Assignment 1: MapReduce with Hadoop

Jean-Pierre Lozi

January 24, 2015

Provided files An archive that contains all files you will need for this assignment can be found at the following URL:

```
http://sfu.ca/~jlozi/cmpt732/assignment1.tar.gz
```

Download it and extract it (using "`tar -xvzf assignment1.tar.gz`", for instance).

1 *Word Count*, Your First Hadoop Program

The objective of this section is to write a very simple Hadoop program that counts the number of occurrences of each word in a text file. In Hadoop, this program, known as *Word Count* is the equivalent of the standard *Hello, world!* program you typically write when you learn a new programming language.

1.1 Setting up your environment

Question 1 We'll first have to configure `ssh` to make it possible to access our Hadoop cluster. To do so, add the following lines to your `/.ssh/config` file (create the file if it doesn't exist):

```
Host hadoop.rcg.sfu.ca
    ForwardAgent yes
    ProxyCommand ssh rcg-linux-ts1.rcg.sfu.ca nc hadoop.rcg.sfu.ca 22
```

Make sure you can connect to the cluster, and create a directory named `CMPT732` in your home directory.

```
$ ssh <username>@hadoop.rcg.sfu.ca
$ mkdir CMPT732
```

Don't forget to replace `<username>` with your username.

Question 2 We will now download Hadoop. We will use Hadoop 2.4.0 since it is the version that is used on our local Hadoop cluster. Download the Hadoop source and extract it:

```
$ wget https://archive.apache.org/dist/hadoop/core/hadoop-2.4.0/hadoop-2.4.0.tar.gz
$ tar -xvzf hadoop-2.4.0.tar.gz
```

Hadoop provide two APIs, the old one (that dates back from versions prior to 0.20.x) and the new one in this course. For backward compatibility reasons, both can be used with Hadoop 2.4, however, we will only use the new one. Always make sure that you only use classes from the `org.apache.hadoop.mapreduce` package, not `org.apache.hadoop.mapred`.

Question 3 Launch Eclipse:

```
$ eclipse &
```

If you don't have one already, create a workspace. Create a new Java project named `CMPT732A1-WordCount`. Right click on the root node of the project, and pick *Build Path* → *Configure Build Path* in the contextual menu. In the *Libraries* tab, click *Add External Jars...*, and locate the `hadoop-2.4.0` directory from the previous question and add `hadoop-common-2.4.0.jar` from `share/hadoop/common/`. Repeat the operation for the following archives:

- `hadoop-2.4.0/share/hadoop/hdfs/hadoop-hdfs-2.4.0.jar`
- `hadoop-2.4.0/share/hadoop/mapreduce/hadoop-mapreduce-client-common-2.4.0.jar`
- `hadoop-2.4.0/share/hadoop/mapreduce/hadoop-mapreduce-client-core-2.4.0.jar`

Question 4 Add a new Java class to your project named `WordCount` in the `org.CMPT732A1` package. We want to be able to generate a Jar archive of the project and to upload it to the Hadoop cluster automatically each time we compile it. To do so, create a `build.xml` file in the `src/` directory of the default package. The file will contain the following:

```
<?xml version="1.0" ?>
<project name="WordCount" default="CreateJar">
    <target name="CreateJar" description="Create Jar file">
        <jar jarfile="WordCount.jar" basedir="../bin/" includes="org/CMPT732A1/*.class" />
        <scp file="WordCount.jar" todir="<username>:<password>@rcg-linux-ts1.rcg.sfu.ca:CMPT732/"
             trust="true" />
    </target>
</project>
```

Replace `<username>` and `<password>` with your username and password. If you don't feel comfortable with storing your password in plain text in a configuration file, you can skip the "`<scp ...`" line, but you'll have to upload the Jar archive using the `scp` command each time you compile it. If you used the "`<scp ...`" line, you will have to download the following Jar:

```
http://sfu.ca/~jlozi/cmpt732/jsch-0.1.51.jar
```

Pick *Window* → *Preferences* → *Ant* → *Runtime* → *Global Entries* → *Add External JARs...*, and select `jsch-0.1.51.jar`. Click *OK*. You only have to do that once as long as you keep the same workspace.

Right-click on the root node of the project, and select *Properties*. Select *Builders*, then click *New...*. Select *Ant Builder*. In the *Main* tab, click on *Browse Workspace...* for the *Buildfile*, and find the `build.xml` file. In the *Targets* tab, click *Set Targets...* for the *Manual Build*, and select the *CreateJar* target. Click *OK* three times. Building everything (*Ctrl+B*) should produce a file named `WordCount.jar`, and upload it to the Hadoop cluster. You will have to repeat these steps for each project for which you use a `build.xml` file.

1.2 Writing and running the code

Question 1 Suppose we use an input file that contains the following lyrics from a famous song:

```
We’re up all night till the sun
We’re up all night to get some
We’re up all night for good fun
We’re up all night to get lucky
```

The input pairs for the Map phase will be the following:

```
(0, "We’re up all night to the sun") (31, "We’re up all night to get some")
(63, "We’re up all night for good fun") (95, "We’re up all night to get lucky")
```

The key is the byte offset starting from the beginning of the file. While we won’t need this value in *Word Count*, it is always passed to the Mapper by the Hadoop framework. The byte offset is a number that can be large if there are many lines in the file.

- What will the output pairs look like?
- What will be the types of keys and values of the input and output pairs in the Map phase?

Remember that instead of standard Java data types (`String`, `Int`, etc.), Hadoop uses data types from the `org.apache.hadoop.io` package. You can check Hadoop’s API at the following URL:

```
http://hadoop.apache.org/docs/r2.4.0/api/
```

Question 2 For the Reduce phase, some of the output pairs will be the following:

```
("up", 4) ("to", 3) ("get", 2) ("lucky", 1) ...
```

- What will the input pairs look like?
- What will be the types of keys and values of the input and output pairs in the Reduce phase?

Question 3 Find the `WordCount.java` file that is provided with this week’s assignment, and copy its contents to the corresponding file in your project. Find the definitions of the `Map` class and the `map()` function:

```
public static class Map extends Mapper</*?*/, /*?*/, /*?*/, /*?*/> {

    @Override
    public void map(/*?*/ key, /*?*/ value, Context context)
            throws IOException, InterruptedException {
        ...
```

Use the data types you found in Question 1 to replace the `/*?*/` placeholders. Similarly, find the definitions of the `Reduce` class and the `reduce()` function and replace the `/*?*/` placeholders with the data types found in Question 2. You will also have to find which arguments to pass to `job.setOutputKeyClass()` and `job.setOutputValueClass()` in `estimate()`.

Question 4 Write the `map()` function. We want to make sure to disregard punctuation: to this end, you can use `String.replaceAll()`. In order to split lines into words, you can use a `StringTokenizer`.

Question 5 Write the `reduce()` function. When you're done, make sure that compiling the project (*Ctrl+B*) doesn't produce any errors.

Question 6 If you haven't added a "`<scp ...`" line to your `build.xml` file, copy the Jar file that has been generated when building the project (`WordCount.jar`) to the Hadoop cluster. You can use the following command, assuming that you use Eclipse default's workspace path and that you used the naming conventions from the previous questions:

```
$ scp ~/workspace/CMPT732A1-WordCount/src/WordCount.jar hadoop.rcg.sfu.ca:~/CMPT732
```

It will copy the archive to the `CMPT732` directory on the Hadoop cluster.

Question 8 Large text files were created by concatenating books from Project Gutenberg.[1] You can find these text files in the following directory on the Hadoop cluster:

```
/cs/bigdata/datasets/
```

The files are named `gutenberg-<size>.txt`, where `<size>` is the size of the file. Three text files are provided, of sizes 100 MB, 200 MB, and 500 MB. We will now run `WordCount` on the 100 MB file. First, we have to copy that file to the HDFS. To do so, run the following command on the cluster:

```
$ hadoop fs -copyFromLocal /cs/bigdata/datasets/gutenberg-100M.txt
```

It will copy `gutenberg-100M.txt` to `/user/<username>/` (where `<username>` is your user name) on the HDFS.

Question 9 It's now time to see if your `WordCount` class works. On the cluster, run the following command in the `~/CMPT732` directory:

```
$ hadoop jar WordCount.jar org.CMPT732A1.WordCount gutenberg-100M.txt output/
```

Did it work so far? No exceptions? If so, very good. Otherwise, you can edit your `WordCount.java` file again, recompile it, copy it again to the cluster like you did it Question 6 if needed, remove the `output/` directory from the HDFS (`hadoop fs -rm -r output`) and launch the command above again. When everything works, you can merge the output from the HDFS to a local file:

```
$ hadoop fs -getmerge output/ output.txt
```

Open `output.txt`, the results should look like this:

[1] Project Gutenberg (`https://www.gutenberg.org`) is a volunteer effort to digitize and archive cultural works (mainly public domain books).

```
A       18282
AA      16
AAN     5
AAPRAMI 6
AARE    2
AARON   2
AATELISMIES     1
...
```

If that is what you see, congratulations! Otherwise, fix your code and your setup until it works. What is the most frequent word?

Question 10 In order to copy data from your local file system to the HDFS you used the following command:

```
$ hadoop fs -copyFromLocal /cs/bigdata/datasets/gutenberg-100M.txt
```

In addition to the `-copyFromLocal` and `-copyToLocal` operations that are pretty self-explanatory, you can use basic UNIX file system commands on HDFS by prefixing them with "`hadoop fs -`". So for instance, instead of `ls`, you would type:

```
$ hadoop fs -ls
```

The output should look like this:

```
drwx------   - jlozi hdfs          0 2014-10-09 23:00 .Trash
drwx------   - jlozi hdfs          0 2014-10-09 14:55 .staging
drwxr-xr-x   - jlozi hdfs          0 2014-10-09 14:55 output
-rw-r--r--   3 jlozi hdfs  104857600 2014-10-09 13:01 gutenberg-100M.txt
```

The result is similar to what you would see for a standard `ls` operation on a UNIX file system. The only difference here is the second column that shows the replication factor of the file. In this case, the file `gutenberg-100M.txt` is replicated three times. Why don't we have a replication factor for directories?

Question 11 For more information on the HDFS commands you can use, type:

```
$ hadoop fs -help
```

Create a directory named `cmpt732_assignment1` on the HDFS, and move the file `gutenberg-100M.txt` into it. What command would you use to show the size of that file, in megabytes? How would you display its last kilobyte of text? How would you display its last five lines in an efficient manner?

Question 12 How many Map and Reduce tasks did running *Word Count* on `gutenberg-100M.txt` produce? Run it again on `gutenberg-200M.txt` and `gutenberg-500M.txt`. Additionally, run the following command on the cluster:

```
$ hdfs getconf -confKey dfs.blocksize
```

What is the link between the input size, the number of Map tasks, and the size of a block on HDFS?

Question 13 Edit `WordCount.java` to make it measure and display the total execution time of the job. Experiment with the `mapreduce.input.fileinputformat.split.maxsize` parameter. You can change its value using:

```
job.getConfiguration().setLong("mapreduce.input.fileinputformat.split.maxsize", <value>);
```

How does changing the value of that parameter impact performance? Why?

Question 14 If you ran buggy versions of your code, it is possible that some of your Hadoop jobs are still running (they could be stuck in an infinite loop, for instance). Make sure that none are running using the following command:

```
$ mapred job -list
```

If some of your jobs are still running, you can kill them using:

```
$ mapred job -kill <job_id>
```

To kill all of your running jobs, you can use the following command (where `<username>` is your username):

```
$ mapred job -list | grep <username> | grep job_ | awk '{ system("mapred job -kill " $1) } '
```

During the assignments, don't forget to check once in a while that you don't have jobs uselessly wasting the Hadoop cluster's resources! Additionally, remove all data files you copied to the HDFS at the end of each exercise, to avoid wasting hard drive space.

2 MapReduce for Parallelizing Computations

We will now estimate the value of Euler's constant (e) using a Monte Carlo method. Let X_1, X_2, ..., X_n be an infinite sequence of independent random variables drawn from the uniform distribution on $[0,1]$. Let V be the least number n such that the sum of the first n samples exceeds 1:

$$V = \min\{n \mid X_1 + X_2 + ... + X_n > 1\}$$

The expected value of V is e:

$$\mathrm{E}(V) = e$$

Each Map task will generate random points using a uniform distribution on $[0,1]$ in order to find a fixed number of values of n. It will output the number of time each value of n has been produced. The Reduce task will sum the results, and using them, the program will calculate the expected value of V and print the result.

Question 1 How can we pass a different seed to initialize random numbers to each Map task, in order to make sure that no two Map tasks will work on the same values? What other parameter will we have to pass to each Map task? What will be the type of the keys and values of the input of Map tasks? What will they represent?

Question 2 Map tasks will produce a key/value pair each time each time they produce a value for n. What will the types of the keys and values output by Map tasks? What will they represent?

Question 3 The Reduce task sums the results. What will the types of the keys and the values of the Reduce task be? What will they represent?

Question 4 Create a new Java project named `CMPT732A1-EulersConstant`. Create a new class in the `org.CMPT732A1` package named `EulersConstant`. Copy/paste the contents of the provided `EulersConstant.java` file into your own. Follow what you did in Section 1 to produce a working Hadoop project: add Hadoop Jars, create a `build.xml` file (you will have to modify it slightly), etc. Replace the `/*?*/` placeholders from `EulersConstant.java` using the answers from the previous questions.

Question 5 Write the `map()` function. You can simply use a `Random.nextDouble()` object to generate random numbers drawn from the uniform distribution on $[0,1]$.

Question 6 Write the `reduce()` function. *Hint:* remember *Word Count*!

Question 7 We will now have to send the right key/value pairs to each Mapper. To this end, we will produce one input file for each Mapper in the input directory. Find the following comment in the code:

```
// TODO: Generate one file for each map
```

And generate the files. *Hint:* you can use a `SequenceFile` to produce a file that contains key/value pairs.

Question 8 We will now compute the result using the output from the Reduce task. Find the following comment in the code:

```
// TODO: Compute and return the result
```

A `SequenceFile.Reader` that reads the output file is created for you. Use it to compute the result. You will have to return a `BigDecimal`, i.e. an arbitrary-precision decimal number. Don't forget to close the `SequenceFile.Reader` and to delete the temporary directory that contains the input and output files when you're done.

Question 9 Copy the program to the Hadoop cluster if needed (depending on what you put in your `build.xml` file), and run it:

```
$ cd ~/CMPT732
$ hadoop jar EulersConstant.jar org.CMPT732A1.EulersConstant 10 100000
```

The first parameter is the number of Mappers, and the second parameter is the number of values each Mapper will produce for n. How many accurate digits does your program find? The value of e is:

$$e = 2.7182818284...$$

Question 10 How long is the Reducer phase? We can make it faster using a custom Combiner phase. The Combiner phase is similar to the Reduce phase, except it's executed locally at the end of each Map task. In our case, the Combiner phase will do the same thing as the Reducer phase. If you picked the right types, you can just use `job.setCombinerClass()` to tell Hadoop to use your Reducer as a Combiner. If you didn't, modify your types! Does using a Combiner phase speed things up? Why?

Question 11 So far, we've used Java's `Random` class to produce random numbers. Better implementations exist, such as the `MersenneTwister` class from *Apache Commons Math*. Try another random number generator. Does it improve results? You can use the current system timestamp to initialize your random number generator in order to obtain different results each time and compute the variance.

3 NCDC Weather Data

You have just been hired by the NCDC[2] to help with analyzing their large amounts of weather data (about 1GB per year). The NCDC produces CSV (Comma-Separated Values) files with worldwide weather data for each year. Each line of one of these files contains:

- The weather station's code.
- The date, in the ISO-8601 format.
- The type of value stored in that line. All values are integers. `TMIN` (resp. `TMAX`) stands for minimum (resp. maximum) temperature. Temperatures are expressed in tenth of degrees Celsius. `AWND` stands for average wind speed, and `PRCP` stands for precipitation (rainfall), etc. Several other types of records are used (`TOBS`, `SNOW`, ...).
- The next field contains the corresponding value (temperature, wind speed, rainfall, etc.)
- All lines contain five more fields that we won't use in this exercise.

We will work on the CSV file for 2013, which has been sorted by date first, station second, and value type third, in order to ease its parsing. It can be found at the following location on the server:
`/cs/bigdata/datasets/ncdc-2013-sorted.csv`
Here is a sample of that file:

```
...
FR000007650,20130102,PRCP,5,,,S,
FR000007650,20130102,TMAX,111,,,S,
FR000007747,20130102,PRCP,3,,,S,
FR000007747,20130102,TMAX,117,,,S,
FR000007747,20130102,TMIN,75,,,S,
FR069029001,20130102,PRCP,84,,,S,
FR069029001,20130102,TMAX,80,,,S,
FS000061996,20130102,PRCP,0,,,S,
FS000061996,20130102,TMAX,206,,,S,
FS000061996,20130102,TMIN,128,,,S,
GG000037279,20130102,TMAX,121,,,S,
GG000037308,20130102,TMAX,50,,,S,
GG000037308,20130102,TMIN,-70,,,S,
GG000037432,20130102,SNWD,180,,,S,
GG000037432,20130102,TMAX,15,,,S,
GG000037432,20130102,TMIN,-105,,,S,
...
```

As you can see, not all stations record all data. For instance, `FR069029001` only recorded rainfall and maximum temperature on 01/02/2013. Not all stations provide data for every day of the year either.

[2] National Climatic Data Center, see: `http://www.ncdc.noaa.gov`.

Question 1 The NCDC wants to plot the difference between the maximum and the minimum temperature in Central Park for each day in 2013. There is a weather station in Central Park: its code is `USW00094728`.[3] If we have a look at the `TMIN` and `TMAX` records for that weather station, they look like this (`USW00094728` provides minimum and maximum temperature data for every day of the year).

```
USW00094728,20130101,TMAX,44,,,X,2400
USW00094728,20130101,TMIN,-33,,,X,2400
USW00094728,20130102,TMAX,6,,,X,2400
USW00094728,20130102,TMIN,-56,,,X,2400
USW00094728,20130103,TMAX,0,,,X,2400
USW00094728,20130103,TMIN,-44,,,X,2400
...
```

In order to ease plotting the data, you are asked to generate a one-column CSV file which one value for each day (the temperature variation, in degrees Celsius):

```
7.7,
11.6,
4.4,
...
```

Jeff, that other new intern that you dislike, proposes to use a MapReduce job that does the following:

- The Map task(s) send(s) `(<tmin>, <tmax>)` pairs to the Reducer. Temperatures are converted to degrees Celsius. The output will be:

  ```
  (4.4, -3.3)
  (6, -5.6)
  (0, -4.4)
  ...
  ```

- For each key/value pair, the Reduce task substracts the minimum temperature from the maximum temperature, converts it to degrees, and writes the result to a file.

Your boss is impressed by Jeff's skills, but you know better and tell your boss that it can't work. What is wrong with this approach?

Question 2 Instead, you propose that the Map phase will be a cleanup phase that discards useless records, and for each day, it will calculate the temperature difference. What will the key and values output by the Map tasks be? What types will they be?

Question 3 Can the Mapper produce a key/value pair from a single input? How can we solve this issue? For each day and weather station, the `TMAX` record always precedes the `TMIN` in NCDC's data, and we suppose that no split occurs between a `TMAX` and a `TMIN` record.

[3] A list of all station codes can be found at: `ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/ghcnd-stations.txt`.

Question 4 By following this approach, what work will be left to the Reduce task? Reducer classes extend Hadoop's `Reducer` class. Read Hadoop's documentation for that class, in particular, what the default behavior of its `reduce()` function is. What can you deduce from this?

Question 5 Create a new Java project named `CMPT732A1-TemperatureVariations`. Follow what you did in Section 1 to produce a working Hadoop project: add Hadoop Jars, create a `build.xml` file, etc. Create a `TemperatureVariations` class in the `org.CMPT732` package that will create and start the job (get your inspiration from `WordCount.java`), and write the `Map` class. Like with the `WordCount` class, the first parameter will be a path to the input file on the HDFS, and the second parameter will be the directory where to store the results (on the HDFS, too). Once the MapReduce job is done executing, your program must read the output file (since you will use a single reducer, the output will be stored in a single file named `part-r-00000` in the output directory), and use it to produce a CSV file that contains the results in the local filesystem. Make sure that `part-r-00000` is a sequence file, using:

```
job.setOutputFormatClass(SequenceFileOutputFormat.class);
```

You can read the output file using a `SequenceFile.Reader`, that you will initialize using:

```
SequenceFile.Reader reader =
        new SequenceFile.Reader(job.getConfiguration(),
                                SequenceFile.Reader.file(new Path(outputDirectory,
                                                                  "part-r-00000")));
```

Of course, the variables `job` and `outputDirectory` must be initialized correctly beforehand.

Question 6 Run your program on `ncdc-2013-sorted.csv` (don't forget to copy the file to the HDFS first). The first values should be 7.7, 6.2, 4.4... Plot the results.

Question 7 Impressed with your work, your boss now asks you to plot average worldwide temperature variations. Similarly, you will produce a one-column CSV file. Since not all stations provide all of the data, you will have to be careful that you always subtract the minimum temperature from the maximum temperature of the same station. If a station doesn't provide both the minimum or maximum temperature for that day, it will be ignored. Keep in mind that given the way the file is sorted, the minimum temperature will always follow the maximum temperature for a station on a given day.

Write a Reducer that will perform the job, and modify your Mapper accordingly. While your Reducer can use the type `FloatWritable` for the result, you will use a `double` when you sum up results to compute the average, in order to make sure to not lose precision when summing up a large number of floating-point values. Plot the results (they should start with 9.857165, 9.882375, 10.542754...).

Question 8 Save the results from the previous question. Reduce the split size by a factor of ten, using:
`mapreduce.input.fileinputformat.split.maxsize`
Use the `diff` command to compare the results you get with the ones from the previous question. Are there differences? Why?

Question 9 To solve the issue, we're going to write a custom `InputFormat` and a custom `RecordReader`. These classes will split the input files by *record*, instead of lines: each record will be a series of lines that contain all data from one station for a given day. You can start with this code:

```
public class NCDCRecordInputFormat extends TextInputFormat {

     public RecordReader<LongWritable, Text> createRecordReader(InputSplit split,
                                                               TaskAttemptContext context) {
         return new NCDCRecordReader();
     }

     public class NCDCRecordReader extends RecordReader<LongWritable, Text> {

         private BufferedReader in;
         private long start, end;
         private LongWritable currentKey = new LongWritable();
         private Text currentValue = new Text();

         ...

         @Override
         public void initialize(InputSplit split, TaskAttemptContext context)
                throws IOException, InterruptedException {

            String line;
            Configuration job = context.getConfiguration();

            // Open the file.
            FileSplit fileSplit = (FileSplit)split;
            Path file = fileSplit.getPath();
            FileSystem fs = file.getFileSystem(job);
            FSDataInputStream is = fs.open(file);
            in = new BufferedReader(new InputStreamReader(is));

            // Find the beginning and the end of the split.
            start = fileSplit.getStart();
            end = start + fileSplit.getLength();

            ...

            // TODO: write the rest of the function. It will initialize needed
            // variables, move to the right position in the file, and start
            // reading if needed.
        }

        @Override
        public boolean nextKeyValue() throws IOException, InterruptedException {
            // TODO: read the next key/value, set the key and value variables
            // to the right values, and return true if there are more key and
            // to read. Otherwise, return false.
        }

       @Override
        public void close() throws IOException {
            in.close();
        }

        @Override
        public LongWritable getCurrentKey() throws IOException, InterruptedException {
            return currentKey;
        }

        @Override
        public Text getCurrentValue() throws IOException, InterruptedException {
            return currentValue;
        }

        @Override
        public float getProgress() throws IOException, InterruptedException {
            // TODO: calculate a value between 0 and 1 that will represent the
            // fraction of the file that has been processed so far.
        }
    }
}
```

No need to retype everything, you will find this file in the provided archive (`NCDCRecordInputFormat.java`).

You can use a `DataOutputBuffer` in which you will write the contents of the value you are currently generating. Handling the limit between splits is a very delicate operation: since the limit between splits can occur in the middle of a line, you have to decide which records will go to previous and the next Mapper. You also have to make sure you never skip a record, and that it never happens that two Mappers read the same record. Debug your functions by testing them locally. Make sure your program uses your new `NCDCInputFormat` class, and rewrite the `Map` class to make it work with records instead of lines.

Once you're done and it seems to work, *make sure that changing the value of the `split.maxsize` parameter does not affect the results anymore*. In order to help with debugging, you can use:

```
job.setNumReducers(0);
```

It will make it possible to see the output of Mappers in files named `part-m-XXXXX` on the HDFS. Additionally, you can use:

```
job.setOutputFormatClass(TextOutputFormat.class);
```

To make sure that the output consists of text files (which are easier to read than binary `SequenceFiles`).

Question 10 You are now asked to plot temperature variations for each continent. The first two letters of each weather station's code indicates which country it's based in. In the files provided with the assignment, you will find the text file `country_codes.txt` which contains a list of country codes for each continent:

Africa: AG,AO,BC,BN,BY,CD,CN,CT,CV,DJ,EG,EK,ER,ET,GA,GB,GH,GV,KE,LI,LT,LY,MA,MI,ML,MO,MP,MR,MZ,NG,NI,PP,PU,RE,RW,SE,SF,SG,SL,SO,SU,TO,TP,TS,TZ,UG,UV,WA,WZ,ZA,ZI

Asia:AF,BA,BG,BT,BX,CB,CE,CH,CK,HK,ID,IN,IO,IR,IS,IZ,JA,JO,KG,KT,KU,KZ,LA,LE,MC,MG,MU,MV,MY,NP,PK,PS,QA,RP,SA,SN,SY,TC,TH,TI,TW,TX,UZ,VM,YE

...

We want to use this information to make it so that we'll have one Reduce task that will calculate the averages for each continent. To this end, we'll create a custom Partitioner:

`http://hadoop.apache.org/docs/r2.4.0/api/org/apache/hadoop/mapreduce/Partitioner.html`

Read up on the `Partitioner` class. Are the keys we've been using until now satisfactory?

Question 11 Create a new class for the keys that solves the problem. Since the keys are sorted during the Shuffle/Sort phase, the class of the key has to implement the `WritableComparable` interface. We will use a single Reducer for now. Use the new class you created for your key in your Mapper and in your Reducer, and make it so that your Reducer will output the average temperature variation for each day in each continent. You will produce a CSV file with one column for each continent (you will add a header for each column), and you will disregard unrecognized country codes.

Question 12 Write a new Partitioner that partitions the data based on continents. Run your code with six Reducers. Plot the results.

4 Back to Counting

Question 1 Create a new class in your project `CMPT732A1-WordCount` named `WordCountByLength` that counts the number of words of each length: it will return the number of 1-letter words, of 2-letter words, and so on. Run your program on one of the `gutenberg-<size>.txt` files, and plot the results (you can use `gnuplot`). Look up the distribution of word lengths in English. You can check the following paper on Arxiv, for instance:

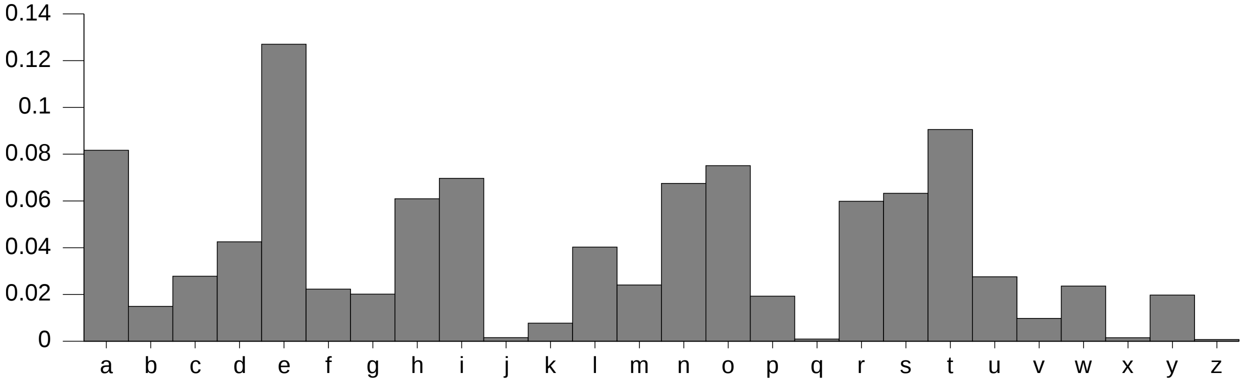
`http://arxiv.org/pdf/1207.2334.pdf`

Are your results close?

Question 2 Create a new class in the same project named `LetterCount` that calculates the frequency of each letter in a file. Plot the results obtained using one of the `gutenberg-<size>.txt` files. You can find the relative frequencies of each letter in the English language at the following URL:

`http://en.wikipedia.org/wiki/Letter_frequency`

Your graph should look like the one from the Wikipedia page:



How similar are your results?