UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Big Data Management (BDMA & MIRI Masters) Session: Spark

Alberto Abelló & Sergi Nadal

Instructions

In this session, as Spark jobs will run locally, you do not need to start the Spark service and everything can be run from eclipse. You can also dismiss the virtual machine and directly use the eclipse version provided in the lab session computers (for Linux).

Once you have fetched the code, it can be imported as a *Maven Project* into eclipse. All executions can be run locally, and thus you do not need to be concerned with the host connection. In the *Main* class you can see that different arguments are expected, and will drive the behavior of the program. The simplest way to define such arguments is by creating a *Run Configuration* of type *Java Application* from the *Run* menu. In the tab menu called *Arguments* you will be able to edit such arguments, in each exercise you will be guided with the required arguments.

Delivery

By the end of the session hand in your answers. Code must be delivered via LearnSQL. Prepare a ZIP file and upload it to LearnSQL with the following content: `Exercise_1.java`, `Exercise_2.java`.

Exercise 1: Analyzing banking data with Spark (40 % weight)

The first exercise consists on exploring a dataset with direct marketing campaigns (phone calls) of a Portuguese banking institution[1]. The file, which is a is a comma-separated (CSV) file, is already provided in the project's *resources* folder with the name `bank.csv`. You can check its description, together with the attributes it contains and their types, in the file `bank-names.txt`.

We are interested in analyzing whether the type of job and having or not a personal loan affects the account balance. Thus, we precisely ask you to implement in class `Exercise_1.java` the computation of *average balance of personal loan holders and non-personal loan holders per job type, ordered by job type name*. You should obtain the following output:

```
"admin.": avg balance with loan = 861.0769230769231, without loan = 1312.71834625323
"blue-collar": avg balance with loan = 799.2115384615385, without loan = 1141.6278481012657
"entrepreneur": avg balance with loan = 1143.2682926829268, without loan = 1807.1417322834645
"housemaid": avg balance with loan = 2144.076923076923, without loan = 2075.8888888888887
"retired": avg balance with loan = 1171.6875, without loan = 2504.6464646464647
"self-employed": avg balance with loan = 792.1, without loan = 1510.1176470588234
"services": avg balance with loan = 628.027027027027, without loan = 1206.6355685131196
"student": avg balance with loan = 1161.0, without loan = 1548.433734939759
"technician": avg balance with loan = 799.8403361344538, without loan = 1428.3882896764253
"unemployed": avg balance with loan = 537.6153846153846, without loan = 1151.8
"unknown": avg balance with loan = 341.0, without loan = 1533.081081081081
```

[1] You can learn more about this dataset in `https://archive.ics.uci.edu/ml/datasets/Bank+Marketing`.

Exercise 2: Optimization of an Spark program (60 % weight)

In this exercise we will analyze data related to the *World Happiness Report* dataset from Kaggle (`https://www.kaggle.com/unsdsn/world-happiness`). It consists of a set of CSV files (one per year) that rank 155 countries by their happiness levels according to metrics about economic production, social support, etc. We will focus on three files, related to the years 2015, 2016 and 2017. Each file provides the ranking of that year, however we are interested on obtaining *the top K European richest countries according to their GDP per capita for 2015, 2016 and 2017.*

Below we provide you with an Spark program that performs such analysis, however it suffers from several performance flaws due to a bad design. First, analyze the following code (also available in class *Exercise_2.java*) and detect such issues:

```
JavaRDD<String> report_2015 = spark.textFile("src/main/resources/2015.csv");
JavaRDD<String> report_2016 = spark.textFile("src/main/resources/2016.csv");
JavaRDD<String> report_2017 = spark.textFile("src/main/resources/2017.csv");

JavaRDD<String> unified = report_2015.union(report_2016).union(report_2017).repartition(1);

List<Tuple2<Double,String>> ranking = unified.flatMap(t -> {
    if (t.contains("Country")) return new ArrayList<String>().iterator();
    else return Lists.newArrayList(t).iterator();
  })
  .mapToPair(t -> new Tuple2<>(Double.parseDouble(t.split(",")[5]),t))
  .groupByKey()
  .sortByKey(false)
  .mapToPair(t -> new Tuple2<>(t._1, t._2.iterator().next()))
  .mapValues(t -> t.contains("Europe") ? t : null)
  .zipWithIndex() //Adds the position of each element in the RDD as a Long
  .collect();

List<String> topK = Lists.newArrayList();
int i = 0;
while (i < ranking.size() && topK.size() < K) {
  if (ranking.get(i) != null) {
    String country = ranking.get(i)._1._2.split(",")[0];
    if (!topK.contains(country)) {
      topK.add(country);
    }
  }
  ++i;
}
out = topK.stream().collect(Collectors.joining("\n"));
```

Your task consists on proposing an improved version of this program such that it optimizes its performance and applies the right set of operations. You are free to modify the provided code as long as the output is exactly the same. You are not allowed to use any external library. Optimize your program with the goal of executing it in a distributed cluster, however bear in mind that you are executing it in local mode.

You can replace the current code in *Exercise_2.java* with your proposal. The rationale behind your decisions should also be discussed in the provided answer sheet, clearly state any required assumptions. Considering that the input files are very small, and in order to test the performance improvements, we provide you with a larger version (e.g., *2015_long.csv*) of each. You can download the files from `https://bit.ly/2FFLmab`.