CSE 488 (Section 1)
[Summer 2022]

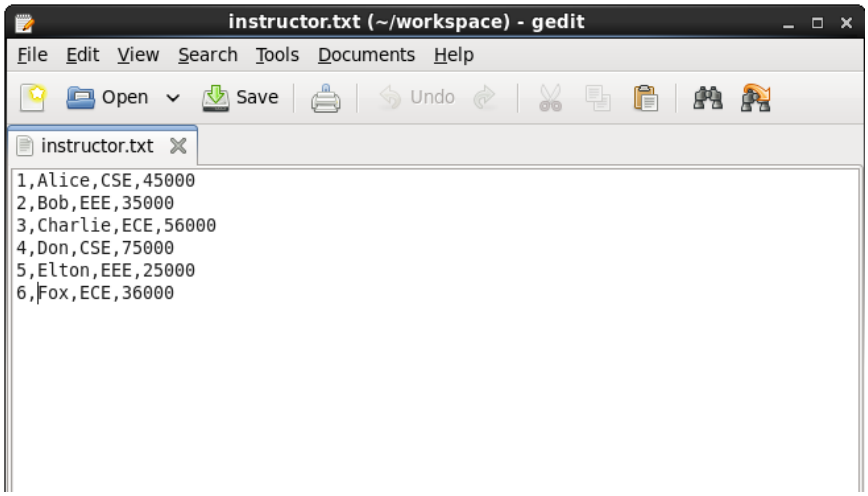
Lab 03 Assignment Submission Report

Assignment Title: MapReduce Programming

Submitted by:
Rizvee Hassan Prito
2019-3-60-041

1. Screenshots

Input File:



Problem 1:

Driver class:

problem1driver.java

```
1
2  import org.apache.hadoop.mapreduce.Job;
10
11 public class problem1driver{
12
13   public static void main(String[] args) throws Exception {
14     /*
15      * Validate that two arguments were passed from the command line.
16      */
17
18     if (args.length != 2) {
19       System.out.printf("Usage: StubDriver <input dir> <output dir>\n");
20       System.exit(-1);
21     }
22
23     Configuration config= new Configuration();
24     Path input= new Path(args[0]);
25     Path output= new Path(args[1]);
26
27
28     /*
29      * Instantiate a Job object for your job's configuration.
30      */
```

```
    @SuppressWarnings("deprecation")
    Job job = new Job(config,"WordCount2");

    /*
     * Specify the jar file that contains your driver, mapper, and reducer.
     * Hadoop will transfer this jar file to nodes in your cluster running
     * mapper and reducer tasks.
     */
    job.setJarByClass(problem1driver.class);
    job.setMapperClass(problem1mapper.class);
    job.setReducerClass(problem1reducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, input);
    FileOutputFormat.setOutputPath(job, output);

    boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}
```

Mapper class:

problem1mapper.java

```
import java.io.IOException;

public class problem1mapper extends Mapper<LongWritable, Text, Text, IntWritable> {

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String[] words= line.split("\n");
    for (String word: words){
        String[] words2= line.split(",");
        if(Integer.parseInt(words2[3])>50000){
            Text outputKey = new Text(words2[0]+","+words2[1]+","+words2[2]);
            IntWritable outputValue = new IntWritable(Integer.parseInt(words2[3]));
            context.write(outputKey,outputValue);
        }
    }

  }
}
```

Reducer class:

problem1reducer.java

```
import java.io.IOException;

public class problem1reducer extends Reducer<Text, IntWritable, Text, IntWritable> {

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

   context.write(key,null);
  }
}
```

Output:

Home / user / cloudera / problem1 / **part-r-00000**

```
3,Charlie,ECE
4,Don,CSE
```

Problem 2:

Driver class:

problem2driver.java

```
33
34      /*
35       * Specify the jar file that contains your driver, mapper, and reducer.
36       * Hadoop will transfer this jar file to nodes in your cluster running
37       * mapper and reducer tasks.
38       */
39      job.setJarByClass(problem2driver.class);
40      job.setMapperClass(problem2mapper.class);
41      job.setReducerClass(problem2reducer.class);
42      job.setOutputKeyClass(Text.class);
43      job.setOutputValueClass(IntWritable.class);
44      FileInputFormat.addInputPath(job, input);
45      FileOutputFormat.setOutputPath(job, output);
46      /*
47       * TODO implement
48       */
49
50      /*
51       * Start the MapReduce job and wait for it to finish.
52       * If it finishes successfully, return 0. If not, return 1.
53       */
54      boolean success = job.waitForCompletion(true);
55      System.exit(success ? 0 : 1);
56    }
57 }
```

Mapper class:

problem2driver.java problem2mapper.java

```java
import java.io.IOException;

public class problem2mapper extends Mapper<LongWritable, Text, Text, IntWritable> {

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String[] words= line.split("\n");
    for (String word: words){
        String[] words2= line.split(",");
        if(Integer.parseInt(words2[3])>40000){
            Text outputKey = new Text("Total_number_of_instructor");
            IntWritable outputValue = new IntWritable(1);
            context.write(outputKey,outputValue);
        }
    }

  }
}
```

Reducer class:

problem2driver.java problem2mapper.java problem2reducer.java

```java
import java.io.IOException;

public class problem2reducer extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
      int sum=0;
      for (IntWritable value:values){
           sum+=value.get();
       }
      context.write(key,new IntWritable(sum));
  }
}
```

Output:

Home / user / cloudera / problem2 / **part-r-00000**

```
Total_number_of_instructor      3
```

Problem 3:

Driver class:

problem3driver.java

```
    Path output = new Path(args[1]);
    //Instantiate a Job object for your job's configuration.
    @SuppressWarnings("deprecation")
    Job job = new Job(config, "AverageForGroupWithCombiner");
    /*
     * Specify the jar file that contains your driver, mapper, and reducer.
     * Hadoop will transfer this jar file to nodes in your cluster running
     * mapper and reducer tasks.
     */
    job.setJarByClass(problem3driver.class);
    job.setMapperClass(problem3mapper.class);
    job.setCombinerClass(problem3combiner.class);
    job.setReducerClass(problem3reducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(CustomAverageSumTuple.class);
    FileInputFormat.addInputPath(job, input);
    FileOutputFormat.setOutputPath(job, output);
    /*
     * Start the MapReduce job and wait for it to finish.
     * If it finishes successfully, return 0. If not, return 1.
     */
    boolean success = job.waitForCompletion(true);
    System.exit(success ? 0 : 1);
  }
}
```

Custom data type:

problem3driver.java | CustomAverageSumTuple.java

```java
import java.io.DataInput;

public class CustomAverageSumTuple implements Writable{

    private Double value = new Double(0);
    private Double total = new Double(0);
    private long count = 1;

    public Double getTotal(){
        return total;
    }

    public void setTotal(Double total){
        this.total = total;
    }

    public Double getValue(){
        return value;
    }

    public void setValue(Double value){
        this.value = value;
    }

    public long getCount(){
        return count;
    }
    public void setCount(long count){
        this.count = count;
    }
    @Override
    public void readFields(DataInput in) throws IOException {
        // TODO Auto-generated method stub
        total= in.readDouble();
        value = in.readDouble();
        count = in.readLong();
    }
    @Override
    public void write(DataOutput out) throws IOException {
        // TODO Auto-generated method stub
        out.writeDouble(total);
        out.writeDouble(value);
        out.writeLong(count);

    }
    public String toString(){
        return value + "," + total;
    }
```

Mapper class:

problem3driver.java | *CustomAverageSumTuple.java | problem3mapper.java

```java
import java.io.IOException;

public class problem3mapper extends Mapper<LongWritable, Text, Text, CustomAverageSumTuple> {

    private CustomAverageSumTuple tuple = new CustomAverageSumTuple();
  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String content = value.toString();

    String[] lines = content.split("\n");

    for(String line: lines){
        String[] tokens = line.split(",");
        tuple.setValue(Double.parseDouble(tokens[3]));
        tuple.setTotal(Double.parseDouble(tokens[3]));
        tuple.setCount(1);
        context.write(new Text(tokens[2]), tuple);

    }
  }
}
```

Combiner class:

problem3driver.java | *CustomAverageSumTuple.java | problem3mapper.java | *problem3combiner.java

```java
import java.io.IOException;

public class problem3combiner extends Reducer<Text, CustomAverageSumTuple, Text, CustomAverageSumTuple> {

    private CustomAverageSumTuple result = new CustomAverageSumTuple();
  @Override
  public void reduce(Text key, Iterable<CustomAverageSumTuple> values, Context context)
      throws IOException, InterruptedException {


      double total = 0;
      double sum = 0;
      long count = 0;
      for(CustomAverageSumTuple value: values){
          sum += value.getValue() * value.getCount();
          count += value.getCount();
          total+=value.getTotal();
      }
      result.setCount(count);
      result.setValue(sum/count);
      result.setTotal(total);

      context.write(key, result);
  }
}
```

Reducer class:

problem3driver.java | *CustomAverageSumTuple.java | problem3mapper.java | *problem3combiner.java | *problem3reducer.java

```
import org.apache.hadoop.mapreduce.Reducer;

public class problem3reducer extends Reducer<Text, CustomAverageSumTuple, Text, CustomAverageSumTuple> {

    private CustomAverageSumTuple result = new CustomAverageSumTuple();
  @Override
  public void reduce(Text key, Iterable<CustomAverageSumTuple> values, Context context)
      throws IOException, InterruptedException {

      double total = 0;
      double sum = 0;
      long count = 0;
      for(CustomAverageSumTuple value: values){
          sum += value.getValue() * value.getCount();
          count += value.getCount();
          total +=value.getTotal();
      }
      result.setCount(count);
      result.setValue(sum/count);
      result.setTotal(total);

      context.write(key, result);
  }
}
```

Output:

Home / user / cloudera / problem3 / **part-r-00000**

```
CSE     60000.0,120000.0
ECE     46000.0,92000.0
EEE     30000.0,60000.0
```

Learning Outcome:

From this lab we have learned how to solve problems by implementing custom data type in MapReduce programming style and execute the program in Hadoop Framework.