M.Sc. in Data Science

Large Scale Data Management

Assignment 1 - MapReduce

Phevos A. Margonis - f3352317

Instructor: Liakos Panagiotis

February 2, 2024

Abstract

This assignment investigates the use of the Hadoop Distributed File System (HDFS) and MapReduce to process and analyze large datasets across distributed clusters. Utilizing Vagrant and Docker for virtualization, we detail two MapReduce jobs: a word count of "Don Quixote" and analysis of Spotify song metrics for danceability.

Contents

1 Introduction

Hadoop is a file system distinguished by its capability to distribute the storage and processing of large datasets across multiple clusters of computers, facilitating scalability, fault tolerance, and cost-effectiveness. Central to its operation is the MapReduce programming model, capable of accessing chunks of a dataset scattered across the cluster, scanning them, preparing key-value pairs (the former for aggregation, the latter for analysis), and then combining that segregated information.

2 Part 1 - Word Counter

2.1 Documentation

The first part of this project is an introductory exercise on the *MapReduce* operations. The process begins by activating the virtual environment through the use of Vagrant. The next step is to facilitate access to the virtual environment using the command `vagrant ssh`. Then we navigate to the environment's shared folder `/vagrant`, where we will download DonQuixote.txt, a text file that will be used for demonstration purposes. Following that, it is necessary to copy the file from the virtual environment to a Docker container that has the *Hadoop* file system installed. Consequently, we choose the Docker `namenode` as an interface. An essential step is to create a directory inside the Hadoop fs with `docker exec namenode hdfs dfs -mkdir -p /user/hdfs/input` and then copy the file inside that. Then, we insert the file from the container's memory to Hadoop utilizing the `hdfs dfs -put <file> <destination>` format. To verify that operation, we can do so either through the console or through a browser by following the address localhost:9870 and navigating to the *Utilities* section. The copy inside the container's memory is no longer needed, thus it is removed. A crucial point to note is the fact that the *MapReduce* operation cannot overwrite older operations, thus we need to remove any output directory along with its files. Now the preparatory phase is over.

To carry out the *MapReduce* operation, two Java files are needed. The first is the `Driver.java`, which acts as a blueprint outlining the way this operation should run. It begins by importing the necessary packages to interface with the Hadoop API. Then, it specifies the workflow by activating a *Mapper* function, then a *Combiner*, and lastly a *Reducer*. The Combiner is essentially the same as the Reducer, which in a real-world scenario, would first reduce the locally mapped results and then pass them to the last Reducer for the final tally. Additionally, it sources the file that contains the detailed *Mapper-Reducer* functions. It assigns data types to the resulting key-value pairs. It also sets the path for data input and output. Lastly, it instructs the program to wait until the completion of the operation.

The second file is the `WordCount.java`, which explicitly defines the functions of *Map* and *Reduce*. The Mapper function splits the words of the book into tokens, which will later act as the *key*, and pads each one with the number 1 as the *value*. Subsequently, the keys are sorted in alphabetical order, and for each key-value pair, the values are summed. The result is a key-value pair that reflects the number of occurrences for each token in the book.

Using the provided template code, the only intervention needed is to specify the new input file as the `DonQuixote.txt`. Following that, the two `.java` files must be compiled into an executable format. Consequently, we change the directory to `cd /vagrant/hadoop-mapreduce-examples/`, where a `pom.xml` resides and is responsible for detailing the building parameters of the output `.jar` executable, and commence the process with `mvn clean install`. Once the executable file is built, similarly to the text file, it must be copied inside the `namenode` Docker container. The execution commences with `docker exec namenode hadoop jar <file.jar>` and once the operation is complete, the last 20 lines of the output file can be inspected with: `docker exec namenode hdfs dfs -text /user/hdfs/output/part-r-00000 | tail -20`.. The *MapReduce* console output can be seen in A1.

2.2 Walk-through

The preparatory steps for acquiring the file and inserting it into Hadoop are:

```
vagrant ssh
cd /vagrant
wget https://www.gutenberg.org/cache/epub/996/pg996.txt -O DonQuixote.txt
docker cp DonQuixote.txt namenode:/
```

```
docker exec namenode hdfs dfs -put DonQuixote.txt /user/hdfs/input/
docker exec namenode hdfs dfs -ls /user/hdfs/input/
docker exec namenode rm DonQuixote.txt
docker exec namenode hdfs dfs -rm -r /user/hdfs/output
```

The workflow for the MapReduce operation is:

```
FileInputFormat.addInputPath(job, new Path("/user/hdfs/input/DonQuixote.txt"));
cd /vagrant/hadoop-mapreduce-examples/
mvn clean install
docker cp /vagrant/hadoop-mapreduce-examples/target/hadoop-map-reduce-examples
↪ -1.0-SNAPSHOT-jar-with-dependencies.jar namenode:/
docker exec namenode hadoop jar
↪ /hadoop-map-reduce-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
docker exec namenode hdfs dfs -text /user/hdfs/output/part-r-00000 | tail -20
```

3 Part 2 - Spotify Analytics

3.1 Documentation

The second part of this project encompasses the creation of a new *MapReduce* job that takes as input a CSV file and produces a set of concise analytics. The input file includes details of Spotify songs, such as name, artist, the country that is being broadcast to, along with various metrics like tempo, liveness, and *danceability*, etc., that are being logged on a daily basis. The aim of this job is to examine each song of a country per month and find the most *danceable* one, along with the average *danceability* of that month.

At the outset, the provided `universal_top_spotify_songs.csv` is placed inside the folder that is shared between the local and virtual machine. Then, the file is copied inside the *Docker container* `namenode` that facilitates the utilization of the *HDFS* tool. By leveraging Hadoop's `hdfs dfs -put <file> <destination>` command, the file is inserted into the *HDFS*. Optionally, the .csv file is removed from the *Docker container*. Lastly, the output folder and its contents are preemptively deleted.

The pivotal part of this endeavor is the creation of the Java files that will facilitate the *MapReduce* job. Starting with the same `Driver.java` from the previous part, the input file is now specified as the `universal_top_spotify_songs.csv`, and the data types of key-value pairs are both set as `Text`. Next, a new `DanceabilityAnalysis.java` is created using the same structure as the previous `WordCount.java`, but the map and reduce functions are written from scratch.

Mapper The Map function receives each CSV line as a string, splits it into tokens, and keeps only the country, snapshot-date, song name, and *danceability*. It is important to note that the snapshot-date uses the long format date of `yyyy-MM-dd`, and since the desired outcome is an aggregation on the `yyyy-MM` level, the new variable `monthYear` is formatted accordingly. Again, for aggregation purposes, the key `countryMonth` is set as the combination of the country and the `monthYear`. Similarly, the remaining variables `songName` and *danceability* are packed into the value. As a result, the keys and values are both strings and are then passed to the reducer function.

Reducer In the first steps of the Reduce function, a set of variables are initialized to keep track of the most danceable song, its *danceability*, the cumulative *danceability*, and the number of entries for that key. Then, the imported value is split into song name and *danceability*, the *danceability* is converted from string to double, and then added to the cumulative *danceability*. An if statement checks whether the song under scrutiny is the most danceable one before the main loop is terminated. Subsequently, the average *danceability* is calculated by dividing the cumulative sum by the total number of values for this key and is then rounded to three decimal points. For future compatibility, the final result is formatted as CSV. To achieve this, the variables country, month, song name, *danceability*, and average *danceability* are strung together in the appropriate manner. Lastly, when specifying the context for output, the key is ignored because it is included in the formatted value string.

Execution Finally, the `pom.xml` inside the `cd /vagrant/hadoop-mapreduce-examples/` is utilized to build the executable jar, which is copied inside the `namenode` and then executed. The first 20 lines of the resulting operation can be seen in Figure 1 and the *MapReduce* console outputs in A2.

3.2 Walk-through

```
vagrant ssh
cd /vagrant
docker cp universal_top_spotify_songs.csv namenode:/
docker exec namenode hdfs dfs -put universal_top_spotify_songs.csv
↪  /user/hdfs/input/
docker exec namenode hdfs dfs -ls /user/hdfs/input/
docker exec namenode rm universal_top_spotify_songs.csv
docker exec namenode hdfs dfs -rm -r /user/hdfs/output
```

```
FileInputFormat.addInputPath(job, new
↪  Path("/user/hdfs/input/universal_top_spotify_songs.csv"));
```

```
cd /vagrant/hadoop-mapreduce-examples/
mvn clean install
docker cp
↪  /vagrant/hadoop-mapreduce-examples/target/hadoop-map-reduce-examples-1.0
↪  -SNAPSHOT-jar-with-dependencies.jar namenode:/
docker exec namenode hadoop jar
↪  /hadoop-map-reduce-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
```

```
    "","2023-10","PERRO NEGRO","0.911","0.669"
    "","2023-11","Lovin On Me","0.943","0.657"
    "","2023-12","Lovin On Me","0.943","0.632"
    "","2024-01","Lovin On Me","0.943","0.674"
    "AE","2023-10","I KNOW ?","0.927","0.657"
    "AE","2023-11","Lovin On Me","0.943","0.659"
    "AE","2023-12","Lovin On Me","0.943","0.630"
    "AE","2024-01","Lovin On Me","0.943","0.655"
    "AR","2023-10","PERRO NEGRO","0.911","0.749"
    "AR","2023-11","24/7 6.5","0.925","0.750"
    "AR","2023-12","PERRO NEGRO","0.911","0.740"
    "AR","2024-01","PERRO NEGRO","0.911","0.731"
    "AT","2023-10","Sprinter","0.916","0.688"
    "AT","2023-11","Lovin On Me","0.943","0.658"
    "AT","2023-12","Lovin On Me","0.943","0.591"
    "AT","2024-01","Lovin On Me","0.943","0.670"
    "AU","2023-10","Sprinter","0.916","0.642"
    "AU","2023-11","Lovin On Me","0.943","0.638"
    "AU","2023-12","Lovin On Me","0.943","0.615"
    "AU","2024-01","Lovin On Me","0.943","0.652"
```

Figure 1: Output of the 20 leading lines for the MapReduce job of part 2.

A Appendix - Outputs

Output of the *MapReduce* job for the Part 1:

```
    vagrant@vagrant:~$ docker exec namenode hadoop jar
    ↪  /hadoop-map-reduce-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
2024-02-02 21:50:04,216 INFO client.RMProxy: Connecting to ResourceManager at
↪  resourcemanager/172.18.0.3:8032
2024-02-02 21:50:04,765 INFO client.AHSProxy: Connecting to Application History
↪  server at historyserver/172.18.0.2:10200
2024-02-02 21:50:05,580 WARN mapreduce.JobResourceUploader: Hadoop command-line
↪  option parsing not performed. Implement the Tool interface and execute your
↪  application with ToolRunner to remedy this.
2024-02-02 21:50:05,621 INFO mapreduce.JobResourceUploader: Disabling Erasure
↪  Coding for path:
↪  /tmp/hadoop-yarn/staging/root/.staging/job_1706910195406_0001
2024-02-02 21:50:06,190 INFO sasl.SaslDataTransferClient: SASL encryption trust
↪  check: localHostTrusted = false, remoteHostTrusted = false
2024-02-02 21:50:07,272 INFO input.FileInputFormat: Total input files to process
↪  : 1
2024-02-02 21:50:07,454 INFO sasl.SaslDataTransferClient: SASL encryption trust
↪  check: localHostTrusted = false, remoteHostTrusted = false
2024-02-02 21:50:07,481 WARN hdfs.DataStreamer: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1252)
        at java.lang.Thread.join(Thread.java:1326)
        at
        ↪  org.apache.hadoop.hdfs.DataStreamer.closeResponder(DataStreamer.java:986)
        at org.apache.hadoop.hdfs.DataStreamer.endBlock(DataStreamer.java:640)
        at org.apache.hadoop.hdfs.DataStreamer.run(DataStreamer.java:810)
2024-02-02 21:50:07,518 INFO sasl.SaslDataTransferClient: SASL encryption trust
↪  check: localHostTrusted = false, remoteHostTrusted = false
2024-02-02 21:50:07,542 WARN hdfs.DataStreamer: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1252)
        at java.lang.Thread.join(Thread.java:1326)
        at
        ↪  org.apache.hadoop.hdfs.DataStreamer.closeResponder(DataStreamer.java:986)
        at org.apache.hadoop.hdfs.DataStreamer.endBlock(DataStreamer.java:640)
        at org.apache.hadoop.hdfs.DataStreamer.run(DataStreamer.java:810)
2024-02-02 21:50:07,544 INFO mapreduce.JobSubmitter: number of splits:1
2024-02-02 21:50:08,119 INFO sasl.SaslDataTransferClient: SASL encryption trust
↪  check: localHostTrusted = false, remoteHostTrusted = false
2024-02-02 21:50:08,218 INFO mapreduce.JobSubmitter: Submitting tokens for job:
↪  job_1706910195406_0001
2024-02-02 21:50:08,219 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-02-02 21:50:08,915 INFO conf.Configuration: resource-types.xml not found
2024-02-02 21:50:08,916 INFO resource.ResourceUtils: Unable to find
↪  'resource-types.xml'.
2024-02-02 21:50:10,063 INFO impl.YarnClientImpl: Submitted application
↪  application_1706910195406_0001
2024-02-02 21:50:10,383 INFO mapreduce.Job: The url to track the job:
↪  http://resourcemanager:8088/proxy/application_1706910195406_0001/
2024-02-02 21:50:10,392 INFO mapreduce.Job: Running job: job_1706910195406_0001
2024-02-02 21:50:39,326 INFO mapreduce.Job: Job job_1706910195406_0001 running in
↪  uber mode : false
2024-02-02 21:50:39,329 INFO mapreduce.Job:  map 0% reduce 0%
2024-02-02 21:50:54,744 INFO mapreduce.Job:  map 100% reduce 0%
2024-02-02 21:51:07,924 INFO mapreduce.Job:  map 100% reduce 100%
```

```
2024-02-02 21:51:08,955 INFO mapreduce.Job: Job job_1706910195406_0001 completed
↪  successfully
2024-02-02 21:51:09,221 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=155360
                FILE: Number of bytes written=769015
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=2391844
                HDFS: Number of bytes written=368129
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Rack-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=50084
                Total time spent by all reduces in occupied slots (ms)=80752
                Total time spent by all map tasks (ms)=12521
                Total time spent by all reduce tasks (ms)=10094
                Total vcore-milliseconds taken by all map tasks=12521
                Total vcore-milliseconds taken by all reduce tasks=10094
                Total megabyte-milliseconds taken by all map tasks=51286016
                Total megabyte-milliseconds taken by all reduce tasks=82690048
        Map-Reduce Framework
                Map input records=43285
                Map output records=437092
                Map output bytes=4095211
                Map output materialized bytes=155352
                Input split bytes=116
                Combine input records=437092
                Combine output records=33432
                Reduce input groups=33432
                Reduce shuffle bytes=155352
                Reduce input records=33432
                Reduce output records=33432
                Spilled Records=66864
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=425
                CPU time spent (ms)=6650
                Physical memory (bytes) snapshot=343990272
                Virtual memory (bytes) snapshot=13150445568
                Total committed heap usage (bytes)=230821888
                Peak Map Physical memory (bytes)=224038912
                Peak Map Virtual memory (bytes)=4955017216
                Peak Reduce Physical memory (bytes)=119951360
                Peak Reduce Virtual memory (bytes)=8195428352
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=2391728
```

```
        File Output Format Counters
                Bytes Written=368129
```

Output of the *MapReduce* job for the 2:

```
    vagrant@vagrant:~$ docker exec namenode hadoop jar
    ↪  /hadoop-map-reduce-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
2024-02-02 21:22:54,344 INFO client.RMProxy: Connecting to ResourceManager at
↪  resourcemanager/172.18.0.4:8032
2024-02-02 21:22:55,008 INFO client.AHSProxy: Connecting to Application History
↪  server at historyserver/172.18.0.6:10200
2024-02-02 21:22:55,728 WARN mapreduce.JobResourceUploader: Hadoop command-line
↪  option parsing not performed. Implement the Tool interface and execute your
↪  application with ToolRunner to remedy this.
2024-02-02 21:22:55,779 INFO mapreduce.JobResourceUploader: Disabling Erasure
↪  Coding for path:
↪  /tmp/hadoop-yarn/staging/root/.staging/job_1706908899701_0001
2024-02-02 21:22:56,374 INFO sasl.SaslDataTransferClient: SASL encryption trust
↪  check: localHostTrusted = false, remoteHostTrusted = false
2024-02-02 21:22:57,495 INFO input.FileInputFormat: Total input files to process
↪  : 1
2024-02-02 21:22:57,638 INFO sasl.SaslDataTransferClient: SASL encryption trust
↪  check: localHostTrusted = false, remoteHostTrusted = false
2024-02-02 21:22:57,700 INFO sasl.SaslDataTransferClient: SASL encryption trust
↪  check: localHostTrusted = false, remoteHostTrusted = false
2024-02-02 21:22:57,735 INFO mapreduce.JobSubmitter: number of splits:1
2024-02-02 21:22:58,324 INFO sasl.SaslDataTransferClient: SASL encryption trust
↪  check: localHostTrusted = false, remoteHostTrusted = false
2024-02-02 21:22:58,449 INFO mapreduce.JobSubmitter: Submitting tokens for job:
↪  job_1706908899701_0001
2024-02-02 21:22:58,450 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-02-02 21:22:59,215 INFO conf.Configuration: resource-types.xml not found
2024-02-02 21:22:59,216 INFO resource.ResourceUtils: Unable to find
↪  'resource-types.xml'.
2024-02-02 21:23:00,281 INFO impl.YarnClientImpl: Submitted application
↪  application_1706908899701_0001
2024-02-02 21:23:00,551 INFO mapreduce.Job: The url to track the job:
↪  http://resourcemanager:8088/proxy/application_1706908899701_0001/
2024-02-02 21:23:00,562 INFO mapreduce.Job: Running job: job_1706908899701_0001
2024-02-02 21:23:27,356 INFO mapreduce.Job: Job job_1706908899701_0001 running in
↪  uber mode : false
2024-02-02 21:23:27,366 INFO mapreduce.Job:  map 0% reduce 0%
2024-02-02 21:23:50,911 INFO mapreduce.Job:  map 5% reduce 0%
2024-02-02 21:23:56,961 INFO mapreduce.Job:  map 9% reduce 0%
2024-02-02 21:24:03,028 INFO mapreduce.Job:  map 13% reduce 0%
2024-02-02 21:24:09,095 INFO mapreduce.Job:  map 17% reduce 0%
2024-02-02 21:24:15,166 INFO mapreduce.Job:  map 21% reduce 0%
2024-02-02 21:24:21,225 INFO mapreduce.Job:  map 25% reduce 0%
2024-02-02 21:24:27,269 INFO mapreduce.Job:  map 29% reduce 0%
2024-02-02 21:24:33,343 INFO mapreduce.Job:  map 33% reduce 0%
2024-02-02 21:24:38,416 INFO mapreduce.Job:  map 37% reduce 0%
2024-02-02 21:24:44,458 INFO mapreduce.Job:  map 42% reduce 0%
2024-02-02 21:24:50,507 INFO mapreduce.Job:  map 46% reduce 0%
2024-02-02 21:24:56,605 INFO mapreduce.Job:  map 50% reduce 0%
2024-02-02 21:25:02,677 INFO mapreduce.Job:  map 54% reduce 0%
2024-02-02 21:25:08,733 INFO mapreduce.Job:  map 58% reduce 0%
2024-02-02 21:25:14,774 INFO mapreduce.Job:  map 62% reduce 0%
2024-02-02 21:25:20,816 INFO mapreduce.Job:  map 66% reduce 0%
2024-02-02 21:25:23,864 INFO mapreduce.Job:  map 100% reduce 0%
2024-02-02 21:25:41,017 INFO mapreduce.Job:  map 100% reduce 100%
2024-02-02 21:25:41,027 INFO mapreduce.Job: Job job_1706908899701_0001 completed
↪  successfully
```

```
2024-02-02 21:25:41,398 INFO mapreduce.Job: Counters: 54
        File System Counters
                FILE: Number of bytes read=942550
                FILE: Number of bytes written=2343069
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=85676844
                HDFS: Number of bytes written=13439
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Rack-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=451476
                Total time spent by all reduces in occupied slots (ms)=113904
                Total time spent by all map tasks (ms)=112869
                Total time spent by all reduce tasks (ms)=14238
                Total vcore-milliseconds taken by all map tasks=112869
                Total vcore-milliseconds taken by all reduce tasks=14238
                Total megabyte-milliseconds taken by all map tasks=462311424
                Total megabyte-milliseconds taken by all reduce tasks=116637696
        Map-Reduce Framework
                Map input records=360798
                Map output records=360797
                Map output bytes=12284242
                Map output materialized bytes=942542
                Input split bytes=133
                Combine input records=0
                Combine output records=0
                Reduce input groups=292
                Reduce shuffle bytes=942542
                Reduce input records=360797
                Reduce output records=292
                Spilled Records=721594
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=911
                CPU time spent (ms)=103150
                Physical memory (bytes) snapshot=389505024
                Virtual memory (bytes) snapshot=13152555008
                Total committed heap usage (bytes)=230821888
                Peak Map Physical memory (bytes)=237195264
                Peak Map Virtual memory (bytes)=4957118464
                Peak Reduce Physical memory (bytes)=153878528
                Peak Reduce Virtual memory (bytes)=8195436544
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=85676711
        File Output Format Counters
                Bytes Written=13439
```

B Appendix - Part 2 Java Files

The *Driver.java* file:

```java
package gr.aueb.panagiotisl.mapreduce.wordcount;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Driver {
    public static void main(String[] args) throws Exception {

        System.setProperty("hadoop.home.dir", "/");

        // Instantiate a configuration
        Configuration configuration = new Configuration();

        // Instantiate a job
        Job job = Job.getInstance(configuration, "Danceability Analysis");

        // Set job parameters
        job.setJarByClass(DanceabilityAnalysis.class); // Point to the
        ↪  map-reduce class in the other file that declares the functions
        job.setMapperClass(DanceabilityAnalysis.DanceabilityMapper.class); //
        ↪  Specify which is the mapper
        job.setReducerClass(DanceabilityAnalysis.DanceabilityReducer.class); //
        ↪  Specify which is the reducer

        // Set the output key and value classes to match the Reducer's output
        ↪  types
        job.setOutputKeyClass(Text.class); // Set the data type for the key
        job.setOutputValueClass(Text.class); // Set the data type for the value

        // set io paths
        FileInputFormat.addInputPath(job, new
        ↪  Path("/user/hdfs/input/universal_top_spotify_songs.csv")); // to
        ↪  arxeio input
        FileOutputFormat.setOutputPath(job, new Path("/user/hdfs/output/")); //
        ↪  Where to store the results

        // Wait and Exit after job completion
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

The *DanceabilityAnalysis.java* file:

```java
package gr.aueb.panagiotisl.mapreduce.wordcount;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
```

```
import javax.naming.Context;

public class DanceabilityAnalysis {

    public static class DanceabilityMapper extends Mapper<LongWritable, Text,
    ↪  Text, Text> {
        private Text countryMonth = new Text(); // variable to store the country
        ↪  and parsed snapshot date --> key
        private Text songData = new Text(); // variable to store the details
        ↪  about this song --> value

        @Override
        public void map(LongWritable key, Text value, Context context) throws
        ↪  IOException, InterruptedException {
            // Split the CSV line considering commas inside quotes
            String[] tokens =
            ↪  value.toString().split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)", -1);

            try {
                // For each song: Keep the country, date, name of the song, its
                ↪  danceability
                String country = tokens[6].replaceAll("\"", ""); // Remove the
                ↪  quotes, ex: "GR" -> GR
                String date = tokens[7].replaceAll("\"", "");
                String songName = tokens[1].replaceAll("\"", "");
                String danceability = tokens[13].replaceAll("\"", "");

                // Format the date from "yyyy-MM-dd" to "YYYY-MM" for
                ↪  aggregation purposes
                SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
                Date snapshotDate = sdf.parse(date);
                sdf.applyPattern("yyyy-MM");
                String monthYear = sdf.format(snapshotDate);

                // Set the key and value pair
                countryMonth.set(country + ":" + monthYear); // This is the key,
                ↪  ex: "MX:2024-01"
                // Set "~" as separator because there are song titles containing
                ↪  ":"
                songData.set(songName + "~" + danceability); // This is the
                ↪  value, ex: "Takata~ 0.92"

                context.write(countryMonth, songData); //The key-value pair to
                ↪  be passed to reduce function
            } catch (Exception e) {
                // Just in case
            }
        }
    }

    public static class DanceabilityReducer extends Reducer<Text, Text, Text,
    ↪  Text> {
        @Override
        public void reduce(Text key, Iterable<Text> values, Context context)
        ↪  throws IOException, InterruptedException {
            // Assume the key right now is "MX:2024-01"
            double maxDanceability = 0; // Store the max danceability for this
            ↪  key
            String mostDanceableSong = ""; // Store the name of the most
            ↪  danceable song
```

```
            double totalDanceability = 0; // Keep the sum of the danceabilities
            ↪ for this key
            int count = 0; // Keep the total number of songs, for the
            ↪ calculation of AVERAGE

            for (Text val : values) { // For each value, ex: "Takata~ 0.92"
                String[] songData = val.toString().split("~"); // Split value
                ↪ into: [Takata, 0.92]
                try {
                    double danceability = Double.parseDouble(songData[1]); //
                    ↪ keep the 0.92 as the danceability
                    totalDanceability += danceability; // Add it to the total
                    ↪ danceability for this month
                    count++; // The increase the denominator

                    // If this song is more danceable than the most danceable,
                    ↪ make this the most danceable song
                    if (danceability > maxDanceability) {
                        maxDanceability = danceability;
                        mostDanceableSong = songData[0];
                    }
                } catch (NumberFormatException e) {
                    // Just in case
                }
            }

            // Calculate AVERAGE danceability for the month
            if (count > 0) { // To catch zero divisions while calculating the
            ↪ average
                double avgDanceability = totalDanceability / count;
                // Round the average danceability to 3 decimal places
                avgDanceability = Math.round(avgDanceability * 1000.0) / 1000.0;

                // Prepare the output in CSV format
                String result =
                ↪ String.format("\"%s\",\"%s\",\"%s\",\"%.3f\",\"%.3f\"",
                // Split the key MX:2024-01 to [MS, 2024-01]
                key.toString().split(":")[0], // Country
                key.toString().split(":")[1], // Date
                mostDanceableSong, // Most danceable song
                maxDanceability, // Max danceability, rounded in the string
                ↪ format
                avgDanceability); // Average danceability, rounded in the string
                ↪ format

                // KEY: Instead of passing the key in the context, i pass it in
                ↪ the value to have more freedom for the formating
                context.write(null, new Text(result)); // Write the result with
                ↪ null as key
            }
        }
    }

}
```