CS 6240: Assignment 2

Name: *Mahesh Bhandarkar*
Class: *Tuesday 1:35 PM*
Homework: *HW 2*

Source Codes with Highlighted Differences

Custom-Partitioner Code

This code is same for all the variations of the Word Count program we are trying to experiment with.

My "Partitioner" class is as follows:

```
// Output types of Mapper should be same as arguments of Partitioner
    public static class customPartitioner
                  extends Partitioner<Text, IntWritable>
    {
        @Override
        /*
         * Returns m - OneOf['m', 'n', 'o', 'p', 'q']
         *  This determines the reduce partition that the word will go to
         *  Word starting from 'm' will goto partition 0
         *                     'n' will goto partition 1
         *                     'o' will goto partition 2
         *                     'p' will goto partition 3
         *                     'q' will goto partition 4
         *
         * */
        public int getPartition
                (Text key, IntWritable value, int numPartitions)
    {
            String myKey = key.toString().toLowerCase();
            char firstCharOfKey = myKey.charAt(0);
            return firstCharOfKey - 'm';
        }
    }
```

NO-Combiner Source Code

In this case we have modified the Map Method, by adding a filter 'isReal(arg)'
That checks if the mapped word is a 'Real' word according to specifications.

The Partitioner , then segregates these words into separate reduce tasks.
'm/M' go to reduce task '

My map class is as follows:

```
public static class Map extends 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,
                                               Context context)
              throws IOException, InterruptedException
          {
           String line = value.toString();
           StringTokenizer tokenizer =
                              new StringTokenizer(line);
           String strWord;
           char firstLetter;
           while (tokenizer.hasMoreTokens()) {
               word.set(tokenizer.nextToken());
               strWord = word.toString();
               firstLetter = strWord.charAt(0);
               // Check if word is real
               if (this.isReal(firstLetter))
               {
                   context.write(word, one);
               }
           }
        }



        // Checks if the Word is Real or not, by accepting its 1st Character
        private boolean isReal(char aFirstCharOfWord)
        {
         boolean result =
              Character.toLowerCase(aFirstCharOfWord) == 'm' ||
              Character.toLowerCase(aFirstCharOfWord) == 'n' ||
              Character.toLowerCase(aFirstCharOfWord) == 'o' ||
              Character.toLowerCase(aFirstCharOfWord) == 'p' ||
              Character.toLowerCase(aFirstCharOfWord) == 'q';

         return result;
        }

    }
```

My Main class for 'NOCombiner' is as follows:

```
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

        job.setJarByClass(WordCount.class);
        // Setting the Partitioner
        job.setPartitionerClass(customPartitioner.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(Map.class);
        // Setting the Combiner here
        //job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

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

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }
```

Si-Combiner Source Code

In this case, the only change from the former code, is that we introduce a reducer in the Main class, which looks like: "`job.setCombinerClass(Reduce.class);` "

My map class for Si-Combiner is as follows:

```
The Si-Combiner Map class is same as that of the
No-Combiner.
```

My Main class for 'Si-Combiner' is as follows:

```
public static void main(String[] args) throws Exception {
         Configuration conf = new Configuration();

         Job job = new Job(conf, "wordcount");

         job.setJarByClass(WordCount.class);
         // Setting the Partitioner
         job.setPartitionerClass(customPartitioner.class);
         job.setOutputKeyClass(Text.class);
         job.setOutputValueClass(IntWritable.class);

         job.setMapperClass(Map.class);
         // Setting the Combiner here
         job.setCombinerClass(Reduce.class);
         job.setReducerClass(Reduce.class);

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

         FileInputFormat.addInputPath(job, new Path(args[0]));
         FileOutputFormat.setOutputPath(job, new Path(args[1]));

         job.waitForCompletion(true);
      }
```

PerMapTally - Source Code

In this case we introduce a HashMap to work as an in-map combiner. This map gets initialized per-map-call, hence the name PerMapTally.
We need to comment out the combiner from the main class, in orfer to see the hashmap work as a in-mapper combiner.

My map class for PerMapTally is as follows:

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        private IntWritable iwCount = new IntWritable();

        //  localMap store count for each map call
    private HashMap<String, Integer> localMap;

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

          // Local map
          localMap = new HashMap<String, Integer>();

          String line = value.toString();
            StringTokenizer tokenizer =
                          new StringTokenizer(line);
            String strWord;
            char firstLetter;
            while (tokenizer.hasMoreTokens())
            {
                word.set(tokenizer.nextToken());
                strWord = word.toString();
                firstLetter = strWord.charAt(0);
                // Check if word is real
                if (this.isReal(firstLetter))
                {
                    if (!localMap.containsKey(strWord))
                    {
                        localMap.put(strWord,1);
                    }
                    else
                    {
                        localMap.put(strWord, localMap.get(strWord) + 1);
                    }
                }
            }

            // EMIT to Context from Local Map
            for (String wrd : localMap.keySet())
            {
                word.set(wrd);
                iwCount.set(localMap.get(wrd));
                context.write(word, iwCount);
            }
        }

        // Checks if the Word is Real or not, by accepting its 1st Character
        private boolean isReal(char aFirstCharOfWord)
        {
         boolean result =
                Character.toLowerCase(aFirstCharOfWord) == 'm' ||
                Character.toLowerCase(aFirstCharOfWord) == 'n' ||
                Character.toLowerCase(aFirstCharOfWord) == 'o' ||
                Character.toLowerCase(aFirstCharOfWord) == 'p' ||
                Character.toLowerCase(aFirstCharOfWord) == 'q';

         return result;
        }

    }.
```

My Main class for ‘PerMapTally’ is as follows:

```
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

        job.setJarByClass(WordCount.class);
        // Setting the Partitioner
        job.setPartitionerClass(customPartitioner.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(Map.class);
        // Setting the Combiner here
        // job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

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

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }
```

PerTaskTally - Source Code

In this case we use the previous Hashmap, but only to initialize it at the setup(), and Emit on cleanup().

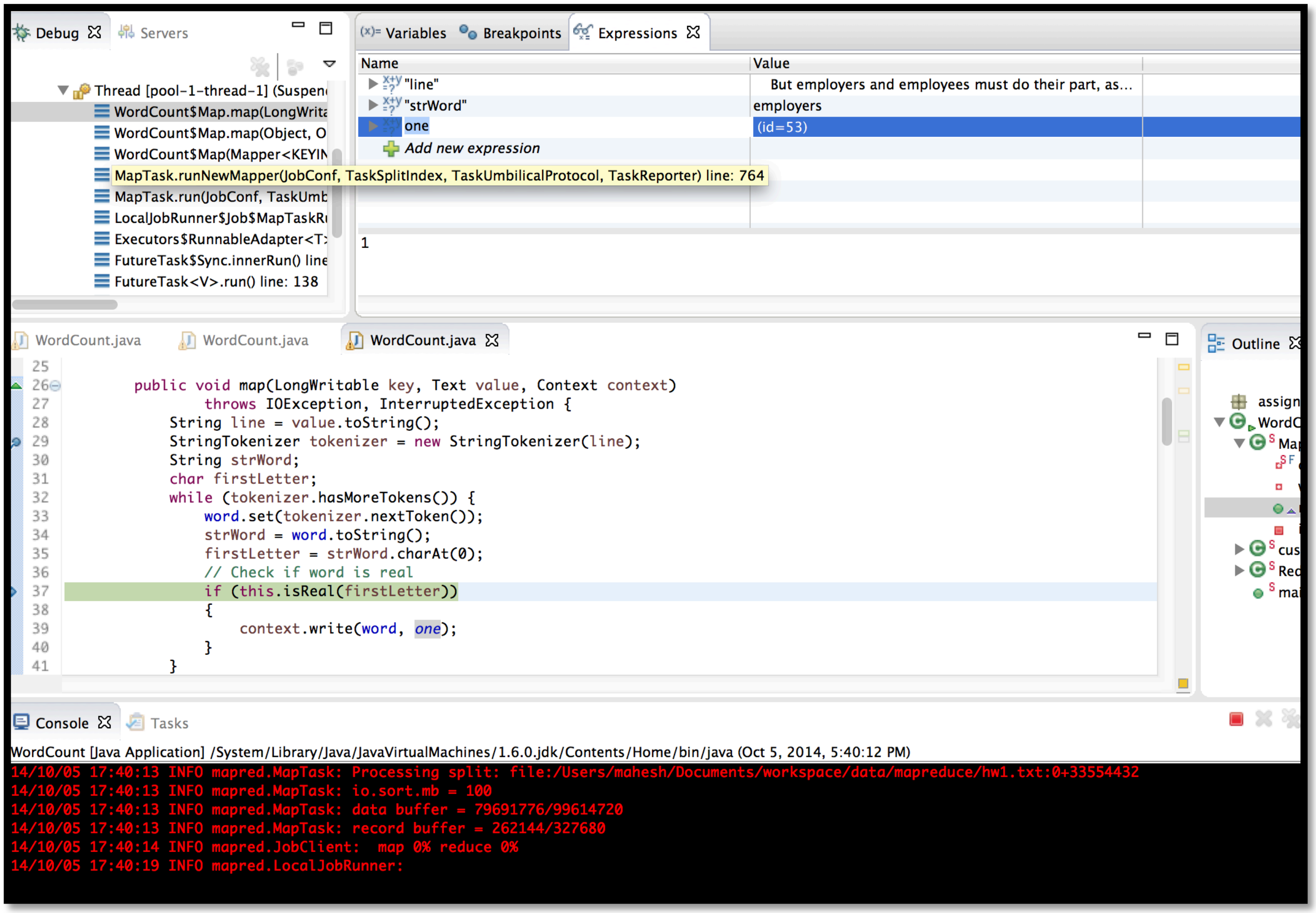
My map class for PerMapTally is as follows:

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        private IntWritable iwCount = new IntWritable();

        //  localMap store count for each map call
    private HashMap<String, Integer> localMap;

    public void setup(Context context) throws IOException,
InterruptedException
   {
         localMap = new HashMap<String, Integer>();
   }

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

         String line = value.toString();
           StringTokenizer tokenizer = new StringTokenizer(line);
           String strWord;
           char firstLetter;
           while (tokenizer.hasMoreTokens())
           {
                word.set(tokenizer.nextToken());
                strWord = word.toString();
                firstLetter = strWord.charAt(0);
                // Check if word is real
                if (this.isReal(firstLetter))
                {
                    if (!localMap.containsKey(strWord))
                    {
                         localMap.put(strWord,1);
                    }
                    else
                    {
                         localMap.put(strWord, localMap.get(strWord) + 1);
                    }
                }
           }
        }
        // Checks if the Word is Real or not, by accepting its 1st Character
        private boolean isReal(char aFirstCharOfWord)
        {
         boolean result =
                  Character.toLowerCase(aFirstCharOfWord) == 'm' ||
                  Character.toLowerCase(aFirstCharOfWord) == 'n' ||
                  Character.toLowerCase(aFirstCharOfWord) == 'o' ||
                  Character.toLowerCase(aFirstCharOfWord) == 'p' ||
                  Character.toLowerCase(aFirstCharOfWord) == 'q';

         return result;
        }

        // EMITTS to the Context
        protected void cleanup (Context context)
                           throws IOException, InterruptedException
        {
         // EMITT to Context for each line in main text
           for (String wrd : localMap.keySet())
           {
              word.set(wrd);
              iwCount.set(localMap.get(wrd));
              //emitt
              context.write(word, iwCount);
           }
        }

      }
```

My Main class for 'PerMapTally' is as follows:
Same as main class for the PerMapTally, as we need the partitioner and nothing but the Hashmap that works as a combiner upon each map task..

```
public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

        job.setJarByClass(WordCount.class);
        // Setting the Partitioner
        job.setPartitionerClass(customPartitioner.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(Map.class);
        // Setting the Combiner here
        // job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

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

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
```

Explanation – Considering the Debug perspective



```
Thread [pool-1-thread-1] (Suspen
    WordCount$Map.map(LongWrita
    WordCount$Map.map(Object, O
    WordCount$Map(Mapper<KEYIN
    MapTask.runNewMapper(JobConf, TaskSplitIndex, TaskUmbilicalProtocol, TaskReporter) line: 764
    MapTask.run(JobConf, TaskUmb
    LocalJobRunner$Job$MapTaskRu
    Executors$RunnableAdapter<T>
    FutureTask$Sync.innerRun() line
    FutureTask<V>.run() line: 138
```

Name	Value
"line"	But employers and employees must do their part, as...
"strWord"	employers
one	(id=53)
Add new expression	

```
1
```

```java
        public void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            String strWord;
            char firstLetter;
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                strWord = word.toString();
                firstLetter = strWord.charAt(0);
                // Check if word is real
                if (this.isReal(firstLetter))
                {
                    context.write(word, one);
                }
            }
```

```
WordCount [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Oct 5, 2014, 5:40:12 PM)
14/10/05 17:40:13 INFO mapred.MapTask: Processing split: file:/Users/mahesh/Documents/workspace/data/mapreduce/hw1.txt:0+33554432
14/10/05 17:40:13 INFO mapred.MapTask: io.sort.mb = 100
14/10/05 17:40:13 INFO mapred.MapTask: data buffer = 79691776/99614720
14/10/05 17:40:13 INFO mapred.MapTask: record buffer = 262144/327680
14/10/05 17:40:14 INFO mapred.JobClient:  map 0% reduce 0%
14/10/05 17:40:19 INFO mapred.LocalJobRunner:
```

In the above snapshot, you can see a debug perspective. This is the debug perspective I got, when I was single stepping the No-Combiner program

In the Expressions tab, you can see Name-Value Pair,
Now when the word count program is run, and we single step through the program, I paused at a step to analyze the situation.

The '**line**' is the variable that stores the line from the input text, for each Map-call,
The 'line' has the following value:

```
"    But employers and employees must do their part, as well, as they
are"
```

'**one**' is the variable with the value **1**, as we count 1 for each word in the map.
'**strWord**' is a variable that has the string form of the 'Text' from the incoming input.
strWord in this case has a string '**employers**' stored in it, which according to the specifications is NOT a 'real' word, and would be ignored by out map class.

Performance Analysis:

Analyzing the performance of the 4 variations of word count that are run in 2 configurations.
We run these variations twice on the given configurations.

Configurations are as follows:

Configuration 1: 6 small machines (1 master, 5 workers)
Configuration 2: 11 small machines (1 master, 10 workers)

Table of Analysis:

Variations of WordCount	Configuration 1		Configuration 2	
	1st Run (seconds)	2nd Run (seconds)	1st Run (seconds)	2nd Run (seconds)
NoCombiner	341	339	296	272
SiCombiner	322	301	233	219
PerMapTally	328	371	291	261
PerTaskTally	252	265	196	192

These values were taken from the **controller** log file that was generated after the programs ran on AWS.

Do you believe Combiner was called in the Si Combiner?

Yes.

For an instance I analyzed the Syslog, when I ran the NoCombiner and SiCombiner on the 1-Master-10-Worker configuration.

I analyzed the syslog files for both program-runs, and found following numbers that convince me that the combiner did run in the SiCombiner.
(Taken from 2nd configuration : 10 Worker, 1 Master 2nd Run)

Syslog Entry: NoCombiner

```
    Combine input records=0
     Combine output records=0
```

Syslog Entry: SiCombiner

```
    Combine input records=42989997
     Combine output records=166275
```

What difference did the use of Combiner make in the Si Combiner, as compared to the NoCombiner?

There is sure a good amount of difference, the use of a combiner makes.

Analyzing the syslofs for both **SiCombiner** and **NoCombiner** I found that:
(Taken from 2nd configuration : 10 Worker, 1 Master 2nd Run)

Syslog entry : NoCombiner

```
     Reduce input records=42842400
```

Syslog entry : SiCombiner

```
     Reduce input records=18678
```

Clearly we can see a significant change in the input size the Reducer is receiving. Apparently the ***reducer*** in **SiCombiner** gets ***significantly less input***, as compared to **NoCombiner**.

Nevertheless, we also have already analyzed the time taken for **NoCombiner** and **SiCombiner**, to run on AWS, and SiCombiner clearly runs faster than NoCombiner.

Variations of WordCount	Configuration 1		Configuration 2	
	1st Run (seconds)	2nd Run (seconds)	1st Run (seconds)	2nd Run (seconds)
NoCombiner	341	339	296	272
SiCombiner	322	301	233	219

Was the local aggregation effective in PerMapTally compared to NoCombiner?

Variations of WordCount	Configuration 1		Configuration 2	
	1st Run (seconds)	2nd Run (seconds)	1st Run (seconds)	2nd Run (seconds)
NoCombiner	341	339	296	272
PerMapTally	328	371	291	261

I believe most of the times, PerMapTally, should be faster than NOCombiner. The numbers in the above table make me believe that.

But in the 2nd run for Configuration 1, the NoCombiner seemed to be faster then PerMapTally.

To conclude, I would say that PerMapTally doesn't make a significant difference by employing local aggregation.

Also to bolster the statement, the syslog for both implementations on 2nd config says, that

PerMapTally's map emits: Map output bytes=396549900

Wheras, NoCombiner's map emits: Map output bytes=412253400

What differences do you see between PerMapTally and PerTaskTally? Try to explain the reason.

The following are the entries I analyzed from the Syslogs of the respective variations of Word Count :
(Taken from 2nd configuration: 10 Worker, 1 Master 2nd Run)

PerTaskTally

```
FILE_BYTES_READ=97223

FILE_BYTES_WRITTEN=1277758

Map output materialized bytes=191082

Spilled Records=37356
```

PerMapTally

```
FILE_BYTES_READ=74564534
 FILE_BYTES_WRITTEN=83605195
 Map output materialized bytes=27142199
 Spilled Records=122598900
```

Clearly you could see that, the File-bytes Read and written, the Spilled Records are far less in PerTaskTally, than in PerMapTally

Also, from the *Map Output Materialized Bytes,* you could observe the fact that there are far less number of bytes written on to the disk in PerTaskTally, than in PerMapTally

Nevertheless, the *PerTaskTally* runs faster than *PerMapTask* as you can see in the following Table:

Variations of WordCount	**Configuration 1**		**Configuration 2**	
	1st Run (seconds)	2nd Run (seconds)	1st Run (seconds)	2nd Run (seconds)
PerMapTally	328	371	291	261
PerTaskTally	252	265	196	192

The main reason why the map in ***PerTaskTally***, outputs lesser byte of data, is because the mapper gets dose some computation on its end, which saves the reducer from doing more work.

The mapper of ***PerTaskTally*** maps the words, and counts the **repeated occurrences** of the words, thus delivering a word and its count in that particular **map task**, to the reducer.

Wheras in the ***PerMapTaly***, the word mapping is done for every line, as we traverse through the massive text, this is much costlier than the way PerTaskTally does it. Thus it takes more time.

Also, ***PerMapTaly*** maps the words, and counts the **repeated occurrences** of the words, thus delivering a word and its count in that particular **map call**, to the reducer. Now some times this may take same time as NoCombiner, considering a case where there are no repeated words in a line.

Moreover, since the Reducer in the ***PerTaskTally*** has lesser records to process, is faster than ***PerMapTaly.***

Which one is better: SiCombiner or PerTaskTally? Briefly justify your answer.

Variations of <u>WordCount</u>	Configuration 1		Configuration 2	
	1st Run (seconds)	2nd Run (seconds)	1st Run (seconds)	2nd Run (seconds)
SiCombiner	322	301	233	219
PerTaskTally	252	265	196	192

Considering the above table, we see that PerTaskTally is a clear winner as far as Speed goes.

Now let us consider the entries from Syslog, for each of them:

SiCombiner:

```
    FILE_BYTES_READ=4416070

     FILE_BYTES_WRITTEN=2708373

     Map output bytes=412253400

     Map output records=42842400

     Spilled Records=184953
```

PerTaskTally:

```
    FILE_BYTES_READ=97223

     FILE_BYTES_WRITTEN=1277758

     Map output bytes=229702

     Map output records=18678

     Spilled Records=37356
```

We observe that the ***PerTaskTally*** has **lower** File Bytes Read, File Bytes Written, Map Output Bytes, Map Output Records and Split Records.
Hence I would regard ***PerTaskTally*** better than ***SiCombiner***.

Comparing the results for configuration 1 & 2, do you believe this MapReduce program scales well to larger cluster? Briefly justify your answer.

While we observe a drastic change in the execution times of these MapReduce programs when on configuration1 versus vonfiguration2, to check for the scalability of this MapReduce program on to a larger cluster, (i.e, from configuration 1 to configuration 2), it is not enough to have the same data set.

As we double the Worker machines, we would also need to double the Data set, as an input to the cluster.

We would require comparing an execution of a MapReduce program, with double data.

With the information we poses:

Variations of WordCount	**Configuration 1**		**Configuration 2**	
	1st Run (seconds)	2nd Run (seconds)	1st Run (seconds)	2nd Run (seconds)
NoCombiner	341	339	296	272
SiCombiner	322	301	233	219
PerMapTally	328	371	291	261
PerTaskTally	252	265	196	192

We can say that, on a larger Cluster, we get a reduced time of execution.

This observation in a way suggests the fact that on increasing the input to double, we still would achieve an approximate similar time of execution, if not same. Hence it would be safe to infer that the MapReduce program scales well.