Source Code

1. NoCombiner

```
public class WordCount_NoCombiner {

   public static void main(String[] args) throws Exception {
       Configuration conf = new Configuration();
       GenericOptionsParser parser = new GenericOptionsParser(conf, args);
       String[] otherArgs = parser.getRemainingArgs();
       if (otherArgs.length != 2) {
           System.err.println("Usage: wordcount <in> <out>");
           System.exit(2);
       }
       Job job = new Job(conf, "word count");
       job.setJarByClass(WordCount_NoCombiner.class);
       job.setMapperClass(TokenizerMapper.class);
       // disable combiner
       //job.setCombinerClass(IntSumReducer.class);
       job.setPartitionerClass(WordPartitioner.class);
       job.setNumReduceTasks(5);
       job.setReducerClass(IntSumReducer.class);
       job.setOutputKeyClass(Text.class);
       job.setOutputValueClass(IntWritable.class);
       FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
       FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
       System.exit(job.waitForCompletion(true) ? 0 : 1);
   }

   public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

       private final static IntWritable one = new IntWritable(1);
       private Text word = new Text();

       public void map(Object key, Text value, Context context)
               throws IOException, InterruptedException {
           StringTokenizer itr = new StringTokenizer(value.toString());
           while (itr.hasMoreTokens()) {
               String w = itr.nextToken();
               word.set(w);
               if(!w.isEmpty() && isValidWord(w)){
                   context.write(word, one);
               }
           }
       }

       /**
        * check if string is valid word or not
        *
        * @param word String representing the word
```

```
         * @return Boolean true only iff word starts from m,n,o,p,q letters
         */
        private boolean isValidWord(String word) {
            char c = Character.toLowerCase(word.charAt(0));
            return c >= 'm' && c <= 'q';
        }

    }

    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
                           Context context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    /**
     * WordPartitioner assign the reduce tasks to reducers
     *
     * @author jalpanranderi
     */
    public static class WordPartitioner extends Partitioner<Text, IntWritable> {
        @Override
        public int getPartition(Text key, IntWritable value, int numPartitions) {
            switch (Character.toLowerCase(key.charAt(0))) {
                case 'm': return 0;
                case 'n': return 1 % numPartitions;
                case 'o': return 2 % numPartitions;
                case 'p': return 3 % numPartitions;
                case 'q': return 4 % numPartitions;
            }
            // unknown word came into the map should not happen
            throw new IllegalStateException();
        }
    }
}
```

2. SiCombiner

```
public class WordCount_SiCombiner {

   public static void main(String[] args) throws Exception {
       Configuration conf = new Configuration();
       GenericOptionsParser parser = new GenericOptionsParser(conf, args);
       String[] otherArgs = parser.getRemainingArgs();
       if (otherArgs.length != 2) {
           System.err.println("Usage: wordcount <in> <out>");
           System.exit(2);
       }
       Job job = new Job(conf, "word count");
       job.setJarByClass(WordCount_SiCombiner.class);
       job.setMapperClass(TokenizerMapper.class);
       job.setCombinerClass(IntSumReducer.class);
       job.setPartitionerClass(WordPartition.class);
       job.setNumReduceTasks(5);
       job.setReducerClass(IntSumReducer.class);
       job.setOutputKeyClass(Text.class);
       job.setOutputValueClass(IntWritable.class);
       FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
       FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
       System.exit(job.waitForCompletion(true) ? 0 : 1);
   }

   public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {
       private final static IntWritable one = new IntWritable(1);
       private Text word = new Text();
       public void map(Object key, Text value, Context context)
               throws IOException, InterruptedException {
           StringTokenizer itr = new StringTokenizer(value.toString());
           while (itr.hasMoreTokens()) {
               word.set(itr.nextToken());
               if (!word.toString().isEmpty() && isValidWord(word.toString())) {
                   context.write(word, one);
               }
           }
       }

       /**
        * check if string is valid word or not
        *
        * @param word String representing the word
        * @return Boolean true only iff word starts from m,n,o,p,q letters
        */
       private boolean isValidWord(String word) {
           char c = Character.toLowerCase(word.charAt(0));
           return c >= 'm' && c <= 'q';
```

```
        }
    }

    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
                           Context context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    /**
     * WordPartition assign the reduce tasks to reducers
     */
    public static class WordPartition extends Partitioner<Text, IntWritable> {
        @Override
        public int getPartition(Text key, IntWritable value, int numPartitions) {
            switch (Character.toLowerCase(key.charAt(0))) {
                case 'm': return 0;
                case 'n': return 1 % numPartitions;
                case 'o': return 2 % numPartitions;
                case 'p': return 3 % numPartitions;
                case 'q': return 4 % numPartitions;
            }
            // unknown word came into the map should not happen
            throw new IllegalStateException();
        }
    }
}
```

3. PerMapTally

```
public class WordCount_PerMapTally {

   public static void main(String[] args) throws Exception {
       Configuration conf = new Configuration();
       GenericOptionsParser parser = new GenericOptionsParser(conf, args);
       String[] otherArgs = parser.getRemainingArgs();
       if (otherArgs.length != 2) {
           System.err.println("Usage: wordcount <in> <out>");
           System.exit(2);
       }
       Job job = new Job(conf, "word count");
       job.setJarByClass(WordCount_PerMapTally.class);
       job.setMapperClass(TokenizerMapper.class);

       // disable combiner
       // job.setCombinerClass(IntSumReducer.class);
       job.setPartitionerClass(WordPartitioner.class);
       job.setNumReduceTasks(5);
       job.setReducerClass(IntSumReducer.class);
       job.setOutputKeyClass(Text.class);
       job.setOutputValueClass(IntWritable.class);
       FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
       FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
       System.exit(job.waitForCompletion(true) ? 0 : 1);
   }

   public static class TokenizerMapper extends
           Mapper<Object, Text, Text, IntWritable> {
       @Override
       public void map(Object key, Text value, Context context)
               throws IOException, InterruptedException {
           // local state to save the words on Hashmap
           HashMap<String, Integer> hashMap = new HashMap<String, Integer>();
           String[] words = value.toString().split(" ");
           for (String s : words) {
               if (!s.isEmpty() && isValidWord(s)) {
                   if (hashMap.containsKey(s)) {
                       hashMap.put(s, hashMap.get(s) + 1);
                   } else {
                       hashMap.put(s, 1);
                   }
               }
           }
           // emiting
           for (String w : hashMap.keySet()) {
               context.write(new Text(w), new IntWritable(hashMap.get(w)));
```

```
            }
        }

        /**
         * check if string is valid word or not
         *
         * @param word String representing the word
         * @return Boolean true only iff word starts from m,n,o,p,q letters
         */
        private boolean isValidWord(String word) {
            char c = Character.toLowerCase(word.charAt(0));
            return c >= 'm' && c <= 'q';
        }
    }

    public static class IntSumReducer extends
            Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
                           Context context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    /**
     * WordPartitioner assign the reduce tasks to reducers
     *
     * @author jalpanranderi
     */
    public static class WordPartitioner extends Partitioner<Text, IntWritable> {

        @Override
        public int getPartition(Text key, IntWritable value, int numPartitions) {
            switch (Character.toLowerCase(key.charAt(0))) {
                case 'm': return 0;
                case 'n': return 1 % numPartitions;
                case 'o': return 2 % numPartitions;
                case 'p': return 3 % numPartitions;
                case 'q': return 4 % numPartitions;
            }
            // unknown word came into the map should not happen
            throw new IllegalStateException();
        }
    }
```

```
}
```

4. PerTaskTally

```
public class WordCount_PerTaskTally {

   public static void main(String[] args) throws Exception {
       Configuration conf = new Configuration();
       GenericOptionsParser parser = new GenericOptionsParser(conf, args);
       String[] otherArgs = parser.getRemainingArgs();
       if (otherArgs.length != 2) {
           System.err.println("Usage: wordcount <in> <out>");
           System.exit(2);
       }
       Job job = new Job(conf, "word count");
       job.setJarByClass(WordCount_PerTaskTally.class);
       job.setMapperClass(TokenizerMapper.class);
       // disable combiner
       // job.setCombinerClass(IntSumReducer.class);
       job.setPartitionerClass(WordPartitioner.class);
       job.setNumReduceTasks(5);
       job.setReducerClass(IntSumReducer.class);
       job.setOutputKeyClass(Text.class);
       job.setOutputValueClass(IntWritable.class);
       FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
       FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
       System.exit(job.waitForCompletion(true) ? 0 : 1);
   }

   public static class TokenizerMapper extends
           Mapper<Object, Text, Text, IntWritable> {
       private HashMap<String, Integer> hashMap;

       @Override
       protected void setup(
               Context context)
               throws IOException, InterruptedException {
           super.setup(context);
           hashMap = new HashMap<String, Integer>();
       }

       @Override
       public void map(Object key, Text value, Context context)
               throws IOException, InterruptedException {

           String[] words = value.toString().split(" ");
           for (String s : words) {
               if (!s.isEmpty() && isValidWord(s)) {
                   if (hashMap.containsKey(s)) {
```

```
                        hashMap.put(s, hashMap.get(s) + 1);
                    } else {
                        hashMap.put(s, 1);
                    }
                }
            }
        }

        /**
         * check if string is valid word or not
         *
         * @param word String representing the word
         * @return Boolean true only iff word starts from m,n,o,p,q letters
         */
        private boolean isValidWord(String word) {
            char c = Character.toLowerCase(word.charAt(0));
            return c >= 'm' && c <= 'q';
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            super.cleanup(context);
            for (String w : hashMap.keySet()) {
                context.write(new Text(w), new IntWritable(hashMap.get(w)));
            }
        }

    }

    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    /**
     * WordPartitioner assign the reduce tasks to reducers
     *
     * @author jalpanranderi
     */
    public static class WordPartitioner extends Partitioner<Text, IntWritable> {
```

```
        @Override
        public int getPartition(Text key, IntWritable value, int numPartitions) {
            switch (Character.toLowerCase(key.charAt(0))) {
                case 'm': return 0;
                case 'n': return 1 % numPartitions;
                case 'o': return 2 % numPartitions;
                case 'p': return 3 % numPartitions;
                case 'q': return 4 % numPartitions;
            }
            // unknown word came into the map should not happen
            throw new IllegalStateException();
        }

    }
}
```

Questions:

1. Explain what the input "key" and input "value" of the provided Word Count's Map function are and state how you found this answer.

In word count program, "key" parameter in map function is offset of Inputsplit Id or more generally it is document id from where the line is read, and "value" parameter is the actual text from that split.

I found this answer from the source code and textbook,
The documentation of Hadoop says "*Maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks which transform input records into a intermediate records. ... The Hadoop Map-Reduce framework spawns one map task for each {@link InputSplit} generated by the {@link InputFormat} for the job.*"

which is backed by the textbook in which it state that *"Input key-values pairs take the form of (docid, doc) pairs stored on distributed file system, where the former is unique identifier for the document and the letter is text of the document itself."*

Performance Comparison:

Time Chart
Formula: running time = Time (Map 100%, Reduce 100%) - Time(Map 0%, Reduce 0%)

Configuration	No Combiner	Si Combiner	PerTaskTally	PerMapTally
6 Machine (run 1)	05 min 54 sec	05 min 00 sec	04 min 37 sec	05 min 11 sec
6 Machines (run 2)	05 min 46 sec	04 min 36 sec	05 min 02 sec	05 min 13 sec
12 Machines (run 1)	04 min 28 sec	03 min 58 sec	03 min 47 sec	04 min 23 sec
12 Machines (run 2)	04 min 28 sec	03 min 42 sec	03 min 58 sec	04 min 21 sec

Input/Output Records Chart

	Combine input records	Combine output records	Map Input records	Map output records	Map output bytes	Reduce input records	Reduce output records	Reduce input groups
NoCombiner	**0**	**0**	**21907700**	**42842400**	**412253400**	**42842400**	**849**	**849**
SiCombiner	**42989997**	**166275**	**21907700**	**42842400**	**412253400**	**18678**	**849**	**849**
PerMapTally	**0**	**0**	**21907700**	**40866300**	**396549900**	**40866300**	**849**	**849**
PerTaskTally	**0**	**0**	**21907700**	**18678**	**229702**	**18678**	**849**	**849**

1. Do you believe the combiner was called at all in program SiCombiner?
Yes I believe combiner was called on SiCombiner,
the syslog file of SiCombiner run says “Combine input records=42989997”,
the syslog file of the NoCombiner run says “Combine input records=0“
so from this we can say that combiner is called in SiCombiner.

2. What difference did the use of a combiner make in SiCombiner compared to NoCombiner?
From the syslog files of both SiCombiner and NoCombiner,
we can see that the number of reduce input record for NoCombiner(42989997) are greater than SiCombiner(166275). So we can deduce that use of SiCombiner will reduce the work for reducer because it aggregate the map processed data which decrease the work for reducers which results in less execution time. This is observed from performance table.

3. Was the local aggregation effective in PerMapTally compared to NoCombiner?
Yes, Local aggrefation was effective in PerMapTally, as execution time is reduced compared to nocombiner, Besides this from syslog file we can notice that number of map output records are significalty less in PerMapTally (**40866300**) compared to NoCombiner(**42842400**).

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

PerTaskTally(03 min 47 sec) has noticeable less execution time compared to PerMapTally(04 min 23 sec),this is suggest us to look closely at the map task, In PerTaskTally all the words are combined in the global HashMap for each mapper which reduced the amount of work required for Reducers. While in PerMapTally, each map method has local HashMap and because of this it emit more key,value pairs which results into the more work for reducers, So PerTaskTally is faster and efficient compare to PerMapTally, so this will result in less map output records. PerMapTally outputed **40866300** map records while PerTaskTally 18678 records.

5. Which one is better: SiCombiner or PerTaskTally? Briefly justify your answer.
We can see that PerTaskTally is better than SiCombiner because PerTaskTally took less time than SiCombiner, Besides this when we look at syslog files of both execution we can see that Map output records for PerTaskTally (18678) are very less compare to SiCombier (412253400). we can fit all the words that starts with m,n,o,p,q into memory and it eliminates the possibility of memory overflow, so PerTaskTally is better compared to SiCombiner, because SiCombiner has to read all the data again and then combiner the same words and then write again, this will result in more time intensive process than updating the word into hashmap which is stored in hashmap.

6. Comparing the results for Configurations 1 and 2, do you believe this MapReduce program scales well to larger clusters? Briefly justify your answer.

Yes, From the above performance results of configuration 1 and 2, we can say that this map reduce programs scales well to large clusters, Because the total executation time for large cluster is significantly less than small cluster.So in large cluster it will process more numbers of inputsplits and resulting in less execution time and greater scalability.