UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
School of Professional & Executive Development

Big Data Management and Analytics
Session: Apache Spark

Lecturer: Petar Jovanovic and Sergi Nadal

November 18th, 2016

In this session we will install a Spark cluster and do some exercices with its core API. Then, we will learn how to use SparkSQL to further analyse data in a relational manner.

It is highly advisable to complete the tasks listed in section 2 prior to attending the class, in order to get full advantage of the available in-class time.

1 Required Tools

- `eclipse` IDE with Java 8 and Maven plugin installed
- The following Java libraries (already included in the provided `pom.xml`):
  - `Spark Core`
  - `SparkSQL`

2 Tasks To Do Before The Session

- Read the slides. It is not necessary to read the examples as they will be discussed in class.
- Checkout the `Spark Core` programming guide (`http://spark.apache.org/docs/latest/programming-guide.html`) and the `SparkSQL` programming guide (`https://spark.apache.org/docs/latest/sql-programming-guide.html`).
- Get familiar with the operations in Spark, for instance you should be able to answer the following questions:
  - What is the difference between `map` and `flatMap`?
  - Why some methods have a `toPair` version? For instance, `map` or `mapToPair`.

- Am I only limited to work with `JavaRDD<T>` or `JavaPairRDD<K,V>`? What are the options if I want RDDs of more complex structures?
- What does the operation `collect`? What is its impact?

3 Part A: Examples & Questions (30min)

During the first 30 minutes of session we will go through the proposed examples in the slides. After each one, questions related to that particular topic will be addressed.

4 Part B: In-class Practice (2h30min)

4.1 Exercise 1 (30 min): Setting up the cluster and testing

Follow the document enclosed to this exercise sheet to set up an Spark cluster.

In order to test if Spark is properly running, let's run a very simple example, precisely we will interact with HDFS where the program will:

1. Read a local file
2. Compute the word count on the local file
3. Write the local file to HDFS
4. Read the file back from HDFS
5. Compute the word count on the file using Spark
6. Compare the word count results

First, start Hadoop and then run the following command. Note the two parameters we are passing are: 1) the local file to use, and 2) the HDFS directory for reading/writing.

```
cd /home/bdma**/spark-2.0.1
./bin/spark-submit --class org.apache.spark.examples.
    DFSReadWriteTest --master spark://master:7077 --deploy-mode
    client examples/jars/spark-examples_2.11-2.0.1.jar README.md
    hdfs://master:27000/user/bdma**/
```

You can track its execution in Spark's web UI. If everything has been properly installed, you should have a similar output like this one (discading operational logs):

```
Success! Local Word Count (461) and DFS Word Count (461) agree.
```

Finally, note that if you execute the program again it will fail as the output file already exists in HDFS. You can remove the results from HDFS by executing the following command:

```
~/hadoop-2.5.1/bin/hdfs dfs -rm -R /user/bdma**/dfs_read_write_test
```

Let the lecturer know when this is finished.

4.2 Exercise 2 (30 min): Your first Spark program

In this exercise we will start programming in Spark. In order to do so, get the provided Java project and import it with Maven as usual. After the Maven import process, if compilation errors still appear, you may right-click in the project and select `Maven` > `Update project`.

After this is done, you should be able to see the following packages:

- **(default package)**: with the `main` method.
- **exercises (2, 3, 4, 5)**: with the required classes per exercise to complete.

To start, let's take a look at class `Exercise_2_warmup.java`, you do not need to implement anything here, as the code is already provided for you. Such method provides basic analysis on top of the `wines` file we used in previous sessions. Note that, for simplicity, a sample of 10MB of such file is provided in the *resources* folder. When you understand everything, and manage to execute it[1], you may proceed to complete `Exercise_2.java`.

In `Exercise_2.java`, you are supposed to implement the group by and aggregation functions on top of the same `wines` file. You may follow a similar approach to that in the MapReduce session. Precisely, you have to return the sum of attribute *ash* (position 5) grouped by *type* (position 1). For extra scoring, note that the `Reduce` step here does not follow the same approach, which operation can you use that follows exactly the same implementation?

For simplicity in the development (in order to be able to test your code locally in eclipse), you can use the provided `wines.10m.txt` as before.

4.3 Exercise 3 (30 min): Cluster mode

In the previous exercise we programmed a local implementation of the `groupByAndAggCluster` method. However, what we want now is to leverage the cluster installation we performed before. Thus, in this exercise it is proposed that you implement the same method as before but now in cluster mode.

You should complete the method provided in class `Exercise_3.java`, beware that such function contains two parameters (input and output files)

[1]Note that the main method requires as argument the exercise to execute, thus you should modify the "Run Configuration" to pass the required "Program arguments".

given that it is no more possible to write the output using stdout. As input file, you should use a complete version of `wines.txt`, please, refer to Hadoop's session if you deleted it from your HDFS, and generate a new version again.

Once you have implemented all necessary changes required for the code to work in cluster mode, you should do the following:

- Package a JAR file and upload it to your *master* node.
- Execute your program using the `spark-submit` command in a similar manner as Exercise 1. Note that, now you should provide two parameters to the execution with the in and out paths respectively. Furthermore, do not forget to launch the program with `--deploy-mode` set to *cluster*.
- In order to display the output, use the `hdfs dfs -getmerge` command in a similar manner as done in exercise 2 in the MapReduce session.

4.4 Exercise 4 (1 h) Querying structured data with SparkSQL

Spark SQL provides a special type of RDD called DataFrame. A DataFrame is an RDD of Row objects, each representing a record. A DataFrame also knows the schema (i.e., data fields) of its rows. While DataFrame look like regular RDDs, internally they store data in a more efficient manner, taking advantage of their schema. In addition, they provide new operations not available on RDDs, such as the ability to run SQL queries. DataFrames can be created from external data sources, from the results of queries, or from regular RDDs. You might find further details in the SparkSQL programming guide: `http://spark.apache.org/docs/latest/sql-programming-guide.html`

In this exercise we ask you to implement queries on top of the `wines.txt` file. In order to do so, first you are required to implement the class with the schema. Precisely, the file contains the following attributes:

- `type`
- `region`
- `alc`
- `m_acid`
- `ash`
- `alc_ash`
- `mgn`

- `t_phenols`
- `flav`
- `nonflav_phenols`
- `proant`
- `col`
- `hue`
- `od280od315`
- `proline`

In `http://spark.apache.org/docs/latest/sql-programming-guide.html#programmatically-specifying-the-schema` you have a full example on how to generate the schema for the file. We ask you to implement the following queries using SparkSQL:

- Number of regions that have *alc* greater than 11 grouped by *type*.
- Get descriptive statistics (count, mean, stddev, min and max) of the column *proline*. (You might get some inspiration from `https://databricks.com/blog/2015/06/02/statistical-and-mathematical-functions-with-dataframes-in-spark.html`

5 Part C: Optional Practice (3h)

5.1 Exercise 5: Combining SparkSQL with RDDs

In this exercise you are provided a local list of regions, which are the labels for the `region` attribute in the wines file. As before, you should load the file in a SparkSQL table in order to be able to issue SQL queries, however now you are required to provide results for the following query:

- Region name and average `hue` grouped by region.

For simplicity, you can apply to the region id a modulo 50 in order to get results for all lines in the file. Note that, in the file there are some very large ids like 900.