

Large Scale Data Management  
Athens University of Economics and Business  
M.Sc. in Data Science

*Programming Project I*  
Today's Date: January 29<sup>th</sup>, 2024  
Due Date: February 16<sup>th</sup>, 2024

## PREAMBLE

In this project, you will use the **Hadoop Map-Reduce** framework. First you will build and submit for execution a map-reduce application using a local hadoop deployment. Then, you will develop your own map-reduce application that provides an analysis of spotify songs.

## PART I:

In this part you will familiarize yourself with building and submitting map-reduce applications for execution. Using the Vagrantfile provided with this project you will initialize a virtual machine that offers you access to Java, maven, and Hadoop. You will use Java and maven to build your application and Hadoop to execute it. To initialize your virtual machine you must have vagrant and virtualbox (or some other virtualization software) installed. If you do, you can initialize your virtual machine by executing the following command in your terminal:

```
$ vagrant up
```

You have to make sure that the execution of this command was successful, which means that it went through all of its steps, including docker installation, software package updates, and maven build. To verify that, you should connect to the virtual machine through ssh and check whether the docker containers are up and running. To do so, execute the following commands and see if your output looks like the one below:

```
$ vagrant ssh
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
151d229e49ba	bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8	"/entrypoint.sh /run..."	6 minutes ago	Up 6 minutes (healthy)	0.0.0.0:9864->9864/tcp, :::9864->9864/tcp
0cf9e9cb0680	bde2020/hadoop-historyserver:2.0.0-hadoop3.2.1-java8	"/entrypoint.sh /run..."	6 minutes ago	Up 6 minutes (healthy)	8188/tcp
be2cbe31437b	bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8	"/entrypoint.sh /run..."	6 minutes ago	Up 6 minutes (healthy)	8042/tcp
b72fdbf2aadf	bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8	"/entrypoint.sh /run..."	6 minutes ago	Up 6 minutes (healthy)	0.0.0.0:9000->9000/tcp, :::9000->9000/tcp
c4ba94e45cbb	bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8	"/entrypoint.sh /run..."	6 minutes ago	Up 6 minutes (healthy)	0.0.0.0:8088->8088/tcp, :::8088->8088/tcp

If not, you can try deleting the virtual machine and re-executing the above command, or execute the following that will re-attempt to initialize your virtual machine:

```
$ vagrant provision
```

To build the application change directory and execute the following:

```
$ cd /vagrant/hadoop-mapreduce-examples/
$ mvn clean install
```

The executable of your application can then be found in ‘/vagrant/hadoop-mapreduce-examples/target/’. You should copy the application inside a docker container, and submit it for execution.

```
$ docker cp /vagrant/hadoop-mapreduce-examples/target/hadoop-map-reduce-examples-1.0-SNAPSHOT-jar-with-dependencies.jar namenode:/
Successfully copied 24.4MB to namenode:/
$ docker exec namenode hadoop jar /hadoop-map-reduce-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
```

If you receive an error about the input file / path not existing, you can add it yourself by executing the following commands, and then submitting the application again:

```
$ wget https://www.gutenberg.org/files/2701/2701-0.txt -O MobyDick.txt
$ docker cp MobyDick.txt namenode:/
$ docker exec namenode hdfs dfs -mkdir -p /user/hdfs/input
$ docker exec namenode hdfs dfs -put MobyDick.txt /user/hdfs/input/
```

You can observe the progress of the execution in your terminal, as well as through a web-ui accessible in <http://localhost:8088>. After the job has finished you can see the first 100 lines of the results by executing:

```
docker exec namenode hdfs dfs -text /user/hdfs/output/part-r-00000 | head -100
```

To successfully complete the first part of this project you have to provide the output of the execution of this application (terminal logs) for an input file of your choice. You should include in your report details about the file you used (URL where it can be found) and all the execution logs of the application as they were printed in your terminal.

## PART II:

For the second part of this project you will develop your own map-reduce application. You will first download and go through your input file, with statistics about spotify songs: [https://auebgr-my.sharepoint.com/:u:/g/personal/panagiotisliakos\\_aueb\\_gr/ERwr7Zgn5EVMtp1\\_I80D5vYBfzv1GqegDY-Ej4rI77Ijjg](https://auebgr-my.sharepoint.com/:u:/g/personal/panagiotisliakos_aueb_gr/ERwr7Zgn5EVMtp1_I80D5vYBfzv1GqegDY-Ej4rI77Ijjg)

This is a compressed .csv file and you are going to use only some of its columns in your project. The .csv has a header (first line) which will help you identify the information you need.

Your job is to produce a result file that provides for every country and month pair (for example: “GR: 2024-01”) the most “danceable” song, along with its “danceability” and the average danceability of all songs for the same country and month pair (for example: “Tacata (Remix): 0.922, avg: 0.454).

While developing your solution keep in mind the following:

- You have to come up with appropriate data types for your intermediate and final results and use them in your classes.
- A combiner function should be used if your combiner function is commutative and associative. Otherwise, it should be disabled.
- Your application should ignore the .csv header.
- Csv files use ‘,’ as a separator. However when this character is used inside double quotes it should be escaped. The following one-liner<sup>1</sup> will help you obtain the fields of a line:

---

<sup>1</sup><https://www.baeldung.com/java-split-string-commas>

```
String[] tokens = input.split("(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)", -1);
```

To successfully complete the second part of this project you have to provide the code of your application, i.e., two Java files for the Driver and Map/Reduce classes. You should provide comments in your map and reduce functions so that their operations are crystal-clear.

#### COOPERATION:

You do not have the option of forming a team in this project. However, discussions on most aspects of the project in the classroom and the classroom's forum are encouraged.

#### REPORTING:

The final *typed* project report (brief report) must consist of:

1. Details about the input files that you've used for the first part.
2. The (successful) execution output of the first part.
3. Two Java classes for your application for the second part.