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

Programming Project I
Today's Date: January 27^{th}, 2025
Due Date: February 14^{th}, 2025

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 car sales.

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 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 following:

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

```
CONTAINER ID   IMAGE                                                COMMAND                  CREATED         STATUS                   PORTS
928b3d6a24fb   bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8   "/entrypoint.sh /run..."   4 minutes ago   Up 4 minutes (healthy)   0.0.0.0:8088->8088/tcp, :::8088->80
4f58e4558ce0   bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8          "/entrypoint.sh /run..."   4 minutes ago   Up 4 minutes (healthy)   0.0.0.0:9864->9864/tcp, :::9864->98
6057b52e4ed2   bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8       "/entrypoint.sh /run..."   4 minutes ago   Up 4 minutes (healthy)   8042/tcp
cfd752d79d71   bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8          "/entrypoint.sh /run..."   4 minutes ago   Up 4 minutes (healthy)   0.0.0.0:9000->9000/tcp, :::9000->90
f21482916f17   bde2020/hadoop-historyserver:2.0.0-hadoop3.2.1-java8     "/entrypoint.sh /run..."   4 minutes ago   Up 4 minutes (healthy)   8188/tcp
```

If not, you can try deleting the virtual machine and re-executing the above command:

```
$ vagrant halt
$ vagrant destroy
```

or execute the following that will re-attempt to initialize your virtual machine:

```
$ vagrant provision
```

To build the application, change the 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 above command for an input file of your choice (other than MobyDick.txt). 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.

Keep in mind that you cannot use the same output location for multiple executions, so you should delete any temporary results.

```
$ docker exec namenode hdfs dfs -rm -r /user/hdfs/output/
```

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 records of car sales: `https://auebgr-my.sharepoint.com/:x:/g/personal/panagiotisliakos_aueb_gr/EXHUKfvONytBvZrjA3HbzzYBjihX_K4LQNhKMB1O-c3qHA?e=fKpjbK`

This is a .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 seller and month pair (for example: "kia motors america inc:2024-12") the car with the largest (sellingprice - mmr) difference, along with this "difference" and the average difference of all cars for the same seller and month pair (for example: "Kia Sorento LX: 1000, avg: 250.46).

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, the combiner function should be disabled.
- Your application should ignore the .csv header.

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.

SLOW MAVEN BUILD:
If the compilation process is taking too long you can consider using an alternative way to build your project without Maven, as described in this link: `https://auebgr-my.sharepoint.com/:u:/g/personal/panagiotisliakos_aueb_gr/EUhyiPxcuVJPhvH9rXEouI0BXC18ykwPNKsPZttMkbfaiw?e=GKXA1E`

USING DOCKER DIRECTLY

If you have trouble setting up the virtual machine, you could instead use docker[1] directly to set up the Hadoop cluster.

You should use the programming assignment files that are inside the following folder `project-1/map-reduce/docker-`
Using the docker terminal change directory (cd) to this folder and execute:

```
$ docker-compose up -d
```

This will create the necessary containers. Then you can use the following commands:

```
// make sure you are in the folder that has hadoop-mapreduce-examples
$ docker cp ./hadoop-mapreduce-examples/target/hadoop-map-reduce-examples-1.0-SNAPSHOT-jar-with-dependencies.jar namenode:/
// this will execute the jar file in the hadoop cluster
$ docker exec namenode hadoop jar /hadoop-map-reduce-examples-1.0-SNAPSHOT-jar-with-dependencies.jar
// this will create the input path
$ docker exec namenode hdfs dfs -mkdir -p /user/hdfs/input
// this will output all the lines of the output file
$ docker exec namenode hdfs dfs -text /user/hdfs/output/part-r-00000
// this will delete the output directory
$ docker exec namenode hdfs dfs -rm -r /user/hdfs/output/
```

[1] `https://docs.docker.com/desktop/setup/install/windows-install/`

You should also use a Java IDE such as IntelliJ IDEA (`https://www.jetbrains.com/idea/download/`) to compile the jar file. You should open the Java Project with this IDE, and using the maven module (m symbol on the right side panel) you should create the jar file by executing the install directive. You should use Java 1.8 for this project.