Lab 5 – Apache Spark

CC5212-1 – April 21, 2021

Today we're going to analyse TV shows and to get (1) their average rating and (2) a list of the best rated episodes. We will use data from IMDb (the same as Lab 4), which look like:

```
0000000016       181740       9.4       The Wire       2002       null       TV_SERIES       null
0000000017         1855       9.6       The Wire       2002       null       TV_SERIES       -30- (#5.10)
0000000125         1098       9.2       The Wire       2002       null       TV_SERIES       A New Day (#4.11)
0000000223         1166       8.7       The Wire       2002       null       TV_SERIES       All Due Respect (#3.2)
0.00000124         1304       9.0       The Wire       2002       null       TV_SERIES       All Prologue (#2.6)
...
```

The columns are: (1) a vector indicating distribution of votes, (2) the number of votes, (3) the average vote, (4) the series name, (5) the series year, (6) a disambiguator we will ignore, (7) the type, (8) the name of the episode. Where column (8) is null, the voting information is the overall for the series (on IMDb, one can vote on the series page or on an individual episode). The data are on the HDFS storage of the cluster: `hdfs://cm:9000/uhadoop/shared/imdb/imdb-ratings.tsv`. What we want to produce is:

```
The Wire#2002       -30- (#5.10)|Final Grades (#4.13)|Middle Ground (#3.11)       9.6       8.835
```

Where the columns are: (1) a key for the series (name#year), (2) a list of episodes tied for best rating using '`|`' as a delimiter, (3) the best rating of an episode (and thus the ratings of episodes in (2)), (4) the average rating of all episodes (excluding the series value, e.g., the first line in the input above).[1]

We will use Spark for processing. There are multiple options with which to run Spark, including Java 8, Python, Scala and R. The server we will use can run Java 8, Scala and Python. You can submit the work in Java 8, Python or Scala.[2] However, this document will focus on Java 8.

- Download the code project from u-cursos and open it in Eclipse. In the `src` folder you will find a few utility classes and examples to get you started. Have a look first at `WordCountTask`; we will go through this in class. More interesting is `AverageSeriesRating`, which actually gives us the first part of the task we need: computing the average rating for the series. The main goal of the lab will be to copy and extend this class and compute the information about the best episode. These examples use lambdas from Java 8.
  - Some other examples are given in the folder `ref` without lambdas using anonymous classes from Java 7. Lambdas do make things a lot more concise!
  - Optional: If you wish to try Python, in in the folder `py` you will find examples in Python. For instance `wordcount.py` can be run with (one command):
    `spark-submit --master spark://cluster-01:7077 wordcount.py`
    `hdfs://cm:9000/uhadoop/shared/wiki/es-wiki-abstracts-100l.txt`.
    (You can if you wish adapt this example, but in the class we will focus on Java 8.)
- Before we start coding, let's try run an example.
  - First we will build the JAR package:
    * In the project in Eclipse, right click on `build.xml`, then `Run As ...`, then make sure `dist` is clicked and hit `Run`. If it fails, you may need to manually make a new `dist` folder in the project root. Refresh the project (F5) and make sure you have the JAR file.

[1] Note that the actual output will use commas and brackets rather than tabs: that is not important.
[2] You must use Spark Core however, not, e.g., Spark SQL.

    * If you get an error mentioning javac1.8 or similar, then right click on the `build.xml`, go to External Tools Configurations and add the line `-Dbuild.compiler=javac1.8` to arguments.
    * If you get a new error saying something about `JAVA_HOME` not found, follow: `Window >` `Preferences >` Expand `Ant` tree `> Runtime > Highlight Global Entries > Add External Jars >` Then find and add `tools.jar` in the `lib/` folder of an available JDK (e.g., `C:/Program Files (x86)/Java/jdk.../lib/tools.jar`).
  - Next you need to copy the JAR to your local folder on the server using `scp`, WinSCP, etc. (like in previous labs). Log in and go to the same folder.
  - Run (all one command) `spark-submit --master spark://cluster-01:7077 mdp-spark.jar AverageSeriesRating hdfs://cm:9000/uhadoop/shared/imdb/imdb-ratings-test.tsv hdfs://cm:9000/uhadoop2021/`USERNAME`/series-avg/`
  - Revise the results on HDFS with `hdfs dfs -ls` ..., `hdfs dfs -cat` ... `| more`, etc.

- Now to the main task: your goal is to use Spark Core to produce an output as per the example in the introduction. You need to copy `AverageSeriesRating` to a new class called `InfoSeriesRating` and start extending it to look for info on the best episode. In so doing, you should follow the example of `AverageSeriesRating`, reading the input from HDFS and writing the output to HDFS. You should also ensure to cache any RDDs you use more than once. Please see the example output in the introduction for `The Wire#2002` for the task you need to build; note that is not necessary to have the exact formatting, just the correct information on each line.
  - The JavaDoc for RDDs may also be of use, particularly this page, which lists the various transformations, actions and caching methods and what arguments they take:
    `https://spark.apache.org/docs/2.1.0/api/java/org/apache/spark/api/java/JavaRDD.html`
- Once your class is ready, you should run it on the same data as before for `AverageSeriesRating` and revise the results.
- OPTIONAL: You can try to code this in other languages if you like (Python, Scala). Also try to code the exercises for Lab 3 and Lab 4 in Spark (or think about how you would)!
- SUBMIT: (1) your `InfoSeriesRating.java` class [or alternative in Java 8, Python or Scala], (2) the output tuple for the series "`The Simpsons#1989`" and another output tuple of your choice. You can include (2) as a comment on the submission or at the top of the code.