CS6240 – PARALLEL DATA PROCESSING IN MAP-REDUCE

Class: CS6240-02
Homework Number: 2
Name: Arpit Mehta

Part 1 - Source Code:
Note: The lines of code that differ between the files are highlighted in blue.

No Combiner:
```
public class WordCountNoCombiner {
	/**
	 * Mapper Class
	 */
	public static class WordCountMapper 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());

				char first_char = word.toString().toLowerCase().charAt(0);

				// Check if "real" word. Lower case first character between 'm' & 'q'
				if ((first_char >= 'm') && (first_char <= 'q')) {
					context.write(word, one);
				}
			}
		}
	}

	/**
	 * Reducer Class
	 */
	public static class WordCountReducer extends
			Reducer<Text, IntWritable, Text, IntWritable> {
		private IntWritable result = new IntWritable();

		public void reduce(Text key, Iterable<IntWritable> values,
				Context context) throws IOException, InterruptedException {
```

```java
			int sum = 0;
			for (IntWritable val : values) {
				sum += val.get();
			}
			result.set(sum);
			context.write(key, result);
		}
	}

	/**
	 * Partitioner Class
	 */
	public static class WordPartitioner extends Partitioner<Text, IntWritable> {

		@Override
		public int getPartition(Text key, IntWritable value, int numPartitions) {
			char first_character = key.toString().toLowerCase().charAt(0);

		// Return the offset of first character of word from 'm' as the partition number.
			return (first_character - 'm');
		}
	}

	/**
	 * main: driver function
	 *
	 * @param args
	 *		list of arguments for the Job - Input file & output directory.
	 *
	 * @throws Exception
	 */
	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: WordCountNoCombiner <in> <out>");
			System.exit(2);
		}

		/**
		 * The Job
		 */
		Job job = new Job(conf, "WordCount with custom partitioner.");

		/**
		 * Set the Jar by finding where a given class came from.
```

```
         */
        job.setJarByClass(WordCountNoCombiner.class);

        /**
         * Set Mapper Class
         */
        job.setMapperClass(WordCountMapper.class);

        /**
         * Set Reducer Class
         */
        job.setReducerClass(WordCountReducer.class);

        /**
         * Note: Combiner disabled
         */
        //job.setCombinerClass(WordCountReducer.class);

        /**
         * Set the number of reduce tasks for the job.
         */
        job.setNumReduceTasks(5);

        /**
         * Set Partitioner Class
         */
        job.setPartitionerClass(WordPartitioner.class);

        /**
         * Set the key class for the job output data.
         */
        job.setOutputKeyClass(Text.class);

        /**
         * Set the value class for job outputs.
         */
        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);
    }
}
```

Si Combiner:

```
public class WordCountSiCombiner {

        /**
         * Mapper Class
         */
        public static class WordCountMapper 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());

                                char first_char = word.toString().toLowerCase().charAt(0);

                                // Check if "real" word.
                                if ((first_char >= 'm') && (first_char <= 'q')) {
                                        context.write(word, one);
                                }
                        }
                }
        }

        /**
         * Reducer Class
         */
        public static class WordCountReducer 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);
                }
        }

        /**
         * Partitioner Class
```

```
 */
public static class WordPartitioner extends Partitioner<Text, IntWritable> {

       @Override
       public int getPartition(Text key, IntWritable value, int numPartitions) {
              char first_character = key.toString().toLowerCase().charAt(0);

       // Return the offset of first character of word from 'm' as the partition number.
              return (first_character - 'm');
       }
}

/**
 * main: driver function
 *
 * @param args
 *                  list of arguments for the Job - Input file & output directory.
 *
 * @throws Exception
 */
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: WordCountSiCombiner <in> <out>");
              System.exit(2);
       }

       /**
        * The Job
        */
       Job job = new Job(conf, "WordCount with custom partitioner.");

       /**
        * Set the Jar by finding where a given class came from.
        */
       job.setJarByClass(WordCountSiCombiner.class);

       /**
        * Set Mapper Class
        */
       job.setMapperClass(WordCountMapper.class);

       /**
        * Set Reducer Class
        */
```

```
            job.setReducerClass(WordCountReducer.class);

            /**
             * Set Combiner Class.
             */
            job.setCombinerClass(WordCountReducer.class);

            /**
             * Set the number of reduce tasks for the job.
             */
            job.setNumReduceTasks(5);

            /**
             * Set Partitioner Class
             */
            job.setPartitionerClass(WordPartitioner.class);

            /**
             * Set the key class for the job output data.
             */
            job.setOutputKeyClass(Text.class);

            /**
             * Set the value class for job outputs.
             */
            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);
      }
}
```

Per Map Tally:

```
public class WordCountPerMapTally {

      /**
       * Mapper Class
       */
      public static class WordCountMapper extends
                  Mapper<Object, Text, Text, IntWritable> {

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

            public void map(Object key, Text value, Context context)
```

```
				throws IOException, InterruptedException {
			StringTokenizer itr = new StringTokenizer(value.toString());

			// HashMap to store the real words.
			HashMap<Text, IntWritable> realWordMap = new HashMap<Text, IntWritable>();

			while (itr.hasMoreTokens()) {
				String token = itr.nextToken();
				char first_char = token.toLowerCase().charAt(0);

				// Check if "real" word.
				if ((first_char >= 'm') && (first_char <= 'q')) {
					Text word = new Text(token);
					if(!realWordMap.containsKey(word)) {
						realWordMap.put(word, one);
					}
					else {
						int oldValue = realWordMap.get(word).get();
						IntWritable newValue = new IntWritable(oldValue+1);
						realWordMap.put(word, newValue);
					}
				}
			}

			for(Text realWord : realWordMap.keySet()) {
				context.write(realWord, realWordMap.get(realWord));
			}
		}
	}

	/**
	 * Reducer Class
	 */
	public static class WordCountReducer 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);
		}
	}
```

```java
/**
 * Partitioner Class
 */
public static class WordPartitioner extends Partitioner<Text, IntWritable> {

       @Override
       public int getPartition(Text key, IntWritable value, int numPartitions) {
              char first_character = key.toString().toLowerCase().charAt(0);

       // Return the offset of first character of word from 'm' as the partition number.
              return (first_character - 'm');
       }
}

/**
 * main: driver function
 *
 * @param args
 *                    list of arguments for the Job - Input file & output directory.
 *
 * @throws Exception
 */
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: WordCountSiCombiner <in> <out>");
              System.exit(2);
       }

       /**
        * The Job
        */
       Job job = new Job(conf, "WordCount with custom partitioner.");

       /**
        * Set the Jar by finding where a given class came from.
        */
       job.setJarByClass(WordCountPerMapTally.class);

       /**
        * Set Mapper Class
        */
       job.setMapperClass(WordCountMapper.class);

       /**
```

```
         * Set Reducer Class
         */
        job.setReducerClass(WordCountReducer.class);

        /**
         * Note: Combiner disabled.
         */
        //job.setCombinerClass(WordCountReducer.class);

        /**
         * Set the number of reduce tasks for the job.
         */
        job.setNumReduceTasks(5);

        /**
         * Set Partitioner Class
         */
        job.setPartitionerClass(WordPartitioner.class);

        /**
         * Set the key class for the job output data.
         */
        job.setOutputKeyClass(Text.class);

        /**
         * Set the value class for job outputs.
         */
        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);
    }
}
```

<u>Per Task Tally:</u>

```
public class WordCountPerTaskTally {

    /**
     * Mapper Class
     */
    public static class WordCountMapper extends
            Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private static HashMap<Text, IntWritable> realWordMap;

        @Override
        protected void setup(Context context) throws IOException,
                InterruptedException {
            super.setup(context);

            // Initialize the HashMap of realWords.
            realWordMap = 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()) {
                String token = itr.nextToken();
                char first_char = token.toLowerCase().charAt(0);

                // Check if "real" word.
                if ((first_char >= 'm') && (first_char <= 'q')) {
                    Text word = new Text(token);
                    if (!realWordMap.containsKey(word)) {
                        realWordMap.put(word, one);
                    } else {
                        int oldValue = realWordMap.get(word).get();
                        IntWritable newValue = new IntWritable(oldValue+1);
                        realWordMap.put(word, newValue);
                    }
                }
            }
        }

        @Override
        protected void cleanup(Context context) throws IOException,
                InterruptedException {
            // Emit & Remove key from HashMap
            Iterator<Entry<Text, IntWritable>> it = realWordMap.entrySet().iterator();

            while(it.hasNext()) {
                Text realWord = it.next().getKey();
                context.write(realWord, realWordMap.get(realWord));

                //  Note:  it.remove()  does  not  cause  the  concurrent  exception  in  the
HashMap.
                it.remove();
            }
```

```
                super.cleanup(context);
        }
}

/**
 * Reducer Class
 */
public static class WordCountReducer 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);
        }
}

/**
 * Partitioner Class
 */
public static class WordPartitioner extends Partitioner<Text, IntWritable> {

        @Override
        public int getPartition(Text key, IntWritable value, int numPartitions) {
                char first_character = key.toString().toLowerCase().charAt(0);

        // Return the offset of first character of word from 'm' as the partition number.
                return (first_character - 'm');
        }
}

/**
 * main: driver function
 *
 * @param args
 *             list of arguments for the Job - Input file & output directory.
 *
 * @throws Exception
 */
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: WordCountSiCombiner <in> <out>");
       System.exit(2);
}

/**
 * The Job
 */
Job job = new Job(conf, "WordCount with custom partitioner.");

/**
 * Set the Jar by finding where a given class came from.
 */
job.setJarByClass(WordCountPerTaskTally.class);

/**
 * Set Mapper Class
 */
job.setMapperClass(WordCountMapper.class);

/**
 * Set Reducer Class
 */
job.setReducerClass(WordCountReducer.class);

/**
 * Note: Combiner disabled.
 */
// job.setCombinerClass(WordCountReducer.class);

/**
 * Set the number of reduce tasks for the job.
 */
job.setNumReduceTasks(5);

/**
 * Set Partitioner Class
 */
job.setPartitionerClass(WordPartitioner.class);

/**
 * Set the key class for the job output data.
 */
job.setOutputKeyClass(Text.class);

/**
 * Set the value class for job outputs.
```

```
             */
            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);
        }
}
```

Part 2 - Performance Comparison:

The criteria on which I have calculated the running time is by subtracting the time-stamp of the message ‘Running job: job_20141003XXXX_XX’ from the time-stamp of the message ‘Job Complete: job_20141003XXXX_XX’ in the syslog file.

First Run:

Word Count Program	6 Instances (1 Master + 6 Workers)	11 Instances (1 Master + 10 Workers)
No Combiner	4 min 56 sec	3 min 46 sec
Si Combiner	4 min 15 sec	3 min 6 sec
Per Map Tally	4 min 28 sec	3 min 6 sec
Per Task Tally	2 min 31 sec	2 min 9 sec

Second Run:

Word Count Program	6 Instances (1 Master + 6 Workers)	11 Instances (1 Master + 10 Workers)
No Combiner	5 mins 0 sec	3 min 47 sec
Si Combiner	3 min 58 sec	3 min 11 sec
Per Map Tally	4 min 4 sec	3 min 1 sec
Per Task Tally	2 min 39 sec	2 min 4 sec

For answering the questions, below are the stats from the syslog of first run of all programs on 11 instances.

Stats	NoCombiner	SiCombiner	PerMapTally	PerTaskTally
Map Input Records	21907700	21907700	21907700	21907700
Map Output Bytes	412253400	412253400	396549900	229702
Combine Input Records	0	42989997	0	0
Reduce Input Records	42842400	18678	40866300	18678
Combine Output Records	0	166275	0	0
Map Output Records	42842400	42842400	40866300	18678
Reduce Input Groups	849	849	849	849

Q: Do you believe the combiner was called at all in program SiCombiner?
A: Yes. As per the stats above, the combiner processed 42989997 input records and produced 166275 output records.

Q: What difference did the use of a combiner make in SiCombiner compared to NoCombiner?
A: Because of the use of combiner, the number of input records to the reducer reduced significantly from 42842400 to just 18678. As a consequence, this led to faster processing of input records and hence lesser running time.

Q: Was the local aggregation effective in PerMapTally compared to NoCombiner?
A: Yes. This is because the number of Map Output Bytes and Map Output Records are lesser in PerMapTally compared to that of NoCombiner. This leads to faster processing of records.

Q: What differences do you see between PerMapTally and PerTaskTally? Try to explain the reasons.
A: The difference between PerMapTally and PerTaskTally is that the number of Reduce Input Records is way less in PerTaskTally than compared to PerMapTally.
The reason for this is PerMapTally does the local aggregation of words for each line. Mostly, the words in a line are unique, the aggregation is not effective.
In PerTaskTally, the aggregation happens per Map task, i.e, the aggregation happens on a chunk of text which is more effective.

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

A: PerTaskTally is better than SiCombiner. This conclusion is based on comparing the number of Map Output Bytes of SiCombiner (412253400) and PerTaskTally(229702). In SiCombiner, it is not guaranteed when the combiner would be used for combining the Map Output Records. In PerTaskTally, the combining happens in the Mapper. This is more effective.

Q: Comparing the results for Configurations 1 and 2, do you believe this MapReduce program scales well to larger clusters? Briefly justify your answer.
A: The program doesn't scale well to large clusters. Ideally, for the same size of data, the running time should ideally be half for 10 worker machines than that of 5 worker machines. The reason for this is that the number of reduce tasks is constrained to 5.
With the given hw1.txt (size: 1.5GB), the number of map tasks that would be launched is 24. In the given configurations, as the number of worker machines in the cluster increases, more number of parallel tasks would run, hence, the map phase would finish faster. The number of reduce tasks will always be 5 as specified in the driver code. Now with the increase in the size of data (possibly in hundred's of GB), the program scales well again to a certain number of machines and then the runtime remains constant after that.