055.txt 3 rfc865.txt 3 rfc1356.txt 3 rfc3551.txt 6
UDP     rfc862.txt 3 rfc867.txt 3 rfc951.txt 11 rfc1002.txt 50 rfc5531.txt 2 rfc4502.txt 5 rfc1213.txt 19 rfc791.txt 2 rfc341
7.txt 12 rfc1981.txt 3 rfc868.txt 1 rfc4862.txt 2 rfc1001.txt 33 rfc1629.txt 1 rfc1350.txt 3 rfc863.txt 3 rfc792.txt 3 rfc246
0.txt 8 rfc5036.txt 10 rfc1191.txt 3 rfc2131.txt 5 rfc1034.txt 2 rfc1123.txt 25 rfc2865.txt 24 rfc2348.txt 1 rfc864.txt 4 rfc
3411.txt 2 rfc1035.txt 13 rfc3550.txt 15 rfc2132.txt 1 rfc866.txt 3 rfc1122.txt 65 rfc2453.txt 2 rfc1055.txt 1 rfc768.txt 6 r
fc865.txt 3 rfc1542.txt 21 rfc950.txt 1 rfc2895.txt 3 rfc3551.txt 4
[poojan26@cssmpi1h prog3]$
```

Output in Hadoop account:



Performance:
Time taken to run Inverted Indexing using 1 node: 213058
Time taken to run Inverted Indexing using 4 nodes: 67201
Performance = 213058/67201 **= 3.17**

Code for InvertedIndexing program:

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;
import java.io.IOException;
import java.util.*;


public class InvertedIndexing {


    public static class Map extends MapReduceBase implements Mapper<LongWritable,
Text,  Text, Text> {
        JobConf conf;
        private Text word = new Text();
        private Text fileName = new Text();


        public void configure(JobConf job) {
            this.conf = job;
        }


        public void map(LongWritable docId, Text value, OutputCollector<Text, Text>
output,
                        Reporter reporter) throws IOException {
            // retrieve # keywords from JobConf
            int argc = Integer.parseInt(conf.get("argc"));

            //adding all keywords to a HashSet
            Set<String> keywords = new HashSet<>();
            for(int i=0; i<argc; i++){
                keywords.add(conf.get("keyword"+i));
            }

            // get the current file name
            FileSplit fileSplit = (FileSplit) reporter.getInputSplit();
            String filename = "" + fileSplit.getPath().getName();

            //splitting each line into words
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                String curWord = tokenizer.nextToken();
                //collecting the word if it is one among the keywords
                if (keywords.contains(curWord)){
                    word.set(curWord);
                    fileName.set(filename);
                    output.collect(word, fileName);
                }
            }
        }
    }
```

```
    public static class Reduce extends MapReduceBase implements Reducer<Text, Text, Text,
Text> {
        public void reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text>
output,
                           Reporter reporter) throws IOException {

            //using a map to keep count of keywords for each file
            HashMap<String, Integer> docWordCount = new HashMap<String, Integer>();
            while(values.hasNext()){
                String curFileName = values.next().toString();
                int count = docWordCount.getOrDefault(curFileName, 0);
                docWordCount.put(curFileName, count+1);
            }
            // finally, print it out.
            StringBuilder sb = new StringBuilder();
            for (HashMap.Entry<String, Integer> entry : docWordCount.entrySet()) {
                sb.append(entry.getKey() + " ");
                sb.append(entry.getValue() + " ");
            }
            Text docListText = new Text(sb.toString());
            output.collect(key, docListText);
        }
    }


    public static void main(String[] args) throws Exception{
        // input format:
        // hadoop jar invertedindexes.jar InvertedIndexes input output keyword1 keyword2
...
        JobConf conf = new JobConf(InvertedIndexing.class);  // AAAAA is this program's
file name
        conf.setJobName("invertedindexing");                 // BBBBB is a job name,
whatever you like

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(Text.class);

        conf.setMapperClass(Map.class);
        conf.setReducerClass(Reduce.class);

        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));  // input directory name
        FileOutputFormat.setOutputPath(conf, new Path(args[1])); // output directory name

        conf.set( "argc", String.valueOf( args.length - 2 ) );   // argc maintains
#keywords
        for ( int i = 0; i < args.length - 2; i++ )
            conf.set( "keyword" + i, args[i + 2] );              // keyword1, keyword2,
...
        long start = System.currentTimeMillis();
        JobClient.runJob(conf);
        long end = System.currentTimeMillis();
        long timeElapsed = end - start;
        System.out.println("Time elapsed: " + timeElapsed)
        }
}
```

Additional Work:
The below program is written to sort the filenames in output.
Hadoop by default sorts by the key. In case we want the values also to be sorted then we use secondary sort.
Here we use a composite key i.e., keyword+filename so that the values are sorted by filename.
But the output of map phase which is given to reduce phase should be sorted by the natural key only, hence a customGroupComparator is written which is used when providing the input for reduce phase.

Note: This program could not be fixed completely to provide correct output.

Following is my code for secondary sorting:

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

public class InvertedIndexingWithSort {

    public class CustomPartitioner extends Partitioner<KeyWordFilenamePair, Text> {
         @Override
         public int getPartition(KeyWordFilenamePair pair, Text text, int
numberOfPartitions) {
             return Math.abs(pair.keyword.hashCode() % numberOfPartitions);
         }
    }

    public class CustomGroupingComparator extends WritableComparator {
         public CustomGroupingComparator() {
                 super(KeyWordFilenamePair.class, true);
         }
         /**
           * This comparator controls which keys are grouped
           * together into a single call to the reduce() method
          */
         @Override
         //here comparing the natural keys so that the keys are sorted accordingly.
         public int compare(WritableComparable wc1, WritableComparable wc2) {
             KeyWordFilenamePair pair = (KeyWordFilenamePair) wc1;
             KeyWordFilenamePair pair2 = (KeyWordFilenamePair) wc2;
                 return pair.keyword.compareTo(pair2.keyword);
         }
    }

    public class KeyWordFilenamePair implements Writable,
WritableComparable<KeyWordFilenamePair>{
        private Text keyword = new Text();                  // natural key
        private IntWritable filename = new IntWritable(); // secondary key

        @Override
        public int compareTo(KeyWordFilenamePair pair) {
            int compareValue = this.keyword.compareTo(pair.keyword);
```

```
            if (compareValue == 0) {
                    compareValue = filename.compareTo(pair.filename);
            }
            return compareValue;
        }

        @Override
        public void write(DataOutput dataOutput) throws IOException {
            //do nothing
        }

        @Override
        public void readFields(DataInput dataInput) throws IOException {
            //do nothing
        }
    }

    public static class Map extends MapReduceBase implements Mapper<LongWritable,
Text, Text, Text> {
        JobConf conf;
        private Text compositeKey = new Text();
        private Text fileName = new Text();

        public void configure(JobConf job) {
            this.conf = job;
        }

        public void map(LongWritable docId, Text value, OutputCollector<Text, Text>
output,
                        Reporter reporter) throws IOException {
            // retrieve # keywords from JobConf
            int argc = Integer.parseInt(conf.get("argc"));

            //adding all keywords to a HashSet
            Set<String> keywords = new HashSet<>();
            for(int i=0; i<argc; i++){
                keywords.add(conf.get("keyword"+i));
            }

            // get the current file name
            FileSplit fileSplit = (FileSplit) reporter.getInputSplit();
            String filename = "" + fileSplit.getPath().getName();

            //splitting each line into words
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                String curWord = tokenizer.nextToken();
                //collecting the word if it is one among the keywords
                if (keywords.contains(curWord)){
                    //here creating composite key
                    curWord = curWord + filename;
                    compositeKey.set(curWord);
                    fileName.set(filename);
                    output.collect(compositeKey, fileName);
                }
            }

        }
    }


    public static class Reduce extends MapReduceBase implements Reducer<Text, Text, Text,
```

```
Text> {
        public void reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text>
output,
                           Reporter reporter) throws IOException {
            //using a map to keep count of keywords for each file
            HashMap<String, Integer> docWordCount = new HashMap<String, Integer>();
            while(values.hasNext()){
                String curFileName = values.next().toString();
                int count = docWordCount.getOrDefault(curFileName, 0);
                docWordCount.put(curFileName, count+1);
            }
            // finally, print it out.
            StringBuilder sb = new StringBuilder();
            for (HashMap.Entry<String, Integer> entry : docWordCount.entrySet()) {
                sb.append(entry.getKey() + " ");
                sb.append(entry.getValue() + " ");
            }
            Text docListText = new Text(sb.toString());
            output.collect(key, docListText);

        }
    }


    public static void main(String[] args) throws Exception {
        // input format:
        // hadoop jar invertedindexes.jar InvertedIndexes input output keyword1 keyword2
...
        JobConf conf = new JobConf(InvertedIndexing.class);  // AAAAA is this program's
file name
        conf.setJobName("invertedindexing");                 // BBBBB is a job name,
whatever you like

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(Text.class);

        conf.setMapperClass(InvertedIndexing.Map.class);
        //conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(InvertedIndexing.Reduce.class);
        conf.setPartitionerClass((Class<? extends org.apache.hadoop.mapred.Partitioner>)
CustomPartitioner.class);
        conf.setOutputValueGroupingComparator(CustomGroupingComparator.class);

        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));  // input directory name
        FileOutputFormat.setOutputPath(conf, new Path(args[1])); // output directory name

        conf.set( "argc", String.valueOf( args.length - 2 ) );   // argc maintains
#keywords
        for ( int i = 0; i < args.length - 2; i++ )
            conf.set( "keyword" + i, args[i + 2] );              // keyword1, keyword2,
...

        long start = System.currentTimeMillis();
        JobClient.runJob(conf);
        long end = System.currentTimeMillis();
        long timeElapsed = end - start;
        System.out.println("Time elapsed: " + timeElapsed);

    }
}
```

(1) Explain how you will distribute files over MPI ranks. How tedious is it as compared to MapReduce?

➔ There are two approaches this can be done. In the first approach data from all the files is read by the master which stores it in key-value pair i.e., FileName : FileData. Later the master distributes this FileData to other processors using MPI_Send() command. So, each processor receives full FileData of different files. For example, if there are 6 files rf1 to rf6, and 3 processors, then each processor gets 2 files

Processor 1 – rf1, rf2
Processor 2 – rf3, rf4
Processor 3 – rf5, rf6

But sending such huge data over the network can be time consuming and inefficient due to network issues.

In second approach the different files can be copied to different machines and each processor can process its own files by reading the data from the files placed on these machines.

But in case of MapReduce, the input files are added to HDFS file system, and all the nodes access the files from this location.

(2) Explain how you will collect document IDs for a given keyword in MPI. How tedious is it as compared to MapReduce?

➔ In case of MPI, each processor after its computation of counting the terms, returns the results to master. When sending the results to the master each processor needs to send MPI_Send command per file so that master accumulates it accordingly.

The master can hold an array of size equal to number of files and receives the count from each processor in the array address corresponding to that file index. For example, if there are 6 files rf1 to rf6 then an array of count 6 where array index 0 holds count results of rf1 and array index 5 holds count results of rf6. This is tedious in comparison to MapReduce as in case of MapReduce in the map phase, document ID is maintained against each keyword. No extra calculation or processing is required from programmer to keep track of document ID i.e., programmers do not need to consider workload partitioning and synchronization.

(3) Estimate the amount of boilerplate code specific to MapReduce and MPI respectively, which is irrelevant to the essence of the inverted indexing algorithm. Which would have a larger amount of boilerplate?

➔ In this case MPI will have larger boilerplate code. In case of MPI the programmer needs to take care of workload partitioning i.e., here we need to take care of dividing the file data to all processors, keep track of which document ID is assigned to which processor, and accumulation of each keyword count results from worker nodes to master node. But in case of MapReduce this is taken care by the framework.

(4) Estimate the execution performance of MapReduce and MPI respectively. Which would run faster?

➔ In general MPI performs better than MapReduce but for this problem statement I think MapReduce gives better performance when compared to MPI. For huge data processing tasks which can be performed parallelly MapReduce works faster. When data size is moderate and the problem is computation intensive, MPI is the considered better choice. When data size is large and the tasks do not require iterative processing, MapReduce is better framework.

(5) Explain how you will recover from any computational crash that may happen during inverted indexing in MPI.

➔ In case of a crash due to computational error, the execution will be stopped. The program will have to start again from the beginning with reading of data from each file and counting the frequency of the keywords.

Lab 3 – WordCount

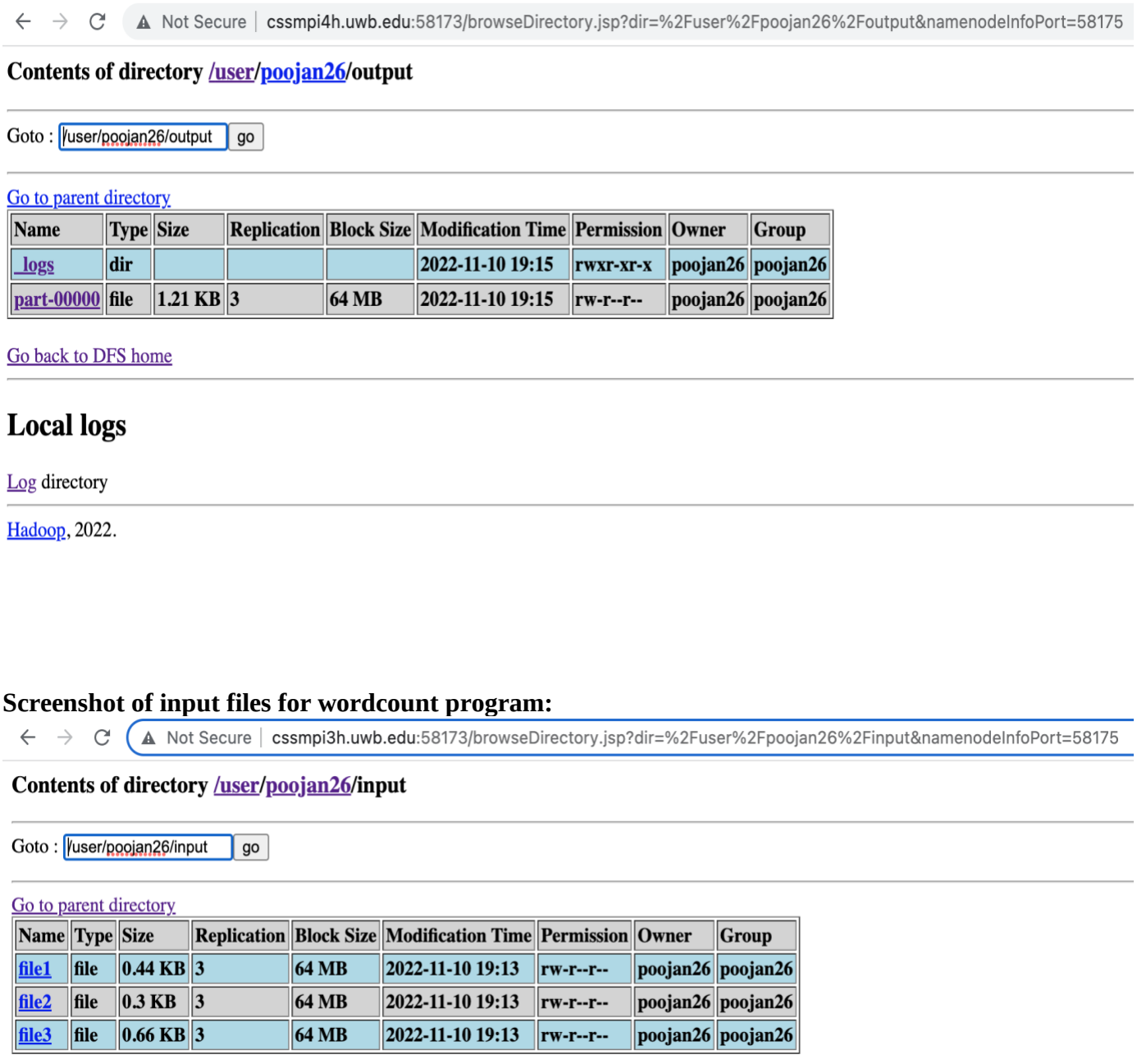
This program reads all the files and outputs the count of each word from all the files.
This program is run using 3 files which are attached as part of this zip folder (file1, file2, file3)

Following are the screen shots of execution of wordcount program:

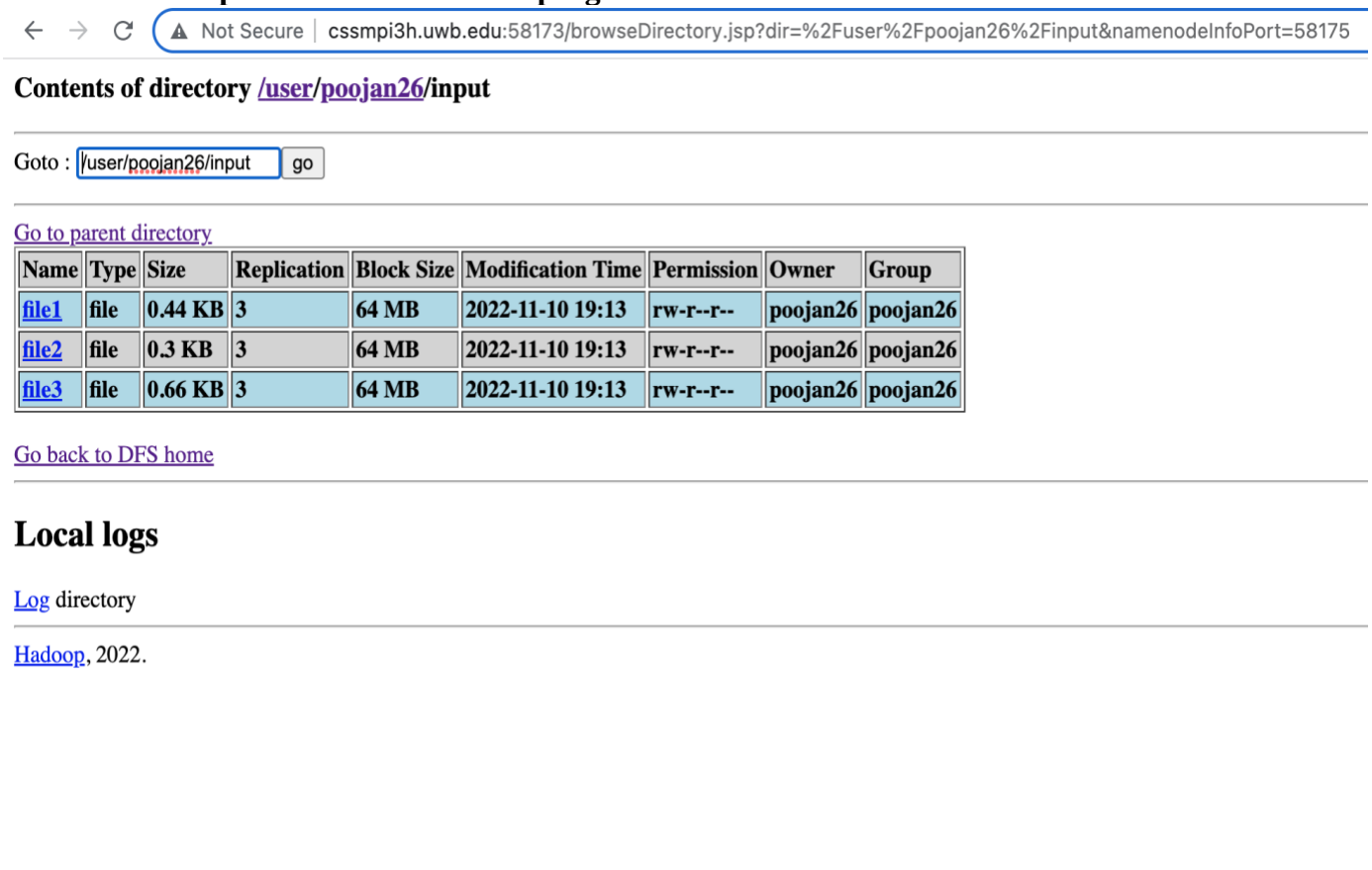
Screenshot of wordcount execution in terminal:

```
[poojan26@cssmpi1h wordcount_2.0]$ ~/hadoop-0.20.2/bin/hadoop fs -rmr /user/poojan26/output
Deleted hdfs://cssmpi1h.uwb.edu:58170/user/poojan26/output
[poojan26@cssmpi1h wordcount_2.0]$ ~/hadoop-0.20.2/bin/hadoop jar wordcount.jar WordCount input output
22/11/10 19:15:06 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement To
ol for the same.
22/11/10 19:15:07 INFO mapred.FileInputFormat: Total input paths to process : 3
22/11/10 19:15:07 INFO mapred.JobClient: Running job: job_202211101855_0005
22/11/10 19:15:08 INFO mapred.JobClient:  map 0% reduce 0%
22/11/10 19:15:17 INFO mapred.JobClient:  map 66% reduce 0%
22/11/10 19:15:18 INFO mapred.JobClient:  map 100% reduce 0%
22/11/10 19:15:29 INFO mapred.JobClient:  map 100% reduce 100%
22/11/10 19:15:31 INFO mapred.JobClient: Job complete: job_202211101855_0005
22/11/10 19:15:31 INFO mapred.JobClient: Counters: 18
22/11/10 19:15:31 INFO mapred.JobClient:   Map-Reduce Framework
22/11/10 19:15:31 INFO mapred.JobClient:     Combine output records=164
22/11/10 19:15:31 INFO mapred.JobClient:     Spilled Records=328
22/11/10 19:15:31 INFO mapred.JobClient:     Reduce input records=164
22/11/10 19:15:31 INFO mapred.JobClient:     Reduce output records=137
22/11/10 19:15:31 INFO mapred.JobClient:     Map input records=9
22/11/10 19:15:31 INFO mapred.JobClient:     Map output records=240
22/11/10 19:15:31 INFO mapred.JobClient:     Map output bytes=2394
22/11/10 19:15:31 INFO mapred.JobClient:     Reduce shuffle bytes=2101
22/11/10 19:15:31 INFO mapred.JobClient:     Combine input records=240
22/11/10 19:15:31 INFO mapred.JobClient:     Map input bytes=1438
22/11/10 19:15:31 INFO mapred.JobClient:     Reduce input groups=137
22/11/10 19:15:31 INFO mapred.JobClient:   FileSystemCounters
22/11/10 19:15:31 INFO mapred.JobClient:     HDFS_BYTES_READ=1438
22/11/10 19:15:31 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=4286
22/11/10 19:15:31 INFO mapred.JobClient:     FILE_BYTES_READ=2089
22/11/10 19:15:31 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=1239
22/11/10 19:15:31 INFO mapred.JobClient:   Job Counters
22/11/10 19:15:31 INFO mapred.JobClient:     Launched map tasks=3
22/11/10 19:15:31 INFO mapred.JobClient:     Launched reduce tasks=1
22/11/10 19:15:31 INFO mapred.JobClient:     Data-local map tasks=3
[poojan26@cssmpi1h wordcount_2.0]$ ~/hadoop-0.20.2/bin/hadoop fs -get /user/poojan26/output/part-00000 output
[poojan26@cssmpi1h wordcount_2.0]$ cat output
"MapReduce"     1
(key/value      1
```

Screenshot of Hadoop account output folder:



Screenshot of input files for wordcount program:



Screenshot of the output of wordcount program: