**Class Number:** CS6240 [Tuesday Evening Section] – Section 01
**HW Number:** 1
**Name:** Pavitra Srinivasan

## Source Code

The lines that differ between the codes have been highlighted in yellow.

### 1. NoCombiner

```
public class NoCombiner {

  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());
                          // Extract the first letter of the word, converting it
                          // to lowercase for comparison
              char letter = word.toString().toLowerCase().charAt(0);

                          // Check if the starting letter of the word sarts with 'm' or 'M' or 'n' or 'N'
                          // or 'o' or 'O' or 'p' or 'P' or 'q' or 'Q' then write to the local disk
                          // of the worker machine or node
              if(letter>='m' && letter<='q')
```

```
            {
                        context.write(word, one);
            }
    }
  }
}

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);
  }
}

          // A new class called CustomPartitioner is defined and getPartitoner method is
          // overridden. It takes in three arguments: key, value and the number of Partitions
          // Character.getNumericValue() gives the UNICODE value for the alphabet.
          // Subtracting the UNICODE of 'm' from 'm' gives 0, 'n' from 'm' gives 1 and so on.
          // This ensures the key value distribution does not go beyond the available number
          // of partitions.
public static class CustomPartitioner extends Partitioner<Text,IntWritable> {
```

```
            @Override
            public int getPartition(Text key,IntWritable value, int numPartitions)
            {
                        String word = key.toString();
                        char ch = word.toLowerCase().charAt(0);
                        System.out.println("Word: "+ word);
                        System.out.println("First Character: "+ ch);
                        System.out.println("Reducer Number:"+ (Character.getNumericValue(ch)-Character.getNumericValue('m')));
                        System.out.println();
                        return  Character.getNumericValue(ch)-Character.getNumericValue('m');
            }
}

public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
  if (otherArgs.length < 2) {
    System.err.println("Usage: wordcount <in> [<in>...] <out>");
    System.exit(2);
  }

  @SuppressWarnings("deprecation")
        Job job = new Job(conf, "word count");
  job.setJarByClass(NoCombiner.class);
  job.setMapperClass(TokenizerMapper.class);
  // set to false by default
  boolean combinerToggle = false;
  // Disable combiner
  System.out.println("Combiner Status: " +combinerToggle);
  if(combinerToggle == true) {
        System.out.println("Combiner is set");
```

```
            job.setCombinerClass(IntSumReducer.class);
    }
    else {
            System.out.println("Combiner not set");
    }

    job.setReducerClass(IntSumReducer.class);
            // set the CustomPartitioner class
    job.setPartitionerClass(CustomPartitioner.class);
            //set the Number of Reduce tasks to 5
    job.setNumReduceTasks(5);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job,  new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length  - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

## 2. SiCombiner

```
public class SiCombiner {

  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());

                        // Extract the first letter of the word, converting it
                        // to lowercase for comparison
            char letter = word.toString().toLowerCase().charAt(0);

                        // Check if the starting letter of the word sarts with 'm' or 'M' or 'n' or 'N'
                        // or 'o' or 'O' or 'p' or 'P' or 'q' or 'Q' then write to the local disk
                        // of the worker machine or node
            if(letter>='m' && letter<='q')
            {
                        context.write(word, one);
            }
    }
  }
}

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);
  }
}

        // A new class called CustomPartitioner is defined and getPartitoner method is
        // overridden. It takes in three arguments: key, value and the number of Partitions
        // Character.getNumericValue() gives the UNICODE value for the alphabet.
        // Subtracting the UNICODE of 'm' from 'm' gives 0, 'n' from 'm' gives 1 and so on.
        // This ensures the key value distribution does not go beyond the available number
        // of partitions.
public static class CustomPartitioner extends Partitioner<Text,IntWritable> {

          @Override
          public int getPartition(Text key,IntWritable value, int numPartitions)
          {
                    String word = key.toString();
                    char ch = word.toLowerCase().charAt(0);
                    System.out.println("Word: "+ word);
                    System.out.println("First Character: "+ ch);
                    System.out.println("Reducer Number:"+ (Character.getNumericValue(ch)-Character.getNumericValue('m')));
                    System.out.println();
                    return  Character.getNumericValue(ch)-Character.getNumericValue('m');
          }
}

public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
```

```
    if (otherArgs.length < 2) {
      System.err.println("Usage: wordcount <in> [<in>...] <out>");
      System.exit(2);
    }

    @SuppressWarnings("deprecation")
            Job job = new Job(conf, "word count");
    job.setJarByClass(SiCombiner.class);
    job.setMapperClass(TokenizerMapper.class);
            job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
            // set the CustomPartitioner class
    job.setPartitionerClass(CustomPartitioner.class);
            //set the Number of Reduce tasks to 5
    job.setNumReduceTasks(5);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job,  new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

## 3. ParMapTally

```
public class PerMapTally {
```

```
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());

    // Created a hashmap
    HashMap<Text,IntWritable>  tally = new HashMap<Text,IntWritable>();

    while (itr.hasMoreTokens()) {
           word.set(itr.nextToken());

                      // Extract the first letter of the word, converting it
                      // to lowercase for comparison
           char letter = word.toString().toLowerCase().charAt(0);

                      // Check if the starting letter of the word sarts with 'm' or 'M' or 'n' or 'N'
                      // or 'o' or 'O' or 'p' or 'P' or 'q' or 'Q' then write to the local disk
                      // of the worker machine or node
           if(letter>='m' && letter<='q')
           {
                      if(tally.containsKey(word)) {
                                 System.out.println("Word "+word+" already present");
                                 IntWritable count = new IntWritable(tally.get(word).get()+1);
                                 tally.put(word,count);
                      }
                      else {
                        System.out.println("Word "+word+" occured for the first time");
                        tally.put(new Text(word), one);
                  }

            }
      }

      Iterator<Entry<Text, IntWritable>> iterator = tally.entrySet().iterator();
      while(iterator.hasNext()){
            Entry<Text, IntWritable> entry = iterator.next();
            context.write(entry.getKey(), entry.getValue());
      }

    }
  }

  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);
    }
  }
```

```
        // A new class called CustomPartitioner is defined and getPartitoner method is
        // overridden. It takes in three arguments: key, value and the number of Partitions
        // Character.getNumericValue() gives the UNICODE value for the alphabet.
        // Subtracting the UNICODE of 'm' from 'm' gives 0, 'n' from 'm' gives 1 and so on.
        // This ensures the key value distribution does not go beyond the available number
        // of partitions.
public static class CustomPartitioner extends Partitioner<Text,IntWritable> {

         @Override
         public int getPartition(Text key,IntWritable value, int numPartitions)
         {
                 String word = key.toString();
                 char ch = word.toLowerCase().charAt(0);
                 System.out.println("Word: "+ word);
                 System.out.println("First Character: "+ ch);
                 System.out.println("Reducer Number:"+ (Character.getNumericValue(ch)-Character.getNumericValue('m')));
                 System.out.println();
                 return  Character.getNumericValue(ch)-Character.getNumericValue('m');
         }
}

public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
  if (otherArgs.length < 2) {
    System.err.println("Usage: wordcount <in> [<in>...] <out>");
    System.exit(2);
  }

  @SuppressWarnings("deprecation")
```

```
        Job job = new Job(conf, "word count");
    job.setJarByClass(PerMapTally.class);
    job.setMapperClass(TokenizerMapper.class);
    // set to false by default
    boolean combinerToggle = false;
    // Disable combiner
    System.out.println("Combiner Status: " +combinerToggle);
    if(combinerToggle == true) {
        System.out.println("Combiner is set");
        job.setCombinerClass(IntSumReducer.class);
    }
    else {
        System.out.println("Combiner not set");
    }

    job.setReducerClass(IntSumReducer.class);
        // set the CustomPartitioner class
    job.setPartitionerClass(CustomPartitioner.class);
        //set the Number of Reduce tasks to 5
    job.setNumReduceTasks(5);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job,  new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

## 4. PerTaskTally

```
public class PerTaskTally {

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

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

    // Declared a hashmap
    HashMap<Text,IntWritable>  tally;

            // Called once at the beginning of the task
    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
            System.out.println("Inside Setup");
        // Initialized a hashmap
        tally = new HashMap<Text,IntWritable>();
    }

    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());

                        // Extract the first letter of the word, converting it
                        // to lowercase for comparison
              char letter = word.toString().toLowerCase().charAt(0);
```

```
                    // Check if the starting letter of the word sarts with 'm' or 'M' or 'n' or 'N'
                    // or 'o' or 'O' or 'p' or 'P' or 'q' or 'Q' then write to the local disk
                    // of the worker machine or node
          if(letter>='m' && letter<='q')
          {
                    if(tally.containsKey(word)) {
                              System.out.println("Word "+word+" already present");
                              IntWritable count = new IntWritable(tally.get(word).get()+1);
                              tally.put(word,count);
                    }
                    else {
                              System.out.println("Word "+word+" occured for the first time");
                              tally.put(new Text(word), one);
                    }

          }
    }


  }

        // Called once at the end of the task.
  @Override
  protected void cleanup(Context context) throws IOException, InterruptedException {
        System.out.println("Inside Cleanup");
     Iterator<Entry<Text, IntWritable>> iterator = tally.entrySet().iterator();
     while(iterator.hasNext()){
          Entry<Text, IntWritable> entry = iterator.next();
          context.write(entry.getKey(), entry.getValue());
     }
  }

}

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);
  }
}

          // A new class called CustomPartitioner is defined and getPartitoner method is
          // overridden. It takes in three arguments: key, value and the number of Partitions
          // Character.getNumericValue() gives the UNICODE value for the alphabet.
          // Subtracting the UNICODE of 'm' from 'm' gives 0, 'n' from 'm' gives 1 and so on.
          // This ensures the key value distribution does not go beyond the available number
          // of partitions.
public static class CustomPartitioner extends Partitioner<Text,IntWritable> {

            @Override
            public int getPartition(Text key,IntWritable value, int numPartitions)
            {
```

```
                        String word = key.toString();
                        char ch = word.toLowerCase().charAt(0);
                        System.out.println("Word: "+ word);
                        System.out.println("First Character: "+ ch);
                        System.out.println("Reducer Number:"+ (Character.getNumericValue(ch)-Character.getNumericValue('m')));
                        System.out.println();
                        return  Character.getNumericValue(ch)-Character.getNumericValue('m');
            }
}

public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
  if (otherArgs.length < 2) {
    System.err.println("Usage: wordcount <in> [<in>...] <out>");
    System.exit(2);
  }

  @SuppressWarnings("deprecation")
          Job job = new Job(conf, "word count");
  job.setJarByClass(PerTaskTally.class);
  job.setMapperClass(TokenizerMapper.class);
  // set to false by default
  boolean combinerToggle = false;
  // Disable combiner
  System.out.println("Combiner Status: " +combinerToggle);
  if(combinerToggle == true) {
          System.out.println("Combiner is set");
          job.setCombinerClass(IntSumReducer.class);
  }
  else {
        System.out.println("Combiner not set");
    }

    job.setReducerClass(IntSumReducer.class);
        // set the CustomPartitioner class
    job.setPartitionerClass(CustomPartitioner.class);
        //set the Number of Reduce tasks to 5
    job.setNumReduceTasks(5);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job,  new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

## Explanation for Key and Value

I used the Eclipse Debugging option in order to find out the following:

1. The input “key” to the map function is the offset which means it is the beginning of the line from the beginning of the file of every line in the document.
2. The input “value” is the sentence or the group of words in every line.

I referred the following documentation for answering about the Text Type.

https://hadoop.apache.org/docs/r2.6.0/api/org/apache/hadoop/io/Text.html

3. Hadoop Specific Datatypes provide operational efficiency suited for massive parallel operations and fast read write operations. All the datatypes are based out of Java datatypes e.g. Text for String, IntWritable for int.

Text Type stores the text using standard UTF8 encoding. It provides methods to serialize, deserialize, and compare texts at byte level. In addition, it provides methods for string traversal without converting the byte array to a string.

## Performance Comparison

Config1: 6 small machines (1 master and 5 workers)

Config2:11 small machines (1 master and 10 workers)

Total running time of each program execution.

(Note: I ran all my codes on AWS by making runnable JARS by including all external JARS which resulted in low performance when running on AWS. Hence the time for the job run is more than what other students are getting. Also I ran on m1.medium machine. I put a Piazza post from which I figured out the problem Post# 121. However my code works perfectly fine and generates correct output files. I also spoke with the TA about this and they said that this is fine. Also informed the Professor about the same. Just letting you know as well.)

| Type | Time from Syslog (in sec) | Time from Controller (in sec) |
|---|---|---|
| NoCombiner Run1 Config1 | 589 | 652 |
| NoCombiner Run2 Config1 | 578 | 636 |
| NoCombiner Run1 Config2 | 315 | 388 |
| NoCombiner Run2 Config2 | 362 | 418 |
| SiCombiner Run1 Config1 | 549 | 602 |
| SiCombiner Run2 Config1 | 551 | 608 |

|  |  |  |
|---|---|---|
| SiCombiner Run1 Config2 | 293 | 340 |
| SiCombiner Run2 Config2 | 311 | 346 |
| PerMapTally Run1 Config1 | 629 | 680 |
| PerMapTally Run2 Config1 | 695 | 754 |
| PerMapTally Run1 Config2 | 383 | 430 |
| PerMapTally Run2 Config2 | 429 | 474 |
| PerTaskTally Run1 Config1 | 297 | 350 |
| PerTaskTally Run2 Config1 | 279 | 330 |
| PerTaskTally Run1 Config2 | 154 | 196 |
| PerTaskTally Run2 Config2 | 155 | 202 |

• Do you believe the combiner was called at all in program SiCombiner?

⇨ Yes the combiner was called in SiCombiner program as evident from the following from the syslog file:

⇨

⇨ Combine input records=42842400

⇨ Combine output records=18678

⇨ This shows that combiner was called

• What difference did the use of a combiner make in SiCombiner compared to NoCombiner?

⇨ In SiCombiner, the combiner is called hence combine input and output records show the following value.

⇨

⇨ Combine input records=42842400

⇨ Combine output records=18678

⇨ Reduce input records=18678

⇨

⇨ So the number of records for the reduce input is lower. Hence data transfer from Mapper to Reducer was less.

⇨

⇨ In NoCombiner, the combiner is not called hence the combine input and output are 0 as shown below.

⇨

⇨ Combine input records=0

- ⇨ Combine output records=0
- ⇨ Reduce input records=42842400
- ⇨
- ⇨ So the number of records for the reduce input is much higher. Hence data transfer from Mapper to Reducer was more limiting the network bandwidth.

• Was the local aggregation effective in PerMapTally compared to NoCombiner?

- ⇨ The PerMapTally was not very effective since it only aggregated data on a single line basis. Hence it reduced data transfer very little. Percentage of effectiveness is only approximately 5% or less.
- ⇨ PerMapTally output
- ⇨ Map input records=21907700
- ⇨ Map output records=40866300
- ⇨ Physical memory (bytes) snapshot=9672585216
- ⇨ NoCombiner output
- ⇨ Map input records=21907700
- ⇨ Map output records=42842400
- ⇨ Physical memory (bytes) snapshot=9739759616

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

- ⇨ PerMapTally
- ⇨ Total time spent by all maps in occupied slots (ms)=15040152
- ⇨ Total time spent by all reduces in occupied slots (ms)=3922868
- ⇨
- ⇨ Map input records=21907700
- ⇨ Map output records=40866300
- ⇨ Map output bytes=396549900
- ⇨ Map output materialized bytes=27132293
- ⇨ Reduce shuffle bytes=27132293
- ⇨ Reduce input records=40866300

⇨

⇨ PerTaskTally

⇨ Total time spent by all maps in occupied slots (ms)=5911623

⇨ Total time spent by all reduces in occupied slots (ms)=1151840

⇨

⇨ Map input records=21907700

⇨ Map output records=18678

⇨ Map output bytes=229702

⇨ Map output materialized bytes=186855

⇨ Reduce shuffle bytes=186855

⇨ Reduce input records=18678

⇨

⇨ The total time for all maps for PerMapTally is greater than PerTaskTally.

⇨ The reduce input records of the PerTaskTally is significantly lesser than that of the PerMapTally. This is because, the aggregation in PerTaskTally is done at the map task level and the aggregation in PerMapTally is done in the level of map calls.

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

⇨ In SiCombiner, the combiner is not guaranteed to be executed. If not executed, then the network bandwidth would be used more than that of PerTaskTally. PerTaskTally decreased local CPU and disk I/O costs. Both of them, have more or less same running time. However we could notice that the time taken by the all the maps in PerTaskTally is less since the map output records are emitted only once per map task.

⇨ SiCombiner

⇨

⇨ Map output records=42842400

⇨ Map output bytes=412253400

⇨ Combine input records=42842400

⇨ Combine output records=18678

⇨ CPU time spent (ms)=1603140

⇨

⇨ PerTaskTally

⇨

- ⇨ Map output records=18678
- ⇨ Map output bytes=229702
- ⇨ Combine input records=0
- ⇨ Combine output records=0
- ⇨ CPU time spent (ms)=698800

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

- ⇨ Yes the Map Reduce Program scales well. The running time for config 2 jobs are lesser than the running time of config 1 jobs.
- ⇨ The Map phase is scalable which uses all the 10 machines assigned. However the reducer still uses only 5 machines as we have specified fixed numReduceTask to 5. Hence there is no scalability in terms of reducer.