Lab 5

The objective of this laboratory is to start playing around with Apache Spark. We provide you a template in Java, from which you can write your first application.

The workflow for building and executing an Apache Spark application is the same as the one we used with Hadoop: once unzipped the template, import and open the project in your IDE (Eclipse) by using the option “import maven project”, modify it, and then export a jar associated with your application. Then, submit this job by means of spark-submit either locally (i.e., on your PC if you installed Spark on your PC):

```
spark-submit  --class it.polito.bigdata.spark.example.SparkDriver
--deploy-mode client --master local SparkProject.jar arguments
```

or on the cluster (you must execute spark-submit inside a terminal of the jupyter.polito.it gateway to submit your application on the cluster):

```
spark-submit  --class it.polito.bigdata.spark.example.SparkDriver
--deploy-mode client --master yarn SparkProject.jar arguments
```

Notice how in the template we did not, even if possible, configure the Spark master (we did not use the setMaster(…) method) and other settings explicitly in the main class of the driver. In this way, you can always submit the same jar on any configuration, either Spark standalone locally or a Yarn cluster. The main advantage of this method is for testing your application: first you can try your application on your machine, on a sample file, and then safely submit the same application for the real work.

You can also run Spark applications locally inside Eclipse, without installing Spark. In that case you must explicitly set the Spark master in your application by invoking the method setMaster(“local”) on the SpakConf object in the main method of your application.

```
.......
public static void main(String[] args) {
     SparkConf conf = new SparkConf()
                          .setAppName("Application name")
                          .setMaster("local");
     ........
```

1. Problem specification

If you completed Lab 1, you should now have (at least one) files with the word frequencies in the Amazon food reviews, in the format `word\tfreq`, where *freq* is an integer (a copy of the output of Lab 1 is available in the HDFS shared folder /data/students/bigdata-01QYD/Lab2/). Your task is to write a **Spark** application to filter these results, analyze the filtered data and compute some statistics on them.

Task 1

The first filter you should implement in your application is the following:

- Keep only the lines of the input data containing words that start with the prefix “ho”

The returned RDD contains the set of lines (`word\tfreq`) that satisfy the filtering operation.
Print on the standard output of the driver the following statistics, based on the content of the RDD returned by the filtering operation:

- The number of selected lines
- The maximum frequency (*maxfreq*) among the ones of the selected lines (i.e., the maximum value of *freq* in the lines obtained by applying the filter).

The application has three arguments: (1) input folder, (2) output folder, and (3) prefix

Task 2

Extend the previous application to further filter the analyzed data. Specifically, in the second part of your application, among the lines selected by the first filter, you have to apply another filter to select only the most frequent words. Specifically, your application must select those lines that contain words with a frequency (*freq*) greater than 80% of the maximum frequency (*maxfreq*) computed before.

Hence, implement the following filter:

- Keep only the lines with a frequency *freq* greater than 0.8**maxfreq*.

Finally, perform the following operations on the selected lines (the ones selected by applying both filters):

- Count the number of selected lines and print this number on the standard output
- Save the selected words (without frequency) in an output folder (one word per line)

2. Testing the application

1. Run you application locally on your PC inside Eclipse (if you are using Windows you must install winutils.exe and set the same windows environmental variables you set also for running MapReduce applications) and select only the lines containing the words starting with “ho” from the local sample file SampleLocalFile.csv

   Analyze the content of the local output folder.

2. Run you application on the cluster by using the command based on spark-submit. Also in this case, select only the lines containing the words starting with "ho" from the content of the input HDFS folder /data/students/bigdata-01QYD/Lab2/

   Steps to submit a Spark application on the cluster by using spark-submit:
   - Open a browser and connect to jupyter.polito.it
   - Copy the jar file associated with your application from your PC to the local file system of the gateway (use the drag and drop approach you already used in the previous labs)
   - Open a Terminal on the gateway jupyter.polito.it
   - Execute the following command inside the Terminal

```
spark-submit  --class it.polito.bigdata.spark.example.SparkDriver
              --deploy-mode client --master yarn
              SparkProject.jar
              /data/students/bigdata-01QYD/Lab2/ HDFSOutputFolder "ho"
```

Analyze the content of the output HDFS folder.

How to access logs files

If you are connecting from outside Polito you can proceed as follows to retrieve the log files from the command line:

1. Open a Terminal on the gateway jupyter.polito.it
2. Execute the following command in the Terminal:
   yarn logs -applicationId application_1521819176307_2195

   The last parameter is the application/job ID. You can retrieve the job ID of your application on the HUE interface: https://bigdatalab.polito.it/hue/jobbrowser#!jobs