# CS 6240 Home Work 2  Rushil Patel

**Note for Discrimination of code between the programs [ Red ] color is used**
**Note for Discrimination of code for practitioner, startup, cleanup[ Blue] color is used**

## Program 1 : NoCombiner

```
public class WordCount {


  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());

                        // Check if the word is a valid word - we make use of doesStartWith Methods as
used below
                        if (doesStartWithA(word.toString())
                                        || doesStartWithB(word.toString())
                                        || doesStartWithC(word.toString())
                                        || doesStartWithD(word.toString())
                                        || doesStartWithE(word.toString())) {
                                context.write(word, one);
                        }
                }
        }
  }


  // Custom Partitioner which allocates the keys as follows
  public static class CustomePartitioner extends
                Partitioner<Text, IntWritable> {

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

```
                // this is done to avoid performing mod with 0
                if (numReduceTasks == 0)
                        return 0;
                // Return 0 the key starts with letter a or A
                if (doesStartWithA(key.toString())) {
                        return 0 % numReduceTasks;
                }

                // Return 1 the key starts with letter b or B
                if (doesStartWithB(key.toString())) {
                        return 1 % numReduceTasks;
                }

                // Return 2 the key starts with letter c or C
                if (doesStartWithC(key.toString())) {
                        return 2 % numReduceTasks;
                }
                // Return 3 the key starts with letter d or D
                if (doesStartWithD(key.toString())) {
                        return 3 % numReduceTasks;
                } else {
                        // Return 4 the key starts with letter e or E
                        return 4 % numReduceTasks;
                }

        }
}

//Reducer
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);
        }
}
```

```
// Method checks if the given string starts with the letter a or A
public static boolean doesStartWithA(String s) {
	boolean result = false;
	if (s.length() > 0) {
		char first = s.charAt(0);
		if (first == 'a' || first == 'A') {
			result = true;
		}
	}
	return result;
}

// Method checks if the given string starts with the letter b or B
public static boolean doesStartWithB(String s) {
	boolean result = false;
	if (s.length() > 0) {
		char first = s.charAt(0);
		if (first == 'b' || first == 'B') {
			result = true;
		}
	}
	return result;
}

// Method checks if the given string starts with the letter c or C
public static boolean doesStartWithC(String s) {
	boolean result = false;
	if (s.length() > 0) {
		char first = s.charAt(0);
		if (first == 'c' || first == 'C') {
			result = true;
		}
	}
	return result;
}

// Method checks if the given string starts with the letter d or D
public static boolean doesStartWithD(String s) {
	boolean result = false;
	if (s.length() > 0) {
		char first = s.charAt(0);
		if (first == 'd' || first == 'D') {
			result = true;
		}
	}
	return result;
```

```
    }

    // Method checks if the given string starts with the letter e or E
    public static boolean doesStartWithE(String s) {
            boolean result = false;
            if (s.length() > 0) {
                    char first = s.charAt(0);
                    if (first == 'e' || first == 'E') {
                            result = true;
                    }
            }
            return result;
    }

    //Main Method
    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> <out>");
                    System.exit(2);
            }
            Job job = new Job(conf, "word count");
            job.setJarByClass(WordCount.class);
            job.setPartitionerClass(CustomePartitioner.class);
            job.setMapperClass(TokenizerMapper.class);
            // In the following comment the combiner is disabled
            // job.setCombinerClass(IntSumReducer.class);
            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);
    }
}
```

## Program 2: SiCombiner

```
public class WordCount {

  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());

                        //Check if the word is a valid word
                        if (doesStartWithA(word.toString())
                                        || doesStartWithB(word.toString())
                                        || doesStartWithC(word.toString())
                                        || doesStartWithD(word.toString())
                                        || doesStartWithE(word.toString())) {
                                context.write(word, one);
                        }
                }
        }
}

  public static class CustomePartitioner extends
                Partitioner<Text, IntWritable> {

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

                // this is done to avoid performing mod with 0
                if (numReduceTasks == 0)
                        return 0;
                // Assign word starting with a or A to Task 0
                if (doesStartWithA(key.toString())) {
                        return 0 % numReduceTasks;
                }
                // Assign word starting with b or B to Task 1
                if (doesStartWithB(key.toString())) {
                        return 1 % numReduceTasks;
```

```
                }
                // Assign word starting with c or C to Task 2
                if (doesStartWithC(key.toString())) {
                        return 2 % numReduceTasks;
                }
                // Assign word starting with d or D to Task 3
                if (doesStartWithD(key.toString())) {
                        return 3 % numReduceTasks;
                } else {
                        // Assign word starting with e or E to Task 4
                        return 4 % numReduceTasks;
                }

        }
}

//Reducer
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);
        }
}

//Method checks if the given string starts with A or a
public static boolean doesStartWithA(String s) {
        boolean result = false;
        if (s.length() > 0) {
                char first = s.charAt(0);
                if (first == 'a' || first == 'A') {
                        result = true;
                }
        }
        return result;
}
//Method checks if the given string starts with B or b
public static boolean doesStartWithB(String s) {
        boolean result = false;
```

```
        if (s.length() > 0) {
                char first = s.charAt(0);
                if (first == 'b' || first == 'B') {
                        result = true;
                }
        }
        return result;
}

//Method checks if the given string starts with C or c
public static boolean doesStartWithC(String s) {
        boolean result = false;
        if (s.length() > 0) {
                char first = s.charAt(0);
                if (first == 'c' || first == 'C') {
                        result = true;
                }
        }
        return result;
}

//Method checks if the given string starts with D or d
public static boolean doesStartWithD(String s) {
        boolean result = false;
        if (s.length() > 0) {
                char first = s.charAt(0);
                if (first == 'd' || first == 'D') {
                        result = true;
                }
        }
        return result;
}

//Method checks if the given string starts with E or e
public static boolean doesStartWithE(String s) {
        boolean result = false;
        if (s.length() > 0) {
                char first = s.charAt(0);
                if (first == 'e' || first == 'E') {
                        result = true;
                }
        }
        return result;
}

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> <out>");
                System.exit(2);
        }
        Job job = new Job(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setPartitionerClass(CustomePartitioner.class);
        job.setMapperClass(TokenizerMapper.class);
        // In the following statement the combiner is set
        job.setCombinerClass(IntSumReducer.class);
        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);
  }
}
```

## Program 3: PerMapTally

```
public class WordCount {

  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());

                // Local HashMap to aggregate count of words with
                // O(1) access for inserting and retrieval
                HashMap<String, Integer> map = new HashMap<String, Integer>();
                while (itr.hasMoreTokens()) {
                        word.set(itr.nextToken());
                        if (doesStartWithA(word.toString())
```

```
                                || doesStartWithB(word.toString())
                                || doesStartWithC(word.toString())
                                || doesStartWithD(word.toString())
                                || doesStartWithE(word.toString())) {
                        // If map already contains the word, increment its count
                        if (map.containsKey(word.toString())) {
                                map.put(word.toString(), map.get(word.toString()) + 1);
                        } else {
                                // insert new word with count 1
                                map.put(word.toString(), 1);
                        }
                }
        }

        // After all the words are scanned
        // Emit all the words with their corresponding counts
        for (String word : map.keySet()) {
                context.write(new Text(word), new IntWritable(map.get(word)));
        }
    }
}

// Custom Partitioner which allocates the keys as follows
public static class CustomePartitioner extends
        Partitioner<Text, IntWritable> {

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

        // this is done to avoid performing mod with 0
        if (numReduceTasks == 0)
                return 0;
        // Return 0 the key starts with letter a or A
        if (doesStartWithA(key.toString())) {
                return 0 % numReduceTasks;
        }

        // Return 1 the key starts with letter b or B
        if (doesStartWithB(key.toString())) {
                return 1 % numReduceTasks;
        }

        // Return 2 the key starts with letter c or C
        if (doesStartWithC(key.toString())) {
                return 2 % numReduceTasks;
        }
```

```
                // Return 3 the key starts with letter d or D
                if (doesStartWithD(key.toString())) {
                        return 3 % numReduceTasks;
                } else {
                        // Return 4 the key starts with letter e or E
                        return 4 % numReduceTasks;
                }

        }
}

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);
        }
}

// Method checks if the given string starts with the letter a or A
public static boolean doesStartWithA(String s) {
        boolean result = false;
        if (s.length() > 0) {
                char first = s.charAt(0);
                if (first == 'a' || first == 'A') {
                        result = true;
                }
        }
        return result;
}

// Method checks if the given string starts with the letter b or B
public static boolean doesStartWithB(String s) {
        boolean result = false;
        if (s.length() > 0) {
                char first = s.charAt(0);
                if (first == 'b' || first == 'B') {
                        result = true;
                }
```

```
        }
        return result;
}

// Method checks if the given string starts with the letter c or C
public static boolean doesStartWithC(String s) {
        boolean result = false;
        if (s.length() > 0) {
                char first = s.charAt(0);
                if (first == 'c' || first == 'C') {
                        result = true;
                }
        }
        return result;
}

// Method checks if the given string starts with the letter d or D
public static boolean doesStartWithD(String s) {
        boolean result = false;
        if (s.length() > 0) {
                char first = s.charAt(0);
                if (first == 'd' || first == 'D') {
                        result = true;
                }
        }
        return result;
}

// Method checks if the given string starts with the letter e or E
public static boolean doesStartWithE(String s) {
        boolean result = false;
        if (s.length() > 0) {
                char first = s.charAt(0);
                if (first == 'e' || first == 'E') {
                        result = true;
                }
        }
        return result;
}

// Main Method
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> <out>");
            System.exit(2);
        }
        Job job = new Job(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setPartitionerClass(CustomePartitioner.class);
        job.setMapperClass(TokenizerMapper.class);
        // In the following statement the combiner is not enabled.
        // job.setCombinerClass(IntSumReducer.class);
        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);
  }
}
```

## Program 4: PerTaskTally

```
public class WordCount {

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

        // Global acess to HashMap
        HashMap<String, Integer> map;


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

        //Method used to setup the Map Task called before all mapcalls
        public void setup(Context context) throws IOException,
                    InterruptedException {
            //Initialize the Hash Map
            map = new HashMap<String, Integer>();
        }
```

```
        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 (doesStartWithA(word.toString())
                                        || doesStartWithB(word.toString())
                                        || doesStartWithC(word.toString())
                                        || doesStartWithD(word.toString())
                                        || doesStartWithE(word.toString())) {
                                if (map.containsKey(word.toString())) {
                                        map.put(word.toString(), map.get(word.toString()) + 1);
                                } else {
                                        map.put(word.toString(), 1);
                                }
                        }
                }
        }

        //Called after all map calls are over. Usually does all the clean up
        public void cleanup(Context context) throws IOException,
                        InterruptedException {

                // Emit all the words and its respective counts
                for (String word : map.keySet()) {
                        context.write(new Text(word), new IntWritable(map.get(word)));
                }
                map = null;
        }

}

// Custom Partitioner which allocates the keys as follows
public static class CustomePartitioner extends
                Partitioner<Text, IntWritable> {

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

                // this is done to avoid performing mod with 0
                if (numReduceTasks == 0)
                        return 0;
                // Return 0 the key starts with letter a or A
                if (doesStartWithA(key.toString())) {
                        return 0 % numReduceTasks;
                }
```

```
                // Return 1 the key starts with letter b or B
                if (doesStartWithB(key.toString())) {
                        return 1 % numReduceTasks;
                }

                // Return 2 the key starts with letter c or C
                if (doesStartWithC(key.toString())) {
                        return 2 % numReduceTasks;
                }
                // Return 3 the key starts with letter d or D
                if (doesStartWithD(key.toString())) {
                        return 3 % numReduceTasks;
                } else {
                        // Return 4 the key starts with letter e or E
                        return 4 % numReduceTasks;
                }

        }
}

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);
        }
}

// Method checks if the given string starts with the letter a or A
public static boolean doesStartWithA(String s) {
        boolean result = false;
        if (s.length() > 0) {
                char first = s.charAt(0);
                if (first == 'a' || first == 'A') {
                        result = true;
                }
        }
        return result;
```

```
}

// Method checks if the given string starts with the letter b or B
public static boolean doesStartWithB(String s) {
	boolean result = false;
	if (s.length() > 0) {
		char first = s.charAt(0);
		if (first == 'b' || first == 'B') {
			result = true;
		}
	}
	return result;
}

// Method checks if the given string starts with the letter c or C
public static boolean doesStartWithC(String s) {
	boolean result = false;
	if (s.length() > 0) {
		char first = s.charAt(0);
		if (first == 'c' || first == 'C') {
			result = true;
		}
	}
	return result;
}

// Method checks if the given string starts with the letter d or D
public static boolean doesStartWithD(String s) {
	boolean result = false;
	if (s.length() > 0) {
		char first = s.charAt(0);
		if (first == 'd' || first == 'D') {
			result = true;
		}
	}
	return result;
}

// Method checks if the given string starts with the letter e or E
public static boolean doesStartWithE(String s) {
	boolean result = false;
	if (s.length() > 0) {
		char first = s.charAt(0);
		if (first == 'e' || first == 'E') {
			result = true;
		}
```

```
        }
        return result;
    }

    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> <out>");
            System.exit(2);
        }
        Job job = new Job(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setPartitionerClass(CustomePartitioner.class);
        job.setMapperClass(TokenizerMapper.class);
        // In the following statement the combiner is not enabled
        // job.setCombinerClass(IntSumReducer.class);
        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);
    }
}
```

For all the programs above:
Explanation for the type of input 'key' and input 'value' for Word Counts Map function.
Key: apache.io.LongWritable – stores a long value
Value: apache.io.Text - stores text using standard UTF8 encoding. It provides methods to serialize, deserializ
compare texts at byte level.
The 'key' gives the starting byte of the value and the 'value' represents one single line of the
Input document.

Eg.

If the i/p document is like:

```
 But employers and employees must do their part, as well, as they are
   If you go back to the beginning of this country, the great strength
   It has fallen to every generation since then to preserve that idea,
this issue, studying it. And she and I are going to convene a White
```

When we debug the program we find that the first line is 72 bytes long.

Therefore for the first map function call value of
key = '0' and value" But employers and employees must do their part, as well, as they are"
Now for the next map function call we get the second line as the input.
Therefore in this case the value of
key="72" and value="If you go back to the beginning of this country, the great strength"

**Performance Comparison:**

The following are the runtime of each of program on AWS
Attempt 1

| Program | Start Time | End Time | Run Time |
|---|---|---|---|
| NoCombiner | 23:01:58 | 23:10:36 | 8.38 |
| SiCombiner | 14:49:36 | 14:56:43 | 8.19 |
| PerMapTally | 22:58:40 | 23:08:07 | 9.27 |
| PerTaskTally | 21:53:23 | 22:00:58 | 7.35 |

Attempt 2

| Program | Start Time | End Time | Run Time |
|---|---|---|---|
| NoCombiner | 14:46:57 | 14:55:59 | 10.56 |
| SiCombiner | 15:38:10 | 15:45:36 | 7.46 |
| PerMapTally | 15:07:45 | 15:17:31 | 11.16 |
| PerTaskTally | 15:42:46 | 15:49:01 | 7.47 |

Attempt 3

| Program | Start Time | End Time | Run Time |
|---|---|---|---|
| NoCombiner | 18:35:19 | 18:43:30 | 8.49 |
| SiCombiner | 23:00:10 | 23:07:51 | 8.01 |
| PerMapTally | 16:48:03 | 16:56:47 | 8.50 |
| PerTaskTally | 15:54:58 | 16:02:28 | 7.30 |

**· Do you believe the combiner was called at all in program SiCombiner? · What difference did the use of a combiner make in SiCombiner compared to NoCombiner?**

Yes. The combiner of the SiCombiner was called as by analyzing the syslog of the SiCombiner
We find that

```
Combine input records=67541859
Combine output records=451481
```

This gives us the evidence that the Combiner was called and it computed the above results.
The major difference the combiner of SiCombiner made is that it reduced the number of
Input records for Reducer drastically
NoCombiner : Reduce input records=67125600
SiCombiner: Reduce input records=35222

Thereby reducing the network traffic and the thus the runtime is also reduced.

**· Was the local aggregation effective in PerMapTally compared to NoCombiner?**

No. The local aggregation was not effective in PerMapTally instead it takes more time to run in PerMapTally. Even though the number of Reduce Output Records is less in PerMapTally, a lot of overhead is added in creating a HashMap<> putting values inside the HashMap<> and then after calling the iterator and emitting all the values of the HashMap<>. This actually takes a lot of time as compared to NoCombiner where the words and its count are emitted directly.

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

The Following are the Major Differences

1) PerMapTally : Map output records=64548900
   PerTaskTally: Map output records=35222
2 PerMapTally : Spilled Records=193646700
   PerTaskTally: Spilled Records=70444

PerTaskTally:
Also amount of CPU consumption is also less.
Less time is spent is creating a HashMap<> local to a reduce function call.
Also the amount of data passed to reducer is less therefore less network data transfers are required.
Due to all of above the runtime of PerTaskTally is verless as compared to PerMapTally.

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

I believe PerTaskTally is better than SiCombiner this is because it tries to its maximum to aggregate all the

Relevant data into one record. There by reducing the number of Map Output Records dramatically.
Also this results in less data traffic over the network. Also the runtime of PerTaskTally is less as compared t
SiCombiner.
Therefore the deign Pattern Used by PerTaskTally definitely give it an upper hand for reduced execution tir

**Note:**

**After research I found out that we can limit the number of reduced tasks can be limited by the following command in the the main() funchtion**

```
job.setNumReduceTasks(5);
```

**As I had already run my program on AWS I am mentioning it here.**