MapReduce Programming

1. Enter the following URL in Google Chrome
   https://classroom-vc.gre.ac.uk

   Launch the Hadoop VM image, following the instructions from last week's lab.

2. Once the Hadoop desktop has loaded, right click on the desktop and start a terminal
3. At the command prompt type:

   ```
   start-dfs.sh
   start-yarn.sh
   ```

   to start Hadoop.

4. Change directory into your `workspace` directory (if you don't have a `workspace` directory then starting Eclipse will create one for you)

   ```
   cd ../workspace
   ```

5. Copy the Lab6 directory from `Datasets` into your `workspace` directory
6. Change current directory to the `workspace/Lab6` directory
7. Open the WordCount.java file using Geany or Eclipse

```
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

  public static class Map
       extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

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

    public void map(LongWritable key, Text value, OutputCollector<Text,
                    IntWritable> output, Reporter reporter) throws IOException {

      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        output.collect(word, one);
      }
    }
  }

  public static class Reduce extends MapReduceBase
            implements Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
             OutputCollector<Text,   IntWritable>   output,   Reporter   reporter)   throws
      IOException {
```

```
      int sum = 0;
      while (values.hasNext()){
        sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
    }
  }

  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }


}
```

8. Create a directory for the compiled WordCount classes

   ```
   mkdir wc_classes
   ```

9. MapReduce programming in Java uses a number of libraries. Compile the WordCount.java file using libraries from the Hadoop Java API as follows:

```
javac -classpath $HADOOP_HOME/share/hadoop/common/hadoop-common-
2.4.1.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-
core-2.4.1.jar:$HADOOP_HOME/share/hadoop/common/lib/hadoop-annotations-
2.4.1.jar -d wc_classes WordCount.java
```

You can copy this command line from the java_command_line.txt file contained in the Lab6 folder

10. Package the compiled classes into a jar file

    ```
    jar -cvf wordcount.jar -C wc_classes/ .
    ```

11. Put the test.txt file from last week’s Lab onto the HDFS if it is not already there

    ```
    hdfs dfs –put test.txt
    ```

12. Run the word count map reduce job using following in the command line:

    ```
    hadoop jar wordcount.jar WordCount test.txt output
    ```

13. Check the results of the Word Count MapReduce, by copying the output directory from the HDFS to your local file system

    ```
    hdfs dfs –get output
    ```

14. Try running Word Count on some of the books from Lab 5 (last week's lab).
15. If you are familiar with Java you might like to try modifying the WordCount.java example so that it counts hash tags:
    - Open WordCount.java in Eclipse (or Geany) and save it to a new file called HashTagCount.java
    - You will need to change the class name to HashTagCount and also the change parts of the main(…) method as follows:

      ```
      JobConf conf = new JobConf(HashTagCount.class);
      conf.setJobName("hashtagcount");
      ```

    - You will need to change the `map(…)` function to count occurrences of particular hashtags. You can use the `startsWith()` method of the java String class to detect hashtags. E.g.

      ```
      if(token.startsWith("#")){…}
      ```

    - The Reduce class can be left as it is
    - You can compile, package and run you code using similar commands to those used for the WordCount example above.
    - There is a data file called `hashtags.txt` in the Lab6 folder for you to test your code on.