Session 1. Big Data Analytics with Spark (2 hours)

Laboratori d'Emmagatzematge i Descripció de Continguts Audiovisuals (EDCAV)

1. Introduction

This session is intended to help the student to get started with Apache Spark, a powerful open source framework to process big data with high speed and easy to use.

NOTE: In order to copy-paste code from this tutorial it's better to do it from the github repository: https://github.com/rtous/edcav/tree/master/spark

2. Environment settings

In order to be able to perform the different tasks you should:

1) Boot the PC with the Linux image (preferable) or Windows
2) Log in with the user/password that you use for Atenea (or, in case it does not work, with (invitado, invitado) on Linux or (A2S105-??\invitado, without password) on Windows)

3. Install Spark

Spark requires Java 6 or higher. As we are going to use the Python interactive shell we will need Python 2.6 or higher:

```
$ java –version
$ python -V
```

First of all, we need to download the Spark environment. To do that, we can just execute the following command:

```
$ wget http://apache.rediris.es/spark/spark-1.6.0/spark-1.6.0-bin-hadoop2.6.tgz
```

Alternatively (if you want a different version or you are a Windows user) you can go to https://spark.apache.org/ and download the files. In all of this hands-on we will work with Spark v1.6.0. Important: you have to download one of pre-built version in "Choose a package type" section (for instance we tested this hands-on with spark-1.6.0-bin-hadoop2.6).

Once we have the tarball file, we need to uncompress it:

```
$ tar -xvzf spark-1.6.0-bin-hadoop2.6.tgz
```

Problem binding to localhost (only in room A2S105)

Perform the following actions.

a) *Rename file conf/spark-env.sh.template to conf/spark-env.sh*

b) *Edit conf/spark-env.sh and set SPARK_LOCAL_IP=127.0.0.1*

Let's execute the interactive Python shell:

```
$ bin/pyspark
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Python version 2.7.5 (default, Mar  9 2014 22:15:05)
SparkContext available as sc, HiveContext available as sqlContext.
>>>
```

4. Example "word count" application

Download *example1.txt*.

```
$ wget https://raw.githubusercontent.com/rtous/edcav/master/spark/example1.txt
```

which has the following content:

```
En un lugar de la Mancha, de cuyo nombre no quiero acordarme,
No ha mucho tiempo que vivía un hidalgo de los de lanza en
astillero, adarga antigua, rocín flaco y galgo corredor.
```

Let's now count the words with Spark:

```
>>> linesRDD = sc.textFile("example1.txt")
>>> countsRDD = linesRDD.flatMap(lambda line: line.split(" ")) \
            .map(lambda word: (word, 1)) \
            .reduceByKey(lambda a, b: a + b)
>>> countsRDD.first()
>>> countsRDD.saveAsTextFile("out.txt")
>>> quit()
```

Now that you have run your first Spark code using the shell, it's time learn about programming in it in more detail. In Spark we express our computation through operations on distributed collections that are automatically parallelized across the cluster. These collections are called Resilient Distributed Datasets, or RDDs. In the example above, the variable called *linesRDD* is an RDD, created here from a text file on our local machine.

Once created, RDDs offer two types of operations: *transformations* and *actions*. Transformations construct a new RDD from a previous one. In our text file example, *flatMap*, *map* and *reduceByKey* are transformations. Spark only computes transformations in a lazy fashion, the first time they are used in an action (e.g. *first()* or *saveAsTextFile(…)* are actions).

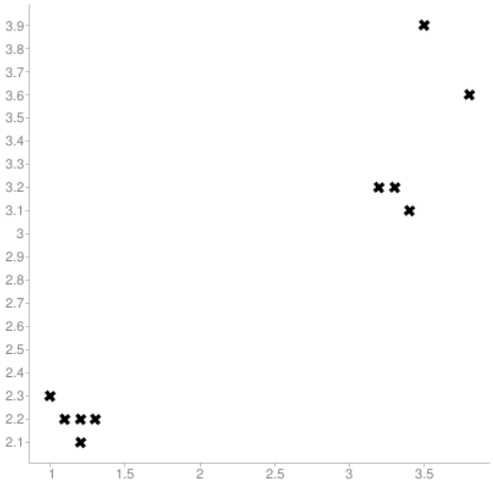
5. Simple clustering example (K-Means)

Download *example2.txt* :

```
$ wget https://raw.githubusercontent.com/rtous/edcav/master/spark/example2.txt
```

which has the following content:

```
1.2 2.1
1.1 2.2
1.0 2.3
1.3 2.2
1.2 2.2
3.3 3.2
3.4 3.1
3.8 3.6
3.5 3.9
3.2 3.2
```



Check the data with the following:

```
>>> data = sc.textFile("example2.txt")
>>> def show (x): print x
>>> data.foreach(show)
```

In the following example, after loading and parsing data, we use the K-Means object to cluster the data into five clusters. The number of desired clusters is passed to the algorithm. We then compute Within Set Sum of Squared Error (WSSSE). You can reduce this error measure by increasing de parameter k.

```
>>> from numpy import array
>>> parsedData = data.map(lambda line: array([float(x) for x in line.split(' ')])).cache()
>>> parsedData.foreach(show)
```

At this point we have an array with the parsed data. With this data, we will train the k-means algorithm and compute the cost. To do that, we need to import some libraries like KMeans and sqrt. We use the KMeans object to cluster the data into two clusters. The number of desired clusters is passed to the algorithm.

```
>>> from pyspark.mllib.clustering import KMeans
>>> clusters = KMeans.train(parsedData, 2, maxIterations=10,
        runs=10, initializationMode="random")
>>> clusters.clusterCenters
```

We can predict the cluster to which a new data point would belong just by doing:

```
>>> clusters.predict([2.2, 2.0])
>>> clusters.predict([3.2, 1.9])
```

We can compute Within Set Sum of Squared Error (WSSSE). You can reduce this error measure by increasing k.

```
>>> from pyspark.mllib.clustering import KMeans
>>> from math import sqrt

def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = parsedData.map(lambda point: error(point)).reduce(lambda x, y: x +
y)
print("Within Set Sum of Squared Error = " + str(WSSSE))
```

6. Working with text

Download *example3.txt* :

```
$                                                                    wget
https://raw.githubusercontent.com/rtous/edcav/master/spark/example3.txt
```

which has the following content:

```
messi barcelona madrid instagram
barcelona paella playa sangria instagram
paella barcelona instagram
ramblas paella ramblas instagram
messi madrid instagram
barcelona messi instagram
messi gol barcelona instagram
sangria paella instagram
```

Let's start by generating an RDD in which each data element corresponds to a line of our input file.

```
>>> documents = sc.textFile("example3.txt").map(lambda line: line.split("
"))
```

Each line of text will represent an independent document (e.g. a headline). We are going to use a classic feature vectorization method (Term Frequency, TF) widely used in text mining to transform our lines of text into numeric vectors. For each term in the entire set of documents (the corpus) we will count the occurrences for a given document. The resulting vector (histogram) will represent the

document. This way we will be able to cluster them with K-Means. Spark already provides functions to compute TF, so let's use them:

```
>>> from pyspark.mllib.feature import HashingTF
>>> hashingTF = HashingTF()
>>> tf = hashingTF.transform(documents)
>>> def show (x): print x
>>> tf.foreach(show)
```

At this point, we have the tf variable that contains the frequencies of the document. Now, we will create two clusters with the KMeans algorithm:

```
>>> from pyspark.mllib.clustering import KMeans
>>> from numpy import array
>>> clusters = KMeans.train(tf, 2, 1, 1)
>>>     results     =     documents.map(lambda     x     :     array([x,
clusters.predict(hashingTF.transform(x))]))
>>> results.foreach(show)
```

7. Working with JSON

Download *example4.txt* :

```
$                                                                    wget
https://raw.githubusercontent.com/rtous/edcav/master/spark/example4.txt
```

which has the following content (JSON data representing tags for pictures):

```
{"tags":["messi","barcelona","madrid"]}
{"tags":["barcelona","paella","playa","sangria"]}
{"tags":["paella","barcelona"]}
{"tags":["ramblas","paella","ramblas"]}
{"tags":["messi","madrid"]}
{"tags":["barcelona","messi"]}
{"tags":["messi","gol","barcelona"]}
{"tags":["sangria","paella"]}
```

First, we will process the data, and store it in a SqlContext:

```
>>> sqlContext = SQLContext(sc)
>>> photos = sqlContext.jsonFile("example4.txt")
>>> def show(x): print x
>>> photos.foreach(show)

>>> photos.registerTempTable("pt")
>>> tags = sqlContext.sql("select tags from pt where tags is not null")
>>> tags.foreach(show)

>>> tagsAsArray = tags.map(lambda x: array(x[0]))
```

Once we have an array with the data, we can repeat the same steps what we did in the previous section, to create the TF and clustering with KMeans.

8. Delivery

A .txt file containing the output of all the commands have to be delivered through the proper section within http://atenea.upc.edu. The delivery has to be done before 1 week.