REPORT

Mukul Sharma Class : CS 6240 - 02

Map functions input key and input value :

The input “key” and input “value” for the map function is the key/value pair that map function uses to spawn a map task for this pair.

The “value” is of the “Text” type that stores text using UTF8 encoding, the Text class provides various methods such as serialize, deserialize and text comparison function. For the current data input that is the text document each “value” has one line of the text document.

The “key” is of the “Object” type and it represents the line number of the value in its pair that is for each value/line from input file a key/linenumber is generated.

I looked up Apache hadoop 2.6.1 API for this information and after printing the values for “key and value”.

Performance Comparison

- Configuration 1: 6 small machines (one master and 5 workers) :

Total running time:

Run 1 :

SiCombiner: 251 seconds
NoCombiner: 283 seconds
PerMapTally: 286 seconds
PerTaskTally: 182 seconds

Run 2 :

SiCombiner: 237 seconds
NoCombiner: 276 seconds
PerMapTally: 291 seconds
PerTaskTally: 184 seconds

- Configuration 2: 11 small machines (one master and 10 workers) :

Total running time:

Run 1 :

SiCombiner: 209 seconds
NoCombiner: 224 seconds
PerMapTally: 229 seconds
PerTaskTally: 151 seconds

Run 2:

SiCombiner: 199 seconds
NoCombiner: 209 seconds
PerMapTally: 225 seconds
PerTaskTally: 169 second

SOURCECODE

SiCombiner:

"This file is identical to Word count program provided by hadoop distribution including the additions to partition the data in different parts using partitioner and the criteria provided in the assignment"

```
public class SiCombiner {

   //Partitioner class
   public static class WordCountPartitioner extends Partitioner<Text, IntWritable>{

          @Override
          public int getPartition(Text t, IntWritable value, int numPartitions) {
                String s = t.toString();
                char first = s.charAt(0);
                if(first=='m'||first=='M')
                      return 0;
                if(first=='n'||first=='N')
                      return 1;
                if(first=='o'||first=='O')
                      return 2;
                if(first=='p'||first=='P')
                      return 3;
                if(first=='q'||first=='Q')
                      return 4;
                else
                      return 0;
          }
   }

  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())
            {   String s = itr.nextToken();
               char first = s.charAt(0);
               if(isLegal(first))
               {
                     word.set(s);
                     context.write(word, one);
               }
            }
    }
```

```
  //Function to filter input
  //Returns true if the letter is in range m-q or M-Q
  public boolean isLegal(char first)
  {
        if(first >= 'm' && first <= 'q' || first >='M' && first <= 'Q')
        {
              return true;
        }
        else
              return false;
  }
}

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(SiCombiner.class);
  job.setMapperClass(TokenizerMapper.class);

  job.setCombinerClass(IntSumReducer.class);

  //setting multiple reducer classes
  job.setNumReduceTasks(5);
  job.setPartitionerClass(WordCountPartitioner.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);
  }
}
```

NoCombiner:

"This program is identical to the SiCombiner program along with changes to disable Combiner, If the user passes false as argument during execution of program then the combiner is disabled"

```
public class NoCombiner {

   //Partitioner class
   public static class WordCountPartitioner extends Partitioner<Text, IntWritable>{

         @Override
         public int getPartition(Text t, IntWritable value, int numPartitions) {
               String s = t.toString();
               char first = s.charAt(0);
               if(first=='m'||first=='M')
                     return 0;
               if(first=='n'||first=='N')
                     return 1;
               if(first=='o'||first=='O')
                     return 2;
               if(first=='p'||first=='P')
                     return 3;
               if(first=='q'||first=='Q')
                     return 4;
               else
                     return 0;
         }
   }

  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())
        {   String s = itr.nextToken();
            char first = s.charAt(0);
            if(isLegal(first))
            {
                  word.set(s);
                  context.write(word, one);
            }
         }
   }

   //Function to filter input
   //Returns true if the letter is in range m-q or M-Q
```

```
    public boolean isLegal(char first)
    {
            if(first >= 'm' && first <= 'q' || first >='M' && first <= 'Q')
            {
                    return true;
            }
            else
                    return false;
    }
  }

  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 != 3) {
      System.err.println("Usage: wordcount <in> <out> <Use Combiner? true/false>");
      System.exit(2);
    }

    Job job = new Job(conf, "word count");
    job.setJarByClass(NoCombiner.class);
    job.setMapperClass(TokenizerMapper.class);

    //disable combiner/default combiner
    if(otherArgs[2] == "true"){
    job.setCombinerClass(IntSumReducer.class);
    }

    //setting multiple reducer classes
    job.setNumReduceTasks(5);
    job.setPartitionerClass(WordCountPartitioner.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);
  }
}
```

PerMapTally:

"This is built on NoCombiner class and it creates a HashMap data structure inside map function, all the words-counts are emitted from HashMap after the value is processed"

```
public class PerMapTally {

   //Partitioner class
   public static class WordCountPartitioner extends Partitioner<Text, IntWritable>{

         @Override
         public int getPartition(Text t, IntWritable value, int numPartitions) {
               String s = t.toString();
               char first = s.charAt(0);
               if(first=='m'||first=='M')
                     return 0;
               if(first=='n'||first=='N')
                     return 1;
               if(first=='o'||first=='O')
                     return 2;
               if(first=='p'||first=='P')
                     return 3;
               if(first=='q'||first=='Q')
                     return 4;
               else
                     return 0;
         }
   }

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

    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException
    {
      IntWritable count = new IntWritable();
      Text text = new Text();
      Map<String, Integer> myMap = new HashMap<String, Integer>();
      StringTokenizer itr = new StringTokenizer(value.toString());

      while (itr.hasMoreTokens())
            {   String s = itr.nextToken();
                char first = s.charAt(0);
               if (isLegal(first))
               {
                     Integer c = myMap.get(s);
                     if(c == null)
                           c = new Integer(0);
                     c=c+1;
                     myMap.put(s, c);
               }
            }
```

```
    Set<String> keys = myMap.keySet();
    for(String s : keys){
          text.set(s);
          count.set(myMap.get(s));
          context.write(text, count);
    }
  }

  //Function to filter input
  //Returns true if the letter is in range m-q or M-Q
   public boolean isLegal(char first)
   {
        if(first >= 'm' && first <= 'q' || first >='M' && first <= 'Q')
        {
              return true;
        }
        else
              return false;
   }
  }

  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 != 3) {
     System.err.println("Usage: wordcount <in> <out> <Use Combiner? true/false>");
     System.exit(2);
   }
   Job job = new Job(conf, "word count");
   job.setJarByClass(PerMapTally.class);
   job.setMapperClass(TokenizerMapper.class);

   if(otherArgs[2] == "true"){
      job.setCombinerClass(IntSumReducer.class);
   }

   //setting multiple reducer classes
   job.setNumReduceTasks(5);
```

```
    job.setPartitionerClass(WordCountPartitioner.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);
  }
}
```

PerTaskTally:

"A HashMap data structure is created and initialized using Setup function and cleanup is done using cleanup function"

```
public class PerTaskTally {

   //Partitioner class
   public static class WordCountPartitioner extends Partitioner<Text, IntWritable>{

         @Override
         public int getPartition(Text t, IntWritable value, int numPartitions) {
               String s = t.toString();
               char first = s.charAt(0);
               if(first=='m'||first=='M')
                     return 0;
               if(first=='n'||first=='N')
                     return 1;
               if(first=='o'||first=='O')
                     return 2;
               if(first=='p'||first=='P')
                     return 3;
               if(first=='q'||first=='Q')
                     return 4;
               else
                     return 0;
         }
   }

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

   //Map variable for in-mapper combination
   private Map<String, Integer> myMap;

   @Override
   protected void setup(Context context) throws IOException, InterruptedException{
         myMap = 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())
         {   String s = itr.nextToken();
             char first = s.charAt(0);
             if(isLegal(first))
             {
                   Integer count = myMap.get(s);
                   if(count == null) count = new Integer(0);
                   count++;
                   myMap.put(s, count);
```

```
                }
        }
  }

  @Override
  protected void cleanup(Context context) throws IOException, InterruptedException{
        IntWritable count = new IntWritable();
        Text text = new Text();
        Set<String> keys = myMap.keySet();
        for (String s : keys){
                text.set(s);
                count.set(myMap.get(s));
                context.write(text, count);
        }
  }

//Function to filter input
//Returns true if the letter is in range m-q or M-Q
  public boolean isLegal(char first)
  {
        if(first >= 'm' && first <= 'q' || first >='M' && first <= 'Q')
        {
                return true;
        }
        else
                return false;
  }
}

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 != 3) {
    System.err.println("Usage: wordcount <in> <out> <Use Combiner? true/false>");
    System.exit(2);
  }
```

```
    Job job = new Job(conf, "word count");
    job.setJarByClass(PerTaskTally.class);
    job.setMapperClass(TokenizerMapper.class);

    if(otherArgs[2] == "true"){
       job.setCombinerClass(IntSumReducer.class);
    }

    //setting multiple reducer classes
    job.setNumReduceTasks(5);
    job.setPartitionerClass(WordCountPartitioner.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);
  }
}
```

Q.1 Do you believe that Combiner was called at all in program SiCombiner?
Ans: Yes the Combiner was called in SiCombiner, as the syslog file show the Combine input records=42989997. However, it cannot be said when hadoop will use combiner.

Q2. What difference the use of a combiner make in SiCombiner Compared to NoCombiner?
Ans:

	SiCombiner	**NoCombiner**
Map input records	21907700	21907700
Map output bytes	412253400	412253400
Combine input records	42989997	0
Reduce input records	18678	42842400
Reduce input groups	849	849
Combine output records	42989997	0
Map output records	42842400	42842400

As shown by the above data, the combiner in the SiCombiner program has to combine all the input records which results in less number of records for the reducer to work upon. Whereas, if we do not use combiner then all of the intermediate input records have to be reduced by the reducer which increases running time.
CPU time spent :
SiCombiner = 524030
NoCombiner = 647780)

Q3. Was the local aggregation effective in PerMApTally compared to NoCombiner?
Ans:

	PerMApTally	NoCombiner
Map input records	21907700	21907700
Map output bytes	396549900	412253400
Combine input records	0	0
Reduce input records	40866300	42842400
Reduce input groups	849	849
Combine output records	0	0
Map output records	849	42842400

The above data clearly represents how the PerMapTally strategy works. The reducer in PerMapTally has to work on lesser number of records also the output bytes in map phase is less as records are clubbed together. If we analyze the running times then it shows that local aggregation was not that effective for this run.

Running time:
Nocombiner : 283 seconds, CPU: 647780
PerMApTally : 286 seconds, CPU: 666750

Q4. What difference do you see in PerMapTally and PerTaskTally? Try to explain the reasons.
Ans:

	PerMApTally	PerTaskTally
Map input records	21907700	21907700
Map output bytes	396549900	229702
Combine input records	0	0
Reduce input records	40866300	21907700
Reduce input groups	849	849

Combine output records	0	0
Map output records	849	18678

The PerTaskTally is a big improvement over PerMapTally as it significantly reduces the total number of output bytes from the map phase which results in lesser number of records for the reduce phase.

This in turn results in better running and CPU time for PerTaskTally as shown below :
PerMapTally:286
PerTaskTally: 182

The way of using HashMap is a big reason for this improvement in PerTaskTally as we create a HashMap only once and reuse it.

Q5. Which one is better SiCombiner or PerTaskTally? Briefly justify your answer.
Ans:

	SiCombiner	PerTaskTally
Map input records	21907700	21907700
Map output bytes	412253400	229702
Combine input records	42989997	0
Reduce input records	18678	21907700
Reduce input groups	849	849
Combine output records	42989997	0
Map output records	42842400	18678

Running time and cpu time:
SiCombiner: 251, CPU: 524030
PerTasktTally: 182, CPU: 326970

The PerTaskTally function produces better results as it uses hashmap to store these keys and the total counts and there values are then emitted. This justifies that the running time of PerTaskTally is much better than the SiCombiner.

Q6. Comapring the results for configuration 1 and 2, do you believe this MapReduce program scales well to larger clusters? Briefly justify your answer.
Ans:
Yes the MapReduce program does scale well to larger clusters as shown by the running time of all the four types of programs, there is a significant improvement of time as we move from a 6 machine cluster to an 11 machine cluster.

As we have specified the reduce tasks as 5 for the programs, in configuration 1 these tasks are distributed over the 5 machines and when we use configuration 2, the same tasks are distributed over 10 machines that is 2 machines working on the same reduce task. This change does bring in improved results and hence the MapReduce program scales well.