HomeWork 2

CS 6240 – Brinal Pereira

Source Code:

SiCombiner.java

```
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();
     private String tempword;
    /*The key is the offset to the line in the file and value is line itself in the document*/
     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());
         tempword = word.toString();
         tempword = tempword.toLowerCase();
         /*Change to filter out real words. The map now emits words starting with m,n,o,p,q*/
         if(tempword.startsWith("p") || tempword.startsWith("q") ||
                 tempword.startsWith("m") || tempword.startsWith("n") || tempword.startsWith("o"))
         {
            context.write(word, one);
         }
       }
     }
    }
```

```
/*
Implemented a custom partitioner to determine which intermediate keys are assigned to which reducer task.
In this scenario we assign words starting with m to M to Reducer 1.
Taking mod helps avoid the divide by zero error when lesser no. of reducers are available.
*/
public static class WCPartitioner extends Partitioner<Text, IntWritable>{

    @Override
    public int getPartition(Text key, IntWritable value, int numPartitions) {
        String tempWord = key.toString();
        char letter = tempWord.toLowerCase().charAt(0);
        int partitionNumber = 0;
        switch(letter){
            case 'm': partitionNumber = 0 % numPartitions; break;
            case 'n': partitionNumber = 1 % numPartitions; break;
            case 'o': partitionNumber = 2 % numPartitions; break;
            case 'p': partitionNumber = 3 % numPartitions; break;
            case 'q': partitionNumber = 4 % numPartitions; break;
        }
        return partitionNumber;
    }
}
```

```
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);
 }
}
```

```
    public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();
     Job job = Job.getInstance(conf, "word count");
     job.setJarByClass(SiCombiner.class);
     /* Set the number of reduce tasks*/
     job.setNumReduceTasks(5);
     job.setMapperClass(TokenizerMapper.class);
     /* Set the partitioner class.*/
     job.setPartitionerClass(WCPartitioner.class);
     job.setCombinerClass(IntSumReducer.class);
     job.setReducerClass(IntSumReducer.class);
     job.setOutputKeyClass(Text.class);
     job.setOutputValueClass(IntWritable.class);
     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));
     System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

WordCountNoCombiner.java

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

     private final static IntWritable one = new IntWritable(1);
     private Text word = new Text();
     private String tempword;
    /*The key is the offset to the line in the file and value is line itself in the document*/
     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());
         tempword = word.toString();
         tempword = tempword.toLowerCase();
         /*Change to filter out real words. The map now emits words starting with m,n,o,p,q*/
         if(tempword.startsWith("p") || tempword.startsWith("q") ||
                 tempword.startsWith("m") || tempword.startsWith("n") || tempword.startsWith("o"))
         {
            context.write(word, one);
         }
       }
     }
    }
```

```java
/*
Implemented a custom partitioner to determine which intermediate keys are assigned to which reducer task.
In this scenario we assign words starting with m to M to Reducer 0.
Taking mod helps avoid the divide by zero error when lesser no. of reducers are available.
*/
public static class WCPartitioner extends Partitioner<Text, IntWritable>{

    @Override
    public int getPartition(Text key, IntWritable value, int numPartitions) {
        String tempWord = key.toString();
        char letter = tempWord.toLowerCase().charAt(0);
        int partitionNumber = 0;
        switch(letter){
            case 'm': partitionNumber = 0 % numPartitions; break;
            case 'n': partitionNumber = 1 % numPartitions; break;
            case 'o': partitionNumber = 2 % numPartitions; break;
            case 'p': partitionNumber = 3 % numPartitions; break;
            case 'q': partitionNumber = 4 % numPartitions; break;
        }
        return partitionNumber;
    }
}
```

```java
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);
 }
}
```

```
    public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();
     Job job = Job.getInstance(conf, "word count");
     job.setJarByClass(WordCountNoCombiner.class);
     job.setNumReduceTasks(5);
     job.setMapperClass(TokenizerMapper.class);
     job.setPartitionerClass(WCPartitioner.class);
     /* We do not set the combiner*/
     //job.setCombinerClass(IntSumReducer.class);
     job.setReducerClass(IntSumReducer.class);
     job.setOutputKeyClass(Text.class);
     job.setOutputValueClass(IntWritable.class);
     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));
     System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

WordCountPerMapTally.java

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

     private final static IntWritable one = new IntWritable(1);
     private Text word = new Text();
     private String tempword;
    /*The key is the offset to the line in the file and value is line itself in the document*/
     public void map(Object key, Text value, Context context
                     ) throws IOException, InterruptedException {
       StringTokenizer itr = new StringTokenizer(value.toString());
       /*In this approach we define a HashMap to store the count of each word instead
       of emitting (word, one) each time*/
       HashMap<String, Integer> dict = new HashMap<String, Integer>();
       while (itr.hasMoreTokens()) {
         word.set(itr.nextToken());
         tempword = word.toString();
         tempword = tempword.toLowerCase();
         if(tempword.startsWith("p") || tempword.startsWith("q") ||
                 tempword.startsWith("m") || tempword.startsWith("n") || tempword.startsWith("o"))
         {
            if(!dict.containsKey(tempword))
                dict.put(tempword, 1);
            else
                dict.put(tempword, dict.get(tempword) + 1);
         }
       }
```

```
      for(String tempword : dict.keySet())
      {
         /*Instead of emitting one we emit the word with the total count
         */
         Text wordToWrite = new Text(tempword);
         context.write(wordToWrite, new IntWritable(dict.get(tempword)));
      }
    }
   }
   /*
   Implemented a custom partitioner to determine which intermediate keys are assigned to which reducer task.
   In this scenario we assign words starting with m to M to Reducer 0.
   Taking mod helps avoid the divide by zero error when lesser no. of reducers are available.
   */
   public static class WCPartitioner extends Partitioner<Text, IntWritable>{

      @Override
      public int getPartition(Text key, IntWritable value, int numPartitions) {
         String tempWord = key.toString();
         char letter = tempWord.toLowerCase().charAt(0);
         int partitionNumber = 0;
         switch(letter){
            case 'm': partitionNumber = 0 % numPartitions; break;
            case 'n': partitionNumber = 1 % numPartitions; break;
            case 'o': partitionNumber = 2 % numPartitions; break;
            case 'p': partitionNumber = 3 % numPartitions; break;
            case 'q': partitionNumber = 4 % numPartitions; break;
         }
         return partitionNumber;
      }
   }

   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);
    }
   }
```

```
    public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();
     Job job = Job.getInstance(conf, "word count");
     job.setJarByClass(WordCountPerMapTally.class);
     job.setNumReduceTasks(5);
     job.setMapperClass(TokenizerMapper.class);
     job.setPartitionerClass(WCPartitioner.class);
     /* We do not set the combiner*/
     //job.setCombinerClass(IntSumReducer.class);
     job.setReducerClass(IntSumReducer.class);
     job.setOutputKeyClass(Text.class);
     job.setOutputValueClass(IntWritable.class);
     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));
     System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

WordCountPerTaskTally.java

```
public class WordCountPerTaskTally {
    public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{
    /*In this approach we define a HashMap to store the count of each word at the Task level*/
     HashMap<String, Integer> dict;
     private final static IntWritable one = new IntWritable(1);
     private Text word = new Text();
     private String tempword;

     /*Called once at the start of the task.
     Used for initialization and clean up at task level.*/
     protected void setup(Context context) throws IOException , InterruptedException
     {
         dict = new HashMap<String, Integer>();
     }
     /*The key is the offset to the line in the file and value is line itself in the document*/
     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());
         tempword = word.toString();
         tempword = tempword.toLowerCase();
         if(tempword.startsWith("p") || tempword.startsWith("q") ||
                 tempword.startsWith("m") || tempword.startsWith("n") || tempword.startsWith("o"))
         {
            if(!dict.containsKey(tempword))
                dict.put(tempword, 1);
            else
                dict.put(tempword, dict.get(tempword) + 1);
         }
       }
```

```
    }
    /*Called once at the end of the task.*/
    public void  cleanup(Context context)throws IOException, InterruptedException
    {
        for(String tempword : dict.keySet())
          {
             Text wordToWrite = new Text(tempword);
             context.write(wordToWrite, new IntWritable(dict.get(tempword)));
          }
    }
  }
```

```
/*
Implemented a custom partitioner to determine which intermediate keys are assigned to which reducer task.
In this scenario we assign words starting with m to M to Reducer 0.
Taking mod helps avoid the divide by zero error when lesser no. of reducers are available.
*/
public static class WCPartitioner extends Partitioner<Text, IntWritable>{

    @Override
    public int getPartition(Text key, IntWritable value, int numPartitions) {
        String tempWord = key.toString();
        char letter = tempWord.toLowerCase().charAt(0);
        int partitionNumber = 0;
        switch(letter){
            case 'm': partitionNumber = 0 % numPartitions; break;
            case 'n': partitionNumber = 1 % numPartitions; break;
            case 'o': partitionNumber = 2 % numPartitions; break;
            case 'p': partitionNumber = 3 % numPartitions; break;
            case 'q': partitionNumber = 4 % numPartitions; break;
        }
        return partitionNumber;
    }
}
```

```
  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);
    }
  }
```

```
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCountPerTaskTally.class);
    job.setNumReduceTasks(5);
    job.setMapperClass(TokenizerMapper.class);
    job.setPartitionerClass(WCPartitioner.class);
    /* We do not set the combiner*/
    //job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Performance Comparison:

Program Name	Total Running Time 1	Total Running Time 2
SiCombiner (Config 1)	3 minutes 27 seconds	3 minutes 36 seconds
No Combiner (Config 1)	3 minutes 37 seconds	3 minutes 56 seconds
Per Map Tally (Config 1)	3 minutes 41 seconds	4 minutes 17 seconds
Per Task Tally (Config 1)	3 minutes 03 seconds	3 minutes 05 seconds
SiCombiner (Config 2)	2 minutes 11 seconds	2 minutes 11 seconds
No Combiner (Config 2)	2 minutes 29 seconds	2 minutes 29 seconds
Per Map Tally (Config 2)	2 minutes 34 seconds	2 minutes 47 seconds
Per Task Tally (Config 2)	2 minutes 31 seconds	1 minutes 51 seconds

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

1) **Do you believe the combiner was called at all in program SiCombiner?**
   Yes, the combiner has been called in the SiCombiner. From the log files for SiCombiner we can confirm that
   Combine input records=42842400
   Combine output records=18678
   As opposed to the other configurations where , the log files state that
   Combine input records=0
   Combine output records=0

2) **What difference did the use of a combiner make in SiCombiner compared to NoCombiner?**
   Using a combiner in SiCombiner helped reduce the number of records passed to the Reducer.
   i.e. In SiCombiner,
   Map output records=42842400
   Combine input records=42842400
   Combine output records=18678
   Reduce input records=18678
   In NoCombiner,
   Map output records=42842400
   Combine input records=0
   Combine output records=0
   Reduce input records=42842400

3) **Was the local aggregation effective in PerMapTally compared to NoCombiner?**
The number of records output from the Map reduced substantially in perMapTally due to local aggregation
In NoCombiner,
Map output records=42842400
In PerMapTally,
Map output records=40866300

4) **What differences do you see between PerMapTally and PerTaskTally? Try to explain the reasons.**
There is a strong difference in the number of map output records in PerTaskTally and PerMapTally approach. The output records are reduced from 40866300 to 18678
In PerMapTally,
Map output records=40866300
In PerTaskTally,
Map output records=18678
Also, the time difference between PerTaskTally and PerMapTally is also approximately 1 minute and 12 seconds.
This is because in PerTaskTally we do not emit map output records for each map function but emit the records after each map Task. Hence, PerTaskTally does more aggregation than PerMapTally.

5) **Which one is better: SiCombiner or PerTaskTally? Briefly justify your answer.**
Based on the data in the log files, PerTaskTally is better than SiCombiner because PerTaskTally program runs faster than the SiCombiner. Also, the number of records emitted by the Mapper in SiCombiner is more than the number of records emitted by the mapper in PerTaskTally. Combiner in SiCombiner combines data after it is emitted from the mapper.
Also, with combiners there is no guarantee when and how often it will be executed.

6) **NEW: Comparing the results for Configurations 1 and 2, do you believe this MapReduce program scales well to larger clusters? Briefly justify your answer.**
Taking into consideration the total running time for the two configurations. We observe that running the job on more machines helps fasten the process. Hence increasing number of cores helps in a slight performance scale up.