Spark Exercises

Introduction

For the execution of the exercises associated with this course we will use a virtual machine provided by Cloudera. We can download this machine (cdh 5.12) from their website: https://www.cloudera.com/downloads/quickstart_vms/5-12.html (Characteristics of that machine can be seen in the previous URL). If the machine is not available, it will be provided by the teacher.

To solve these exercises, you can use the Spark's online documentation: https://spark.apache.org/docs/1.5.1/ (menú "Programming Guides", select "Spark Programming Guide") and ask to your partners or to the teacher.

The exercises are prepared to be solved in Scala, but you can also do them in Java or Python (under the personal discretion of each student)

Exercise: Using the Spark Shell (module 1)

The purpose of this exercise is to work with the Spark Shell in Scala to read a file in a RDD.

Tasks to be done:

1. Start the Spark Shell for Scala and get familiar with the information that appears on the screen (Infos, warnings, Scala's version, Spark's version …). It will take a little while to get started.
   a. `spark-shell`
2. Check if a context "sc" has been successfully created as we can see in the documentation
   a. `Escribir "sc"`
   b. You should see something like this in your screen:

   res0:org.apache.spark.SparkContext= org.apache.spark.SparkContext@"*alphanum*"

3. Using the auto-complete command on SparkContext you can see a list of available methods. The Auto-complete function consists of pressing tabulator key after typing the SparkContext object followed by a dot.
4. To exit the Shell you can type "exit" or you can press CTRL + C.

Exercise: Starting with RDDs (module 2)

The objective of this exercise is to practice with RDD's trough Spark Shell, for which we will use external files.

A common practice testing phase of data analitics is to analyze small files that are subsets of related datasets to be used in production. Sometimes, these files are not physically in any node of the cluster, so it will be necessary to import them in some way

A simple way to do these transfers between our Host and the VM/Cluster is through tools like Winscp (https://winscp.net/eng/download.php).
Another option to do this is as in the previous exercises, in other words, through a shared folder with the VM or copying the files directly to the VM.

A- Exploration of plain file 1

Tasks to be done:

1. Start the Spark Shell if you have exited in the previous exercise.
2. For this exercise we are going to work with local data
    a. To access to a local file, we will type, before the path, the word "file:"
3. Create a folder called BIT in "/home" in a way in which it will be created a path "/home/BIT" and copy inside it every data file necessary to the course:
    a. Copy 'data_spark' folder to the virtual machine as in other occasions y get familiar with their content
4. Inside 'data_spark' you will find the file 'relato.txt'. Copy that file to the virtual machine in the next path:
    a. `"/home/BIT/data/relato.txt"`
5. Visualize the file with a text editor like 'gedit' or 'vi' through the shell with 'cat' command.
6. Create a RDD called "relato", it RDD should contain the content of the file using 'textFile' method
    a.
7. Once you have done it, notes that the RDD has not yet been created. This will happen when we will execute an action on the RDD.
8. Count the number of lines of the RDD y take a look to the result. If that result is 23, it's correct.
    a.
9. Execute "collect()" method on the RDD and observe the resut. Remember what we said during the course about when it's recommendable to use this method.
    a.
10. Observe the rest of methods that we can apply on the RDD like we saw in the last exercise
11. If you have time, research about how to use "foreach" function to visualize the content of the RDD in a more appropriate way to understand it
    a.

B- Exploration of the plane file 2

Tasks to be done

1. Copy the weblogs folder contained in the Spark's exercises folder to "/home/BIT/data/weblogs/" and checks its content
2. Choose one of the files, open it and study how it's structured every one of their lines (the data that contains, separators (white space), etc…)

> **116.180.70.237 - 128** [15/Sep/2013:23:59:53 +0100] **"GET /KBDOC-00031.html HTTP/1.0"** 200 1388 "http://www.loudacre.com" "Loudacre CSR Browser"

3. **116.180.70.237** is the IP, **128** is the user number y **GET /KBDOC-00031.html HTTP/1.0** is the article where the action rests.
4. Create a variable that contains the path of the file, for example: file:/home/BIT/data/weblogs/2013-09-15.log
   a.
5. Create an RDD with the content of the file called 'logs'
   a.
6. Create a new RDD, 'jpglogs', containing only the RDD lines that contain the character string ".jpg". You can use the 'contains()' method.
   a.
7. Print in the screen the 5 first lines of 'jpglogs'
   a.
8. It is possible to nest several methods in the same line. Create a variable 'jpglogs2' that returns the number of lines containing the character string ".jpg".
9. We will now start using one of the most important functions in Spark: "map()". To do this, take the 'logs' RDD and calculate the length of the first 5 lines. You can use the functions: "size()" or "length()". Remember that the "map()" function execute one function on each line of the RDD, not on the total set of the RDD.
   a.
10. Print in the screen every word that contains each of the first 5 lines of the 'logs' RDD. You can use the function: "split()".
    a.
11. Map the contents of the logs to an RDD called "logwords" whose contents are arrays of words for each line.
    a.
12. Create a new RDD called "ips" from RDD "logs" that only contains the IPs of each line
    a.
13. Print in the screen the first 5 lines of "ips"
    a.

14. Take a look to the content of “ips” with “collect()” function. You will find it’s not intuitive enough. Try using the “foreach” command.
15. Create a “for” loop to display the contents of the first 10 lines of “ips”. Help: A ‘for’ loop has the following structure:

```
scala> for (x <- rdd.take()) { print(x) }
```

16. Save the whole content of “ips” in a text file using the method “saveAsTextFile” (in the path: “/home/cloudera/iplist”) and take a look at its contents:
    a.

C- Exploration of a set of plain files in a folder

Tasks to do:

1. Create an RDD that only contains the IPs of every document of the path: “/home/BIT/data/weblogs”. Save its contents in the path: “/home/cloudera/iplistw” and observe its content
    a.
2. From the “logs” RDD, create an RDD called “htmllogs” containing only: IP and user ID of each “html” file. The user ID is the third field of each log line. Then print the first 5 lines. An example would be:
    a.
    b.

```
165.32.101.206/8
100.219.90.44/102
182.4.148.56/173
```

Exercise: Working with PairRDDs (module 4)

The objective of this exercise is to get familiar with the work with pair RDD.

A- Work with every data of the logs folder: "/home/BIT/data/weblogs/*"

Tasks to do:

1. Using MapReduce count the number of requests of each user, in other words, count the number of times that each user appears on a line of a log. For that:
    a. Use a Map to create a RDD that contains the pair (ID,1), where the key is the ID field and the Value is the number 1. Remember that ID field is the third element of each line. The data obtained should look like this:

```
(userida, 1)
(userida, 1)
(useridb, 1)
```

    b. Use a Reduce to sum the values of each "userid". The data obtained should look like this:

```
(userida, 4)
(useridb, 8)
(useridc, 12)
```

    a.
    b.

2. Show the users id and the number of accesses for the 10 users with the highest access number. For that:
    a. Use a "map()" to swap the Key for the Value, so you will get something like this (If you do not know how to do that, search on the internet):

```
(4, userida)
(8, useridb)
(12, useridc)
```

    b. Use the function we saw in theory to order a RDD. Please note that we want to show the data in descending order (From highest to lowest number of requests). Remember that the RDD obtained must have the same structure as the original RDD, in other words, with Key: "userid" and Value: "number of requests". The result should be:

```
(193,1603)
(77,1547)
(119,1540)
(34,1526)
(182,1524)
(64,1508)
(189,1508)
(20,1502)
(173,1500)
(17,1500)
```

3. Create a RDD with Key: “userid” and Value: a list of IP to which the previous “userid” has been connected (In other words, group the IPs for “userID”). Use “groupByKey()” function to do that, so the final result should be something like this:
   a.
   b.

```
(userid,[20.1.34.55, 74.125.239.98])
(userid,[75.175.32.10,245.33.1.1,66.79.233.99])
(userid,[65.50.196.141])
```

If you have enough time try to show the RDD obtained on the screen, so that its structure looks like this:

```
ID:79844
IPS:
136.132.254.160
136.132.254.160
53.251.68.51
53.251.68.51
ID:16669
IPS:
23.137.191.64
23.137.191.64
ID:99640
IPS:
207.61.107.245
207.61.107.245
17.159.12.204
17.159.12.204
96.24.214.109
96.24.214.109
123.79.96.8
123.79.96.8
20.117.86.221
20.117.86.221
```

B- Work with every data of the “logs” folder: “/home/BIT/data/accounts.cvs”

Tasks to do:

1. Open “accounts.csv” with a text editor and study its content. You will see that the first field is the user id, which corresponds to the user id of the web server log files. The rest of the fields are: date, name, surname, direction…

2. Do a JOIN between the logs data of the previous exercise and the data of “accounts.csv”, so that you will get a set of data with “userid” as **Key** and all the information of the user (including “userid” field) followed by the number of visits of each user as **Value**. The steps to execute are:

    a. Do a “map()” of the data of “accounts.csv” so that the Key will be “userid” and the Value will be the complete line (including “userid” field). We will get something like this:

```
(userid1, [userid1, 2008-11-24 10:04:08, \N, Cheryl,
West, 4905 Olive Street, San Francisco, CA, ...])
(userid2, [userid2, 2008-11-23 14:05:07, \N,
Elizabeth, Kerns, 4703 Eva Pearl Street, Richmond, CA,
...])
(userid3, [userid3, 2008-11-02 17:12:12, 2013-07-18
16:42:36, Melissa, Roman, 3539 James Martin Circle,
Oakland, CA, ...])
```

    b. Do a JOIN between the RDD recently created and the RDD that you created in the previous step, whose content is (userid, number of visits), so that you will get something like this:

        i.

```
(userid1,([userid1,2008-11-24
    (userid1, [userid1, 2008-11-24 10:04:08, \N, Cheryl,
    West, 4905 Olive Street, San Francisco, CA, ...])
    (userid2, [userid2, 2008-11-23 14:05:07, \N,
    Elizabeth, Kerns, 4703 Eva Pearl Street, Richmond, CA,
    ...])
    (userid3, [userid3, 2008-11-02 17:12:12, 2013-07-18
    16:42:36, Melissa, Roman, 3539 James Martin Circle,
    Oakland, CA, ...])
```

c. Create a RDD from the previous RDD whose content will be **userid, number of visits, name and surname** of the first 5 lines, to get a structure like the next one (the image shows more lines than needed):

i.

C- Work with more methods on pairs RDD

Tasks to do:

1. Use “KeyBy” to create a RDD with accounts data but with the postal(ZIP) code as Key (ninth field of the accounts.csv file). You can research this method in the Spark’s Online API.
    a.
2. Create a pair RDD with the postal (ZIP) code as the Key and a list of names (surname, name) of that postal code as the Value. Those fields are 5ª and 4ª respectively in “accounts.csv”.
    a. If you have time, study “mapValues()” function and try to use it to fulfill the purpose of this exercise
        i.

3. Sort the data by postal code and then, for the first 5 postal codes, display the postal code and a list of names whose accounts are in this postal (ZIP) code. The output should be similar to:

```
--- 85003
Jenkins,Thad
Rick,Edward
Lindsay,Ivy
...
--- 85004
Morris,Eric
Reiser,Hazel
Gregg,Alicia
Preston,Elizabeth
```

Optional Exercises: Working with PairRDDs (module 4.1)

The aim of these exercises is to strengthen the knowledge gained about Pair RDDs with slightly greater difficulty exercises. These exercises are used to better reflect the different uses that we can make of Big Data

EJ1: Tasks to do

1. Take 'shakespeare' dataset provided by the teacher. Copy it to the virtual machine by dragging and dropping the folder.
2. Using Shell, type that dataset to the HDFS (default path for HDFS is: "hdfs://quickstart.cloudera:8020/user/cloudera", copy this dataset to the previous path in 'shakespeare' folder)
    a.
3. Now we are going to do an analysis of the Shakespeare's writing style, for that we want to know the most repeated words in his works. Displays on screen the 100 words that appear most often in Shakespeare plays, along with the frequency of occurrence of each word, in descending order (from higher to lower frequency of occurrence)
    c.

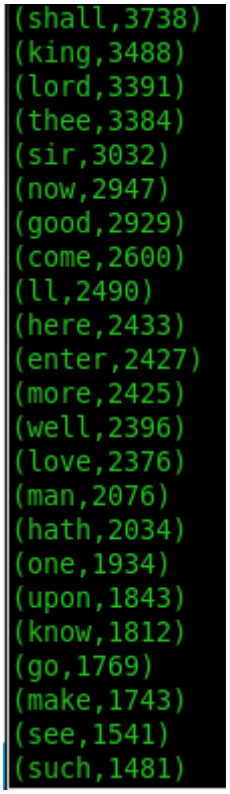
4. The previous analysis is not as complete as a real situation could demand, because among the most repeated words we find: pronouns, articles, prepositions…; in other words, we find words that give us no information. This type of word is known as Stop Words, and we want to remove it from our analysis.

5. To complete this exercise, we are going to need a dataset that contains several Stop Words. Copy the file "stop-word-list.csv" in the virtual machine and copy it again to the HDFS (in the default path)

6. We also have to keep in mind that, in order to make our analysis, we are not interested in lines that only contains blank spaces or in words that only contains one letter. We neither make distinction between upper and lower case letters, so, for example, "Hello" and "hello" should be taken as the same word in our analysis.

    a. The method to change a line to lower-case is: "toLowerCase"
    b. The regular expression for taking only alphanumeric characters is:"[^a-zA-Z]+"
    c. A CSV file is delimited by commas (,)
    d. We are not interested in showing the occurrences of a word with a single letter. So, we must filter the final RDD.

7. The final result should look like this:



EJ2: Tasks to do:

1. Take 'counts' dataset provided by the teacher. Copy it to the virtual machine dragging and dropping the folder.
2. Using the shell, copy this dataset to the HDFS (default path of HDFS is: "hdfs://quickstart.cloudera:8020/user/Cloudera", copy this dataset to the previous path in 'counts' folder).
    a.
3. The dataset we find in 'counts' folder contains lines in which we find a word and a number separated by a tab space. This dataset represents a words frequencies file. From this data: display on screen a RDD composed by key-value pairs, where the key is a letter, symbol or digit and the value is the number of words whose first letter is a key.
    a. The regular expression that represents a tab space is: "\t".
    b. Upper and lower case words should be taken as identical words.

Exercise: SparkSQL (JSON) (module 5.1)

The purpose of this exercise is to get familiar with the use of the SQL module of Spark.

Tasks to do:

1. Create a new SQLContext
   a.
2. Import the implicits that allow us to convert a RDD in a DataFrame
   a.
3. Charge the dataset “zips.json”, which is found in the Spark’s exercises folder, in “/home/BIT/data/zips.json”. This dataset contains United States postal (ZIP) codes. You can use the command “ssc.load(“file path”, “format”)”. The dataset “zips.json” looks like this:

```
{ "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338, "state" : "MA", "_id" : "01001" }
```

4. View the data with “show()” command. You have to see a 5-column table with a subset of the file data. You can see that the postal code is “_id”, the city es “city”, the location “loc”, the population “pop and the state “state”.
   a.
5. Get the postal (ZIP) codes whose population is greater than 10.000 using the API of DataFrames.
   a.
6. Store this table in a temporal file to execute SQL against it.
   a.
7. Make the same query as in point 5, but this time using SQL
   a.
8. Using SQL, get the city with more than 100 postal (ZIP) codes.

```
//Houston, 101
```

9. Using SQL, get the population of the Wisconsin (WI) state.

```
// ([WI,4891769])
```

10. Using SQL, get the 5 most populated states

```
// [CA,29760021]
//[NY,17990455]
//[TX,16986510]
//[FL,12937926]
//[PA,11881643]
```

Exercise: SparkSQL (hive) (module 5.2)

The purpose of this exercise is to get familiar with the use of SparkSQL to access Hive tables, since is a widely used tool in analytical environments.

Tasks to do

1. Open a new Shell and login like root, with pass: cloudera. Execute this command: "cp /usr/lib/hive/conf/hive-site.xml /usr/lib/spark/conf/". For more information, please visit the website "https://community.cloudera.com/t5/Advanced-Analytics-Apache-Spark/how-to-access-the-hive-tables-from-spark-shell/td-p/36609 "
2. Reset Spark's shell.
3. In a terminal, start **Hive** shell and take a look at the databases and tables in each of them.
4. In another terminal, start Spark shell and through SparkSQL create a database and a table with two or three columns. If we do not create the databse, the tables that we are going to create will be stored in the default Hive databse. Before that, you must create a context variable to work with Spark through Hive.
   database: hivespark
   table: employees
   columns: id INT, name STRING, age INT
   config table: FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'
   a.
   b.
   c.
5. Create a file "/home/cloudera/empleado.txt" containing the following data with a similar structure:

   ```
   1201, satish, 25
   1202, krishna, 28
   1203, amith, 39
   1204, javed, 23
   1205, prudvi, 23
   ```

6. Taking advantage of the structure that we have created before, using SparkSQL upload the data of the file "/home/cloudera/empleado.txt" to the Hive table, using HiveQL syntax as we seen in Hives module (LOAD DATA LOCAL INPATH).
   a.

7. Run any queries on the Hive and Spark shells to check that everything works fine and the same data is returned. In the Spark terminal, use the command “show()” to display the data
    a.
    b.
    c.

Exercise: SparkSQL (DataFrames) (module 5.3)

The purpose of this exercise is to get a bit more familiar with the DataFrames API.

1. We create a SQL context
    a.
2. Import the implicits that allow us to convert RDDs to DataFrames and Rows
    a. `import sqlContext.implicits._`
    b. `import org.apache.spark.sql.Row`
    c. `import org.apache.spark.sql.types{StructType,StructField,StringType}`
3. We create a variable with the file path “/home/BIT/data/DataSetPartidos.txt”. The lines have the following format:
    `#idPartido::temporada::jornada::EquipoLocal::EquipoVisitante::golesLocal::golesVisitante::fecha::timestamp`
    a.
4. We store the content of the file in a variable.
    a.
5. We create a variable that contains the schema of the data
    a.
6. We generate a schema based in the variable that contains the data schema that we have recently created
    a.
7. We convert the files of our RDD to Rows
    a)
8. We apply the Schema to the RDD
    a.
9. We register the DataFrame as a table
    a.
10. We are now ready to make queries about the DataFrame in the following format
    a.
    b.
11. The results of the queries are DataFrames and support operations such as normal RDDs. The columns of the results Row are accessible from an index or a field name
    a.
12. Exercise: What is Oviedo’s goal record in a season as visitor?
    a.

b.

13. Who has been more seasons in 1 division: Sporting or Oviedo?
    a.
    b.

Optional exercises: Working with SparkSQL (module 5.4)

The aim of this exercise is to consolidate the knowledge acquired with SparkSQL in a real environment. For this we have a dataset with all the episodes of the Simpsons, their seasons and their IMDB qualification (among other parameters).

Ex1: Tasks to do

1. Take the dataset' simpsons_episodes. csv' provided by the teacher. Copy it to the virtual machine by dragging and dropping the file.
2. Using the terminal, transfer this dataset to the HDFS (The default path of the HDFS is: hdfs: //quickstart. cloudera: 8020/user/cloudera, copy the dataset to this path.)
3. The dataset we have is structured, that is, its columns have headers (as in a database). Take a moment to familiarize yourself with the dataset and its respective fields.
4. The objective of this exercise is to obtain a graphical plot in which we can appreciate, graphically, the average score of the Simpsons' chapters during all their seasons. This exercise is a possible real case in which we can obtain information and extract conclusions by analyzing a multitude of data.
5. To load 'csv' files into Spark we need to use this library, while to create the graphical plot from the data we obtain we need this one. To execute Spark from the terminal and indicate the libraries we want to use, we must execute the command (IMPORTANT TO EXECUTE WITHOUT SPACES):
    1. spark-shell --packages com.databricks:spark-csv_2.10:1.5.0,org.sameersingh.scalaplot:scalaplot:0.0.4
    2. Note that when executing the above command, Spark will automatically load the libraries, as it will download them from the Maven Central repository, where both are hosted.
6. While loading the Spark console, you'll notice that it takes more time and that you see more information on the screen. This is due to the fact that Spark is downloading and adding the libraries indicated in the previous step. Once Spark has loaded, copy these IMPORTS to the console:
    1. import org.sameersingh.scalaplot.Implicits._

       import org.sameersingh.scalaplot.MemXYSeries

       import org.sameersingh.scalaplot.XYData

       import org.sameersingh.scalaplot.XYChart

       import org.sameersingh.scalaplot.gnuplot.GnuplotPlotter;

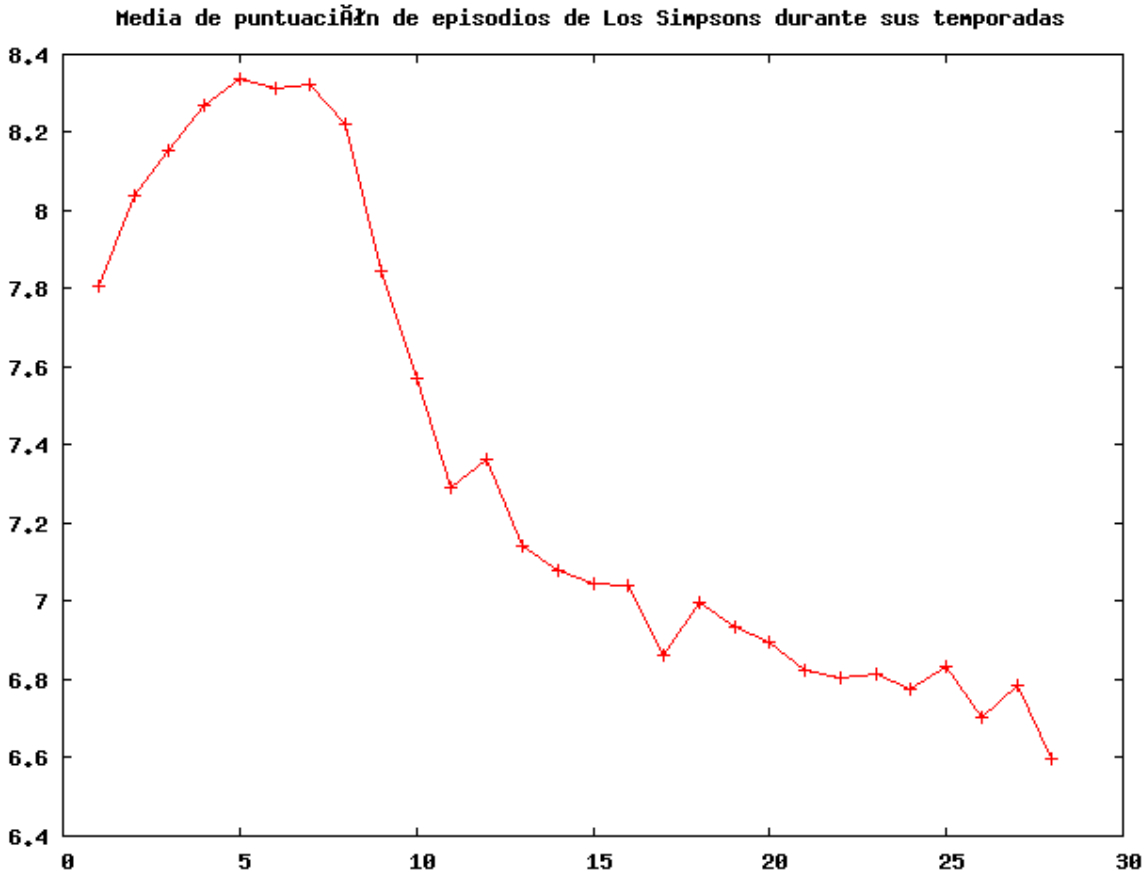
7. Investigate on internet how to transform a CSV file into a Dataframe in Spark 1.6 and perform the transformation.
    1. HINT: As you can see in point 5, to carry out the transformation we have imported the library “com.databricks.spark.csv”.
    2.
8. As the data is structured, we will use SparkSQL. The development from here can be done in different ways, but, logically, the explanation of the exercise will focus on one of them. In the theory part we have seen that Spark can execute SQL commands over a dataset creating a virtual table using the method:
    1. df.createOrReplaceTempView("VirtualTableName") but since the Spark version we are using is 1.6, we will have to use the method: df.registerTempTable("VirtualTableName")
9. Once we have the virtual table created, we will have to create a DataFrame with the data: season and the average of the scores for that season. Look up your notes for the command to execute SQL code in Spark and make the result a DataFrame. Once you have the above command, it executes the SQL code necessary to obtain the data you are looking for. Hints:
    1. The “season” field is in STRING format, you will need to change it to INT.
    2. The resulting DataFrame must be sorted ASCENDENT by season number.
    3. You will find the predefined SQL functions CAST and MEAN very useful.
    4.
10. Once you execute what is required in the previous point, it displays the resulting Dataframe on screen to ensure that the result is as expected.
    1. df.show() Being df the Dataframe resulting from the execution of the previous step.
11. As we are making a real case, we have been able to see that the data is not always in the format that interests us most (we have had to change the type of STRING to INT). It changes the format of the previous Dataframe to a PairRDD (RDD made up of a key and a value, the key being the season number and the value being the mean score for that season). You will find useful the functions seen in the theory.
    1.
12. We are now entering the final stage of the exercise and we want to create, with our results, a linear plot that represents the information obtained, so that it is easier for us to extract conclusions. Now we need to divide the previous RDD, so that we will store in a variable Y the different seasons that we have analyzed, while we will store in a variable X the average scores of those seasons. Note that the variable ‘rdd’ is the name of the variable containing the PairRDD obtained in the previous step.
    1. val x =
    2. val y =

13. Now open a new terminal. Make sure you are on the path:"/home/cloudera" and create a new folder called "docs". Use the command "mkdir" to do this.

14. In that same terminal, run the command: *sudo yum install gnuplot*. This command will install a utility needed to perform the linear diagram.

15. Back at the Scala terminal, run the following code to create the linear diagram. Enter each line one at a time to avoid problems. Once executed and, if everything has gone well, searches for the result in the folder created in step 13.

```
val series = new MemXYSeries(x.collect(), y.collect(), "score")
val data = new XYData(series)
val chart = new XYChart("Average Simpsons episode scores during their seasons", data)
output(PNG("docs/", "test"), chart)
```



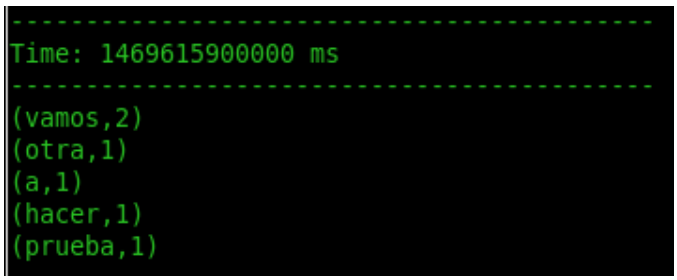
Exercise: Spark Streaming I (module 6.1)

The aim of this exercise is to get started in the use of Spark Streaming and observe its qualities. To do this, we will generate an execute a script in a terminal that will count the words that we introduce in another terminal as simulated streaming.

Tasks to do

1. Visit Spark's documentation https://spark.apache.org/docs/1.5.2/streaming-programming-guide.html and get familiar with the exercise. The aim is to do the same that it says on the web in "A Quick Example" section.
2. Take a time to navigate through the website and to explore it. When you think you are ready, start the exercise.
3. Open a new terminal and type the next command: "nc -lkv 4444". What that command does is to send everything that you write to the 4444 port.
4. Open a new terminal and start Spark's shell in local mode with at least 2 threads, because we will need them to complete this exercise: "spark-shell --master local[2]"
5. Now, access to the file "/usr/lib/spark/conf/log4j.properties", and change the log level to ERROR, so that you can easily see the streaming of counted words returned by your script.
6. Import the necessary classes to work with Spark Streaming
   a. `import org.apache.spark.streaming.StreamingContext`
   b. `import org.apache.spark.streaming.StreamingContext._`
   c. `import org.apache.spark.streaming.Seconds`
7. Create a SparkContext with a 5 seconds duration.
   a. `var ssc = new StreamingContext(sc,Seconds(5))`
8. Create a DStream to read text from the port that you type in the command "nc", you also have to type the hostname of your machine, which is "quickstart.cloudera".
   a. `var mystream = ssc.socketTextStream("localhost",4444)`
9. Create a MapReduce, as we saw in theory, to count the number of words that appear in each Stream.
   a. `var words = mystream.flatMap(line => line.split("\\W"))`
   b. `var wordCounts = words.map(x => (x, 1)).reduceByKey((x,y) => x+y)`
10. Print on the screen the results of each batch
    a. `wordCounts.print()`
11. Start the Streaming Context and call to "awaitTermination" to wait until the task finish
    a. `ssc.start()`
    b. `ssc.awaitTermination()`

12. You should see something like this:



13. Once you have finished, exit the terminal where the command “nc” is executed typing CNTRL +C.

14. To run it from a script:

    spark-shell --master local[2] -i prueba.scala

Exercise: Spark Streaming II (module 6.2)

The purpose of this exercise is to familiarize ourselves with the DStream concept. To do this we will simulate web accesses (those in the weblogs folder) and we will work with them as if they were happening in real time, creating a Spark Streaming application that captures those that meet a specific feature that we will indicate below.

To help us with this, I have created a Script in Python that you can find in the Spark exercise folder. The first task is to copy this script in the path "/home/BIT/examples/streamtest. py" (or in another one of your choice), it script will read the files contained in “weblogs” folder and will simulate a streaming of them (similar to what we did with the command "nc" in the last exercise, but automatically and on a set of files)

Tasks to do:

1. Open a new terminal, go to the path: “/home/BIT/examples” and executes the Python script that I have indicated above.

```
python streamtest.py quickstart.cloudera 4444 5 /home/BIT/data/weblogs/*
```

2. Copy the file "StreamingLogs. scalaspark" located in the Spark exercise folder under "/home/BIT/stubs/StreamingLogs. scalaspark" and familiarize yourself with the contents. The objective is to run in the Shell each of the lines that it contain plus the lines that we are going to program for this exercise.
3. Open a new terminal and start the Spark Shell with at least two threads, as in the previous case. You then runs the imports and create a new StreamingContext at one-second intervals.
4. Create a DStream of logs whose host is “quickstart.cloudera” (you can also use "localhost") and whose port is "4444" (as we have indicated by parameter to the Python script above)
   a.
5. Filter the Stream lines that contains the string of characters “KBDOC”

   a.
6. For each RDD, print the number of lines that contains the string of characters indicated. For that, you can use “*foreachRDD()*”.
   a.
7. Save the result of the filtering in a text file on the local file system (create a path in /home/cloudera/...) of the virtual machine, not in hdfs and check the result at the end of the exercise.
   a.
8. To start the exercise, execute the commands start() y awaitTermination()
   a.

EXTRA

9. If you have enough time, try to expand the exercise in the following way:
10. Every two seconds, it displays the number of KBDOC requests taking 10-second windows. Help yourself with the "countByWindow" function and remember to use the checkpoint necessary to use window functions. The idea is to replicate the code made so far and add the lines that are needed to fulfill the objective set in this point.

11. Starts the StreamingContext and waits until its end.
    a.