Assignment 02

UMANG MUKESH MEHTA

FRIDAY 1:35 – 3:15 PM

Report:

The Key represents a line number from the file being read. The input keys are assumed to be identifiers for documents, which are ignored, and the value to be the content of the document. The map() method just uses the value, which in this case of a text file is a line of the file. The output keys are words found with the document, and the output values are the number of times a word appeared in the document.

1. Source Code

Changes in the original WordCount Source Code:

1. The condition to check the counts for the real word.I have developed a RegularExpression(Regex) to determine the counts of the "real" words starting with m,n,o,p,q,M,N,O,P,Q

```
while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        if(Pattern.matches("[m-qM-Q].*$", word.toString()))
        {context.write(word,one);
```

2. A custom Partioner that assigns words that start with a specific alphabet to the given Reduced task.The problem statement says the reduce task 0 should contain words that start either with m or M.I have implemented the same approach. I have done so by taking the CharAt(0) the first position of the value converted toString() and then checking against the accepted values.

```
public static class WordPartitioner extends Partitioner <Text, IntWritable> {
   @Override
   public int getPartition(Text key, IntWritable value, int numReduceTasks) {

      String val = key.toString();
      if(val.charAt(0)=='m'||val.charAt(0)=='M')
         return 0;

      if(val.charAt(0)=='n'||val.charAt(0)=='N')
         return 1;
```

```
        if(val.charAt(0)=='o'||val.charAt(0)=='O')
            return 2;

        if(val.charAt(0)=='p'||val.charAt(0)=='P')
            return 3;

        if(val.charAt(0)=='q'||val.charAt(0)=='Q')
            return 4;

        else
            return -1;

        }
}
```

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());
              if(Pattern.matches("[m-qM-Q].*$", word.toString()))
              {context.write(word,one);
                 }
            }
          }
        }

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

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

        String val = key.toString();
        if(val.charAt(0)=='m'||val.charAt(0)=='M')
            return 0;

        if(val.charAt(0)=='n'||val.charAt(0)=='N')
            return 1;

        if(val.charAt(0)=='o'||val.charAt(0)=='O')
            return 2;

        if(val.charAt(0)=='p'||val.charAt(0)=='P')
            return 3;

        if(val.charAt(0)=='q'||val.charAt(0)=='Q')
            return 4;

        else
            return -1;

        }
}
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();
    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.setMapperClass(TokenizerMapper.class);
    job.setPartitionerClass(WordPartitioner.class);
// The only change in the NoCombiner program is we have to disable the Combiner
    // 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());
              if(Pattern.matches("[m-qM-Q].*$", word.toString()))
              {context.write(word,one);
                }
            }
          }
        }

        public static class WordPartitioner extends Partitioner <Text, IntWritable> {
            @Override
            public int getPartition(Text key, IntWritable value, int numReduceTasks) {

                String val = key.toString();
                if(val.charAt(0)=='m'||val.charAt(0)=='M')
```

```
        return 0;

      if(val.charAt(0)=='n'||val.charAt(0)=='N')
        return 1;

      if(val.charAt(0)=='o'||val.charAt(0)=='O')
        return 2;

      if(val.charAt(0)=='p'||val.charAt(0)=='P')
        return 3;

      if(val.charAt(0)=='q'||val.charAt(0)=='Q')
        return 4;

      else
        return -1;

      }
}
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();
    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.setMapperClass(TokenizerMapper.class);
    job.setPartitionerClass(WordPartitioner.class);
  //The only change in the SiCombiner from NoCombiner is that the combiner is 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 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());

// HashMap() Initialization
                        HashMap<String,Integer> hmap= new HashMap<String,Integer>();
                        while (itr.hasMoreTokens()) {
                                word.set(itr.nextToken());
                                String wordToString=word.toString();

// Here the HashMap() DataStructure combines counts inside the Mao function itself.
//It's a Simple modification,typically not very effective as it combines the counts only within a single line of
text.
                                if(Pattern.matches("[m-qM-Q].*$", wordToString))
                                {if(hmap.containsKey(wordToString))
                                        hmap.put(wordToString, hmap.get(wordToString)+1);
                                else
                                        hmap.put(wordToString, 1);
                                }
                        }
                        Iterator it = hmap.entrySet().iterator();
                        while (it.hasNext()) {
                                Map.Entry<String,Integer> keyValuePair =
                                (Map.Entry<String,Integer>)it.next();
                                IntWritable valueaccepted = new IntWritable(keyValuePair.getValue());
                                Text texttoremove= new Text(keyValuePair.getKey());
                                context.write(texttoremove, valueaccepted);
                                  it.remove(); // avoids a ConcurrentModificationException

                        }


        }

}

  public static class WordPartitioner extends Partitioner <Text, IntWritable> {
     @Override
     public int getPartition(Text key, IntWritable value, int numReduceTasks) {

        String val = key.toString();
        if(val.charAt(0)=='m'||val.charAt(0)=='M')
            return 0;

        if(val.charAt(0)=='n'||val.charAt(0)=='N')
            return 1;

        if(val.charAt(0)=='o'||val.charAt(0)=='O')
            return 2;
```

```
            if(val.charAt(0)=='p'||val.charAt(0)=='P')
                return 3;

            if(val.charAt(0)=='q'||val.charAt(0)=='Q')
                return 4;

            else
                return -1;

            }
        }

    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();
            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.setMapperClass(TokenizerMapper.class);
            job.setPartitionerClass(WordPartitioner.class);

// Here the combiner is absent as HashMap takes care of merging the counts in the Map function.
            // 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

```
1.  public class WordCount{
2.
3.          public static class TokenizerMapper
4.          extends Mapper<Object, Text, Text, IntWritable>{
5.  //HashMap Initialization
6.
7.                  public HashMap<String,Integer> hmap;
8.
9.                  private final static IntWritable one = new IntWritable(1);
10.                 private Text word = new Text();

//The HashMap() needs to be initialized before first Map function call in the task.
// It gives a NullPointerException() when passed without Arguments.

11.                 public void setup(Context context)
12.                 {
13.                         hmap= new HashMap<String,Integer>();
14.                 }

15.                 public void map(Object key, Text value, Context context
16.                         ) throws IOException, InterruptedException
17.                 {
18.                         StringTokenizer itr = new StringTokenizer(value.toString());
19.                         while (itr.hasMoreTokens()) {
20.                                 word.set(itr.nextToken());
21.                                 String wordToString=word.toString();
22.                                 System.out.println("Words are : " + wordToString);
23.                                 if(Pattern.matches("[m-qM-Q].*$", wordToString))

//Using the Key Value HashMap() pair to check for the matched pattern
//
24.                                 {if(hmap.containsKey(wordToString)) {
25.                                         hmap.put(wordToString, hmap.get(wordToString)+1);
26.                                 }
27.                                 else {
28.                                         hmap.put(wordToString, 1);
29.                                 }
30.                                 }
31.                         }
32.
33.
34.                 }
```

// The cleanup() to emit the final tally of the entire task,the counts in H have to be emitted after the last
//Map call in the last task submitted

```
35.                     public void cleanup(Context context) throws IOException,
InterruptedException{
36.                     Iterator it = hmap.entrySet().iterator();
37.                     while (it.hasNext()) {
38.                             Map.Entry<String,Integer> keyValuePair =
(Map.Entry<String,Integer>)it.next();
39.                             IntWritable valueaccepted = new
IntWritable(keyValuePair.getValue());
40.                             Text texttoremove= new Text(keyValuePair.getKey());
41.                             context.write(texttoremove, valueaccepted);
42.                              it.remove(); // avoids a ConcurrentModificationException}
43.
44.                     }
45.             }
46.             }
47.               public static class WordPartitioner extends Partitioner <Text, IntWritable> {
48.                  @Override
49.                  public int getPartition(Text key, IntWritable value, int numReduceTasks) {
50.
51.                      String val = key.toString();
52.                      if(val.charAt(0)=='m'||val.charAt(0)=='M')
53.                          return 0;
54.
55.                      if(val.charAt(0)=='n'||val.charAt(0)=='N')
56.                          return 1;
57.
58.                      if(val.charAt(0)=='o'||val.charAt(0)=='O')
59.                          return 2;
60.
61.                      if(val.charAt(0)=='p'||val.charAt(0)=='P')
62.                          return 3;
63.
64.                      if(val.charAt(0)=='q'||val.charAt(0)=='Q')
65.                          return 4;
66.
67.                      else
68.                          return -1;
69.
70.                      }
71.               }
72.
73.
74.             public static class IntSumReducer
75.             extends Reducer<Text,IntWritable,Text,IntWritable> {
76.                     private IntWritable result = new IntWritable();
77.
78.                     public void reduce(Text key, Iterable<IntWritable> values,
79.                                     Context context
80.                                     ) throws IOException, InterruptedException {
81.                             int sum = 0;
82.                             for (IntWritable val : values) {
83.                                     sum += val.get();
84.                             }
85.                             result.set(sum);
```

```
86.                         context.write(key, result);
87.                 }
88.         }
89.
90.         public static void main(String[] args) throws Exception {
91.                 Configuration conf = new Configuration();
92.                 String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
93.                 if (otherArgs.length != 2) {
94.                         System.err.println("Usage: wordcount <in> <out>");
95.                         System.exit(2);
96.                 }
97.                 Job job = new Job(conf, "word count");
98.                 job.setJarByClass(WordCount.class);
99.                 job.setMapperClass(TokenizerMapper.class);
100.                        job.setPartitionerClass(WordPartitioner.class);
101.                        // job.setCombinerClass(IntSumReducer.class);
102.                        job.setReducerClass(IntSumReducer.class);
103.                        job.setOutputKeyClass(Text.class);
104.                        job.setOutputValueClass(IntWritable.class);
105.                        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
106.                        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
107.                        System.exit(job.waitForCompletion(true) ? 0 : 1);
108.                }
109.
110.                }
```

2. Performance Comparison:

Programs	Average(in mins)
1. NoCombiner	
Config 1	8mins
Config 2	5mins
2. SiCombiner	
Config 1	6mins
Config 2	4mins
3. PerMapTally	
Config 1	8mins
Config 2	5mins
4. PerTaskTally	
Config 1	24mins
Config 2	20mins

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

- Yes I believe that the combiner was called in the SiCombiner as the total running time reduced significantly with the use of the Combiner.Looking at the syslog file acknowledges the use of Combiner in SiCombiner.
  After the Initialization of the MapReduce Framework

The log for NoCombiner:

2014-10-05 22:52:58,965 INFO org.apache.hadoop.mapred.JobClient (main): Combine input records=0
2014-10-05 22:52:58,965 INFO org.apache.hadoop.mapred.JobClient (main): Combine output records=0

The log for SiCombiner:

2014-10-05 23:14:42,205 INFO org.apache.hadoop.mapred.JobClient (main): Combine input records=42989997
2014-10-05 23:14:42,205 INFO org.apache.hadoop.mapred.JobClient (main): Combine output records=166275

The syslog entry clearly states that the combiner was called in SiCombiner.

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

- The use of Combiner in SiCombiner as against to NoCombiner made a lot of difference in the computation time of the program. The NoCombiner program's Average running time for the Configuration (1 Master and 5 slaves) evidently described the use of Combiner as crucial in decreasing the number of tasks passed on to Reducer. The Average to tal Running time for the execution reduced to a value of 2mins (The time for Computation of the program was checked in the Syslog file and also the AWS UI) which made it clear.
  Also the total memory(Heap Usage is decreased with the use of Combiner)

In case of NoCombiner:

2014-10-05 22:52:58,965 INFO org.apache.hadoop.mapred.JobClient (main):
Total committed heap usage (bytes)=7320567808

In case of SiCombiner:

2014-10-05 23:14:42,205 INFO org.apache.hadoop.mapred.JobClient (main):
Total committed heap usage (bytes)=6373052416

Was the local aggregation effective in PerMapTally compared to NoCombiner?

- Both the PerMapTally and NoCombiner don't have a combiner.The Local aggregation was effective in case of PerMapTally as the Map output records decreased due to local aggregation.

In Case of PerMapTally:

2014-10-05 23:39:25,019 INFO org.apache.hadoop.mapred.JobClient (main): Map output records=40866300

InCase of NoCombiner

2014-10-05 22:52:58,965 INFO org.apache.hadoop.mapred.JobClient (main): Map output records=42842400

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

- **Advantages:**
  The perTaskTally version is an in-Mapper combining design pattern.It is evident from the logs that it preserves the state at the task level,across different Map calls in the same task. It is almost the same pattern used in the Reduced task. The perTaskTally method is better than perMapTally method in the number of bytes read from the file and written.

InCase of PerTaskTally

2014-10-06 00:22:12,122 INFO org.apache.hadoop.mapred.JobClient (main): FILE_BYTES_READ=96901

2014-10-06 00:22:12,122 INFO org.apache.hadoop.mapred.JobClient (main): FILE_BYTES_WRITTEN=1036565

In case of PerMapTally

2014-10-05 23:39:25,016 INFO org.apache.hadoop.mapred.JobClient (main): FILE_BYTES_READ=74284394

2014-10-05 23:39:25,017 INFO org.apache.hadoop.mapred.JobClient (main): ILE_BYTES_WRITTEN=83339817

Also the number of Input records for the both the methods is same but the perTaskTally method with in-mapper combining and state preservation decrease the number of output record to a substantial greater amount.

Incase of perTaskTally

2014-10-06 00:22:12,124 INFO org.apache.hadoop.mapred.JobClient (main): Map input records=21907700

2014-10-06 00:22:12,124 INFO org.apache.hadoop.mapred.JobClient (main): Map output bytes=229702

2014-10-06 00:22:12,124 INFO org.apache.hadoop.mapred.JobClient (main): Map output materialized bytes=187914

2014-10-06 00:22:12,124 INFO org.apache.hadoop.mapred.JobClient (main): Map output records=18678

InCase of PerMapTally

2014-10-05 23:39:25,018 INFO org.apache.hadoop.mapred.JobClient (main): Map input records=21907700

2014-10-05 23:39:25,019 INFO org.apache.hadoop.mapred.JobClient (main): Map output bytes=396549900

2014-10-05 23:39:25,019 INFO org.apache.hadoop.mapred.JobClient (main): Map output materialized bytes=27139031

2014-10-05 23:39:25,019 INFO org.apache.hadoop.mapred.JobClient (main): Map output records=40866300

Disadvantages:

The disadvantages that I encounter looking at the log file is, Firstly the CPU Time spent. The perTaskTally takes a large amount of CPU time as it preserves state as compared to perMapTally.This can be avoided in case of deploying the Wordcount code to many servers to achieve the least CPU time.

InCase of PerTaskTally

2014-10-06 00:22:12,124 INFO org.apache.hadoop.mapred.JobClient (main): CPU time spent (ms)=5223410

In case of PerMapTally

2014-10-05 23:39:25,018 INFO org.apache.hadoop.mapred.JobClient (main): CPU time spent (ms)=1395710

Secondly, the memory for managing the state, to store the hashMap in memory.

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

- In WordCount, Cascading does not support SiCombiner. Combiners are a simple optimization allowing some Reduce functions to run on the Map side of MapReduce. Combiners are very powerful in that they reduce the I/O between the Mappers and Reducers .This example for WordCount answered why send all of your Mapper data to Reducers when you can compute some values on the Map side and combine them in the Reducer? But Combiners are limited to Associative and Commutative functions only, such as "sum" and "max". And the process requires that the values emitted by the Map task must be serialized, sorted (which involves deserialization and comparison), deserialized again, and operated on - after which the results are again serialized and sorted. Combiners trade CPU for gains in I/O.

  PerTaskTally gives us a Cascading approach. It provides a mechanism to perform partial aggregations on the Map side and combine the results on the Reduce side, but trades memory, instead of CPU, for I/O gains by caching values (up to a threshold limit). This bypasses the redundant serialization, deserialization, and sorting. Also, Cascading allows any aggregate function to be implemented - not just Associative and Commutative functions.

Cascading supports a few built-in partial aggregate operations, including AverageBy, CountBy, and SumBy. These are actually SubAssemblies, not Operations, and are subclasses of the AggregateBy SubAssembly.

Also the lack of control over the Combiner process in SiCombiner is a limitation which PerTaskTally averts.

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 it will definitely scale well for large clusters. Using PerTaskTally approach will considerably reduce the number of data tuples to the $1/1000^{th}$ of the database in the Map phase itself.The only disadvantage is it will be very complex to analyze such large scale cluster in MapReduce.