Parallel Programming 3 Assignment

For this program I have implemented Mapper, Combiner and Reducer methods.

Mapper :

On a high level in mapper method I have retrieved the number of keywords from runtime and using FileSplit I have divided each file into multiple chunks and have obtained the name of the file by extracting the path and name from FileSplit and created a dictionary of "*keywords*" by extracting keywords from conf object. By using this dictionary of keywords, the mapper method will be creating a list of words that are present in the chunk of document that is given to it for computation. Mapper take each word from the list and check if that word is present in the keywords dictionary and accordingly it will create another dictionary named "*keywordcount*". Mapper will then take each keyword and its value from keywordcount and will format it in *("TCP fileA", "fileA 3")* format.

Combiner:

Combiner class will be taking the output from Mapper as input and will be reducing it based on its filename and the count. Combiner will search for filenames which have same keyword and will add up its count, this newly created value will be added to "*fileToKeywordCountMap"*. If combiner receives *("TCP fileA", "fileA 3") and ("TCP fileA", "fileA 5")* as inputs then it will reduce it to *("TCP fileA", "fileA 8").*

Reducer*:*

Reducer class will be taking output from Combiner as input and will be reducing it based on the keyword. If the received input is in the form *("TCP fileA", "fileA 8") and ("TCP fileB", "fileB 10")* , then the reduced output that will be collected by reducer will be in this format *TCP : ("fileA 8", "fileB 10").*

Partitioner*:*

Partitioner method will be dividing the documents into chunks and these chunks will be given as inputs to Mapper.

Sourcecode : (InvertedIndexing2.java)

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

import java.io.*;
import java.util.*;

public class InvertedIndexing2 {

    /* Mapper method will divide the file/document into chunks and
     * map the keyword against its count.
     */
    public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text,
Text> {

        JobConf conf;
        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"));
            // dividing the file into multiple chunks
            FileSplit fileSplit = (FileSplit) reporter.getInputSplit();
            // retreiving the filename from the path obtained by the above fileSplit
            String filename = "" + fileSplit.getPath( ).getName( );
            //creating a dictionary of keywords and extracting the arguments by
            // explicitly calculating it in the main function
            Set<String> keywords = new HashSet<>();
            for ( int i = 0; i < argc; i++ ) {
                keywords.add(conf.get("keyword" + i));
            }

            // creates a dictionary where the count of keywords is stored
            java.util.Map<String, Integer> keywordCount = new HashMap<>();
            // creates a list of words in a chunk obtained from a document
            String[] words = value.toString().split(" +");

            /* Here the loop will take word from words list
             * and check if that word is already present keywords dictionary ( which
             * contains the words given during runtime). and if the word is already present
             * in keywords dictionary then we will check if that word is also present in
keywordCount
             * if yes then we will just incerement the count otherwise we will add that word
to keywordCount dictionary
             */
            for (String word : words)
            {
                if (keywords.contains(word))
                {
                    if (keywordCount.containsKey(word))
                    {
                        keywordCount.put(word, keywordCount.get(word) + 1);
                    }
                    else
                    {
                        keywordCount.put(word, 1);
                    }
                }
            }

            /* mapper will be collecting the output in a specific format i.e..
             * ("TCP fileA", "fileA count")
             */
            keywordCount.forEach((keyword, count) ->
            {
                try {
                    output.collect(new Text(keyword + SPACE + filename), new Text(filename +
SPACE + count));
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });
        }
    }
    /* Combiner method will take the output collected by the mapper as
     * input and will reduce the dictionary count by taking keyword as base
     * and add the count of the keywords with same filename.
     */
    public static class Combiner extends MapReduceBase implements Reducer<Text, Text, Text,
Text> {
        @Override
        public void reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text>
output, Reporter reporter) throws IOException {

            String keyword = key.toString().split(SPACE)[0];
            // dictionary to store the reduced values of keywords and count
            java.util.Map<String, Integer> fileToKeywordCountMap = new HashMap<>();

            /* the loop will check if there are anymore values left in the output collected
from mapper or not
             * if fileToKeywordCountMap already has the filename obtained from mapper then we
will icrement the cpunt
             * else we will add that filename with count to the fileToKeywordCountMap
dictionary.
             */
            while (values.hasNext())
            {
```

```
                Text curr = values.next();
                String filename = curr.toString().split(SPACE)[0];
                Integer count = Integer.parseInt(curr.toString().split(SPACE)[1]);

                if (fileToKeywordCountMap.containsKey(filename))
                {
                    fileToKeywordCountMap.put(filename, fileToKeywordCountMap.get(filename) +
count);
                }
                else
                {
                    fileToKeywordCountMap.put(filename, count);
                }
            }

            /* Combiner will be collecting the output in a specific format
             * i.e.. if input from mapper is : ("TCP fileA", "fileA 3") (TCP fileA", "fileA
5")
             * combiner will store it as : (TCP fileA", "fileA 8").
             */
            fileToKeywordCountMap.forEach((fileName, count) -> {
                try {
                    output.collect(new Text(keyword + SPACE + fileName), new Text(fileName +
SPACE + count));
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });
        }
    }

    /* Reduce method will be taking the output of combiner as input
     * and will do the last reduction on the data based on keyword.
     * Ex: input: (TCP fileA", "fileA 8") , (TCP fileB", "fileB 10")
     *     output: (TCP : "fileA 8", "fileB 10")
     */
    public static class Reduce extends MapReduceBase implements Reducer<Text, Text, Text,
Text> {
        private final Text word = new Text();
        public void reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text>
output, Reporter reporter) throws IOException {
            StringBuilder strBuilder = new StringBuilder();
            String keyword = key.toString().split(SPACE)[0];
            word.set(keyword);
            while (values.hasNext())
            {
                strBuilder.append(values.next().toString()).append(SPACE);
            }
            output.collect(word, new Text(strBuilder.toString()));
        }
    }

    // Partitioner method will be dividing the document into chunks
    public static class Partitioner extends MapReduceBase implements
org.apache.hadoop.mapred.Partitioner<Text, Text>
    {
        @Override
        public int getPartition(Text key, Text value, int numPartitions) {
            final String keyword = key.toString().split(SPACE)[0];
            return keyword.hashCode() % numPartitions;
        }
    }

    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(InvertedIndexing2.class);
        conf.setJobName("invertedIndexing");

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

        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Combiner.class);
        conf.setReducerClass(Reduce.class);
        conf.setPartitionerClass(Partitioner.class);

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

```
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        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] );

        JobClient.runJob(conf);
    }

    private static final String SPACE = " ";
}
```

Output Execution:

For single node :

```
[svrb@cssmpi1h invertedIndex]$ javac -classpath ${HADOOP_HOME}/hadoop-${HADOOP_VERSION}-
core.jar InvertedIndexing.java; jar -cvf invertedIndexing.jar *.class; ~/hadoop-
0.20.2/bin/hadoop fs -rmr /user/svrb/output; ~/hadoop-0.20.2/bin/hadoop jar
invertedIndexing.jar InvertedIndexing rfc output TCP UDP LAN PPP HDLC
Note: InvertedIndexing.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
added manifest
adding: InvertedIndexing.class(in = 2414) (out= 1182)(deflated 51%)
adding: InvertedIndexing$Combiner.class(in = 2229) (out= 913)(deflated 59%)
adding: InvertedIndexing$Map.class(in = 4318) (out= 1754)(deflated 59%)
adding: InvertedIndexing$Partitioner.class(in = 880) (out= 452)(deflated 48%)
adding: InvertedIndexing$Reduce.class(in = 1844) (out= 760)(deflated 58%)
rmr: cannot remove /user/svrb/output: No such file or directory.
21/11/15 10:13:33 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
21/11/15 10:13:33 INFO mapred.FileInputFormat: Total input paths to process : 169
21/11/15 10:13:34 INFO mapred.JobClient: Running job: job_202111151006_0001
21/11/15 10:13:35 INFO mapred.JobClient:  map 0% reduce 0%
21/11/15 10:13:46 INFO mapred.JobClient:  map 2% reduce 0%
21/11/15 10:13:52 INFO mapred.JobClient:  map 4% reduce 0%
21/11/15 10:13:55 INFO mapred.JobClient:  map 5% reduce 0%
21/11/15 10:13:58 INFO mapred.JobClient:  map 7% reduce 0%
21/11/15 10:14:04 INFO mapred.JobClient:  map 10% reduce 2%
21/11/15 10:14:10 INFO mapred.JobClient:  map 12% reduce 2%
21/11/15 10:14:13 INFO mapred.JobClient:  map 12% reduce 3%
21/11/15 10:14:16 INFO mapred.JobClient:  map 14% reduce 3%
21/11/15 10:14:19 INFO mapred.JobClient:  map 14% reduce 4%
21/11/15 10:14:22 INFO mapred.JobClient:  map 17% reduce 4%
21/11/15 10:14:28 INFO mapred.JobClient:  map 19% reduce 4%
21/11/15 10:14:31 INFO mapred.JobClient:  map 20% reduce 4%
21/11/15 10:14:34 INFO mapred.JobClient:  map 22% reduce 6%
21/11/15 10:14:40 INFO mapred.JobClient:  map 24% reduce 6%
21/11/15 10:14:43 INFO mapred.JobClient:  map 25% reduce 7%
21/11/15 10:14:46 INFO mapred.JobClient:  map 27% reduce 7%
21/11/15 10:14:49 INFO mapred.JobClient:  map 27% reduce 8%
21/11/15 10:14:52 INFO mapred.JobClient:  map 30% reduce 8%
21/11/15 10:14:56 INFO mapred.JobClient:  map 31% reduce 9%
21/11/15 10:14:59 INFO mapred.JobClient:  map 33% reduce 9%
21/11/15 10:15:05 INFO mapred.JobClient:  map 36% reduce 10%
21/11/15 10:15:08 INFO mapred.JobClient:  map 37% reduce 10%
21/11/15 10:15:11 INFO mapred.JobClient:  map 39% reduce 12%
21/11/15 10:15:14 INFO mapred.JobClient:  map 40% reduce 12%
21/11/15 10:15:17 INFO mapred.JobClient:  map 42% reduce 12%
21/11/15 10:15:20 INFO mapred.JobClient:  map 42% reduce 13%
21/11/15 10:15:23 INFO mapred.JobClient:  map 44% reduce 13%
21/11/15 10:15:26 INFO mapred.JobClient:  map 45% reduce 14%
21/11/15 10:15:29 INFO mapred.JobClient:  map 47% reduce 14%
21/11/15 10:15:35 INFO mapred.JobClient:  map 50% reduce 15%
21/11/15 10:15:38 INFO mapred.JobClient:  map 51% reduce 15%
21/11/15 10:15:41 INFO mapred.JobClient:  map 53% reduce 16%
21/11/15 10:15:44 INFO mapred.JobClient:  map 56% reduce 16%
21/11/15 10:15:47 INFO mapred.JobClient:  map 57% reduce 16%
21/11/15 10:15:50 INFO mapred.JobClient:  map 59% reduce 17%
21/11/15 10:15:53 INFO mapred.JobClient:  map 60% reduce 17%
21/11/15 10:15:56 INFO mapred.JobClient:  map 62% reduce 19%
21/11/15 10:15:59 INFO mapred.JobClient:  map 65% reduce 19%
```

```
21/11/15 10:16:02 INFO mapred.JobClient:  map 66% reduce 20%
21/11/15 10:16:05 INFO mapred.JobClient:  map 69% reduce 20%
21/11/15 10:16:08 INFO mapred.JobClient:  map 71% reduce 20%
21/11/15 10:16:11 INFO mapred.JobClient:  map 72% reduce 22%
21/11/15 10:16:14 INFO mapred.JobClient:  map 75% reduce 22%
21/11/15 10:16:17 INFO mapred.JobClient:  map 77% reduce 24%
21/11/15 10:16:20 INFO mapred.JobClient:  map 79% reduce 24%
21/11/15 10:16:23 INFO mapred.JobClient:  map 82% reduce 24%
21/11/15 10:16:26 INFO mapred.JobClient:  map 84% reduce 25%
21/11/15 10:16:29 INFO mapred.JobClient:  map 86% reduce 25%
21/11/15 10:16:32 INFO mapred.JobClient:  map 88% reduce 27%
21/11/15 10:16:35 INFO mapred.JobClient:  map 90% reduce 27%
21/11/15 10:16:38 INFO mapred.JobClient:  map 92% reduce 29%
21/11/15 10:16:41 INFO mapred.JobClient:  map 95% reduce 29%
21/11/15 10:16:44 INFO mapred.JobClient:  map 97% reduce 29%
21/11/15 10:16:47 INFO mapred.JobClient:  map 100% reduce 30%
21/11/15 10:16:56 INFO mapred.JobClient:  map 100% reduce 100%
21/11/15 10:16:58 INFO mapred.JobClient: Job complete: job_202111151006_0001
21/11/15 10:16:58 INFO mapred.JobClient: Counters: 18
21/11/15 10:16:58 INFO mapred.JobClient:   Map-Reduce Framework
21/11/15 10:16:58 INFO mapred.JobClient:     Combine output records=160
21/11/15 10:16:58 INFO mapred.JobClient:     Spilled Records=320
21/11/15 10:16:58 INFO mapred.JobClient:     Reduce input records=160
21/11/15 10:16:58 INFO mapred.JobClient:     Reduce output records=160
21/11/15 10:16:58 INFO mapred.JobClient:     Map input records=299992
21/11/15 10:16:58 INFO mapred.JobClient:     Map output records=2047
21/11/15 10:16:58 INFO mapred.JobClient:     Map output bytes=60713
21/11/15 10:16:58 INFO mapred.JobClient:     Reduce shuffle bytes=5217
21/11/15 10:16:58 INFO mapred.JobClient:     Combine input records=2047
21/11/15 10:16:58 INFO mapred.JobClient:     Map input bytes=11463932
21/11/15 10:16:58 INFO mapred.JobClient:     Reduce input groups=160
21/11/15 10:16:58 INFO mapred.JobClient:   FileSystemCounters
21/11/15 10:16:58 INFO mapred.JobClient:     HDFS_BYTES_READ=11463932
21/11/15 10:16:58 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=11794
21/11/15 10:16:58 INFO mapred.JobClient:     FILE_BYTES_READ=1126
21/11/15 10:16:58 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=3063
21/11/15 10:16:58 INFO mapred.JobClient:   Job Counters
21/11/15 10:16:58 INFO mapred.JobClient:     Launched map tasks=169
21/11/15 10:16:58 INFO mapred.JobClient:     Launched reduce tasks=1
21/11/15 10:16:58 INFO mapred.JobClient:     Data-local map tasks=169
[svrb@cssmpi1h invertedIndex]$
```

Total Time Taken : 205 seconds

For 4 nodes:

```
[svrb@cssmpi1h invertedIndex]$ javac -classpath ${HADOOP_HOME}/hadoop-${HADOOP_VERSION}-
core.jar InvertedIndexing.java; jar -cvf invertedIndexing.jar *.class; ~/hadoop-
0.20.2/bin/hadoop fs -rmr /user/svrb/output; ~/hadoop-0.20.2/bin/hadoop jar
invertedIndexing.jar InvertedIndexing rfc output TCP UDP LAN PPP HDLC
Note: InvertedIndexing.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
added manifest
adding: InvertedIndexing.class(in = 2414) (out= 1182)(deflated 51%)
adding: InvertedIndexing$Combiner.class(in = 2229) (out= 913)(deflated 59%)
adding: InvertedIndexing$Map.class(in = 4318) (out= 1754)(deflated 59%)
adding: InvertedIndexing$Partitioner.class(in = 880) (out= 452)(deflated 48%)
adding: InvertedIndexing$Reduce.class(in = 1844) (out= 760)(deflated 58%)
rmr: cannot remove /user/svrb/output: No such file or directory.
21/11/15 10:54:04 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
Applications should implement Tool for the same.
21/11/15 10:54:04 INFO mapred.FileInputFormat: Total input paths to process : 169
21/11/15 10:54:05 INFO mapred.JobClient: Running job: job_202111151051_0001
21/11/15 10:54:06 INFO mapred.JobClient:  map 0% reduce 0%
21/11/15 10:54:14 INFO mapred.JobClient:  map 1% reduce 0%
21/11/15 10:54:17 INFO mapred.JobClient:  map 3% reduce 0%
21/11/15 10:54:20 INFO mapred.JobClient:  map 13% reduce 0%
21/11/15 10:54:23 INFO mapred.JobClient:  map 15% reduce 0%
21/11/15 10:54:26 INFO mapred.JobClient:  map 24% reduce 0%
21/11/15 10:54:29 INFO mapred.JobClient:  map 27% reduce 5%
21/11/15 10:54:32 INFO mapred.JobClient:  map 36% reduce 8%
21/11/15 10:54:35 INFO mapred.JobClient:  map 39% reduce 12%
21/11/15 10:54:38 INFO mapred.JobClient:  map 48% reduce 12%
21/11/15 10:54:41 INFO mapred.JobClient:  map 50% reduce 12%
21/11/15 10:54:44 INFO mapred.JobClient:  map 60% reduce 13%
```

```
21/11/15 10:54:47 INFO mapred.JobClient:  map 63% reduce 13%
21/11/15 10:54:50 INFO mapred.JobClient:  map 73% reduce 16%
21/11/15 10:54:53 INFO mapred.JobClient:  map 76% reduce 16%
21/11/15 10:54:56 INFO mapred.JobClient:  map 85% reduce 25%
21/11/15 10:54:59 INFO mapred.JobClient:  map 91% reduce 25%
21/11/15 10:55:01 INFO mapred.JobClient:  map 95% reduce 25%
21/11/15 10:55:02 INFO mapred.JobClient:  map 100% reduce 25%
21/11/15 10:55:04 INFO mapred.JobClient:  map 100% reduce 28%
21/11/15 10:55:10 INFO mapred.JobClient:  map 100% reduce 100%
21/11/15 10:55:12 INFO mapred.JobClient: Job complete: job_202111151051_0001
21/11/15 10:55:12 INFO mapred.JobClient: Counters: 18
21/11/15 10:55:12 INFO mapred.JobClient:   Map-Reduce Framework
21/11/15 10:55:12 INFO mapred.JobClient:     Combine output records=160
21/11/15 10:55:12 INFO mapred.JobClient:     Spilled Records=320
21/11/15 10:55:12 INFO mapred.JobClient:     Reduce input records=160
21/11/15 10:55:12 INFO mapred.JobClient:     Reduce output records=160
21/11/15 10:55:12 INFO mapred.JobClient:     Map input records=299992
21/11/15 10:55:12 INFO mapred.JobClient:     Map output records=2047
21/11/15 10:55:12 INFO mapred.JobClient:     Map output bytes=60713
21/11/15 10:55:12 INFO mapred.JobClient:     Reduce shuffle bytes=5217
21/11/15 10:55:12 INFO mapred.JobClient:     Combine input records=2047
21/11/15 10:55:12 INFO mapred.JobClient:     Map input bytes=11463932
21/11/15 10:55:12 INFO mapred.JobClient:     Reduce input groups=160
21/11/15 10:55:12 INFO mapred.JobClient:   FileSystemCounters
21/11/15 10:55:12 INFO mapred.JobClient:     HDFS_BYTES_READ=11463932
21/11/15 10:55:12 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=11794
21/11/15 10:55:12 INFO mapred.JobClient:     FILE_BYTES_READ=1126
21/11/15 10:55:12 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=3063
21/11/15 10:55:12 INFO mapred.JobClient:   Job Counters
21/11/15 10:55:12 INFO mapred.JobClient:     Launched map tasks=169
21/11/15 10:55:12 INFO mapred.JobClient:     Launched reduce tasks=1
21/11/15 10:55:12 INFO mapred.JobClient:     Data-local map tasks=169
[svrb@cssmpi1h invertedIndex]$
```

Total time taken: 64 seconds

Execution performance : 205/64 = 3.20 times

For additional work I tried implementing shuffle and sort but I honestly couldn't get the proper output. I got the same output without the additional work as well, so I just removed the unnecessary part as it was not yielding my intended output.

Output located in: /home/NETID/svrb/hadoop-0.20.2/invertedIndex/invind2/part-5

If I implemented this code in MPI instead of MapReduce I would do the following:

1. I would initially take arguments as input from run time and would give the list of keywords as input to MPI.
    a. I would give each document to each node and would implement it dynamically by using chunks, I would select only the words that are present int keywords dictionary and would create a dictionary with name as document, where the dictionary consisted of keywords and number of occurrences of that keyword in the specific document.
    b. This would require a lot of time as each node has to go through the complete file and check for the keyword.
    c. After all the documents complete their execution then the output dictionaries should be sent to master where master will compress the dictionaries into required format.
2. Honestly it would be very difficult to obtain doc_ids, but during the time of creating the dictionaries in step 1 I would append the name of document to both Keyword and the count. This way it will be easier to acquire the doc_id.

3. It is a straightforward answer, MapReduce will definitely have more boiler plate code compared to MPI because MapReduce needs to setup some amount of code that will be common for all the users whereas MPI will be more user specific, each user can implement the code in their own manner. MapReduce doesn't give a lot of freedom to user; it restricts user by allowing them to code only in specific fashion.
4. MapReduce would be faster because in map reduce, the programming pattern itself will easily divide the files into chunks be it bigger files/ smaller files. Though the number of reads and writes are more as it is running parallelly the execution time will be reduced, whereas in MPI the number of sends/receives is large and everywhere the receiver node should send an acknowledgement saying that it has received the request for this purpose the node has to wait for some time which is a time-consuming process.
5. I honestly feel there is no way in which MPI can recover from a crash because there are no functions provided by MPI which allow user to recover the lost data. If one send/receive request goes wrong, then there is no way we can retrieve the data the whole program goes wrong and crashes, whereas in MapReduce the Mapper will be storing the data in the disk i/o , even if there is a crash the user can retrieve the data back from the disk.

Discussion of Program 3

	MapReduce	MPI
File distribution over a cluster system	Hadoop can handle multiple small clusters easily because it stores multiples copies of each cluster in lists and chunks, and each node that stores the clustered file can easily store the entire cluster of files. So, as the data transfer between the nodes is fast accessing these small clusters will be faster whereas for larger files the case will be vice versa, because accessing the larger files will be slow .	MPI works better with larger files because each file is given to a node, the node can easily handle large data and can share it with its neighbors whereas sending smaller files over MPI will result in multiple sends/receives and cause an overhead and increases the accessing time. So, MPI is better in handling larger cluster of jobs.
Collective/Reductive operation to create inverted indexing	Reduce operation performs better on inverted indexing because, once all the data in the files is mapped with the given keywords then we can easily access the HashMap's and combine all the HashMap's together, form a single key for a word and apply the reduce operation on them and obtain the output. This functionality will produce a better output.	Collective communication operations will be performed on the larger files but as mentioned above there will be a lot of overhead and even if one Send operation fails, then it will result in wrong output.
Amount of boiler plate code	Requires a lot of boilerplate code which will be common	Contains less amount of boilerplate code.

	for all the users. It has the highest percent of boilerplate code amongst all the parallelization strategies.	
Anticipated Execution performance	MapReduce communicates between nodes by disk i/o which is very fast	MPI communicates between nodes by using Message Passing technology which is slower compared to MapReduce.
Fault tolerance/recovery from crash	MapReduce has better fault tolerance than MPI because the map task writes the output to local disk, if the machine crashes, then we can again run the map task and collect the data from local disk.	MPI has very bad fault tolerance compared to MapReduce because, in MPI even if one send/receive request goes wrong then the whole process crashes and can't be recovered back. It doesn't provide any fault tolerant constructs to handle failures

Parallel Programming Lab 3

Source Code(WordCount.java) :

```
import java.io.IOException;
import java.util.*;

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.*;

public class WordCount {

    public static class Map 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);
                       }
               }
    }

    public static class Reduce 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));
       }
    }

    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(WordCount.class);
        conf.setJobName("wordcount");
```

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

        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.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);
    }
}
```

Input files:

file01:
I am srilekha Bandaru and it took me a lot of time to complete MapReduce
Mapreduce is a very tedious task and very irritating

file02:
Hello Hadoop Goodbye Hadoop
I am srilekha Bandaru and it took me a lot of time to complete MapReduce
Mapreduce is a very tedious task and very irritating

Output:

```
[svrb@cssmpi1h output]$ cd ..
[svrb@cssmpi1h wordcount_2.0]$ ~/hadoop-0.20.2/bin/hadoop fs -get /user/svrb/output/part-00000 output
[svrb@cssmpi1h wordcount_2.0]$ cd output/; ls
part-00000
[svrb@cssmpi1h output]$ cat part-00000
Bandaru 2
Goodbye 1
Hadoop  2
Hello   1
I       2
MapReduce       2
Mapreduce       2
a       4
am      2
and     4
complete        2
irritating      2
is      2
it      2
lot     2
me      2
of      2
srilekha        2
task    2
tedious 2
time    2
to      2
took    2
very    4
[svrb@cssmpi1h output]$ cd ..
```

My Observation: when I first implemented wordcount program after installing Hadoop, I had many issues. I resolved them by deleting the tmp files and output files after every run. I changed the input files in the local system and tried running it but I was not getting the intended output, and I could constantly see that in HDFS the input file count was wrong.

So,after spending lot of time I realized that I have been removing files locally and not in HDFS,so I removed all the files from HDFS by usning this command *" ~/hadoop-0.20.2/bin/hadoop fs -rmr /user/svrb/input/*"* . After removing the unnecessary files from HDFS, I again copied my newly created files to HDFS and ran them which is when I received the intended output. Overall the whole process was very tedious but gained better results.

-Srilekha Venkata Ramani Bandaru