Máster en Big Data

Tecnologías de Almacenamiento

8. Hands-On: Desarrollo Apache Spark

Àlex Balló

2019

Índice

1. Introducción

El objetivo de este Hands-On es el de familiarizarse con el framework de Spark para el análisis y procesamiento de datos

2. Entorno

Para la realización de los ejercicios se va a utilizar spark-shell en scala ya que nos proporciona un entorno muy dinámico para la introducción de funciones y nos permite recibir una respuesta inmediata.

Para ello, utilizaremos la máquina virtual desplegada en Hands-On anteriores llamada c_Hadoop y ejecutaremos el Spark Shell ubicado en /home/training/spark-1.3.1/bin

El dataset que utilizaremos se llama auctiondata.csv y está ubicado en /home/training/training_materials/developer/data/auction.csv

3. Primer contacto

Lo primero que vamos a hacer en este Hands-On es la realización del ejemplo que aparece en las transparencias del tema 3 en la página 61. Sigue los pasos que se describen a continuación:

i. Cargar el primer RDD con el archivo auctiondata.csv (el path ha de ser el correcto y tiene que hacer referencia a la localización del mismo en el file system local)

```
scala> val auctionRdd= sc.textFile("/home/training/training_materials/developer/
data/auctiondata.csv")
```

ii. Aplicar una transformación de filtrado para los resultados que contengan la palabra “Xbox”

iii.
```
scala> val bidAuctionRDD= auctionRdd.filter(line=>line.contains("xbox"))
```

iv. Aplica una acción para contar el número de resultados que aparecen

```
scala> println(bidAuctionRDD.count())
2784
```

4. Carga e Inspección de datos

Primero se recomienda mapear las variables:

```
val auctionid = 0
val bid = 1
val bidtime = 2
val bidder = 3
val bidderrate = 4
val openbid = 5
val price = 6
val itemtype = 7
val daystolive = 8
```

Contesta a los apartados siguientes enganchando la función que contesta a la pregunta y el resultado obtenido si es que existe

a) Carga el archivo auctiondata.csv haciendo un Split sobre el mismo con el separador ",".
El RDD debe llamarse auctionRDD
Pista: La función recibirá un condición del tipo: `_.split(",")`

sc.textFile("/home/training/training_materials/developer/data/auctiondata.csv")

.map(_.split(","))

b) Que transformaciones y/o acciones se deben utilizar en cada uno de los casos?
   a. Como ver el primer elemento del inputRdd?

```
scala> println(auctionRdd.first().deep)
```

Resultado:

```
Array(8213034705, 95, 2.927373, jake7870, 0, 95, 117.5, xbox, 3)
```

   b. Como ver los 5 primeros elementos del RDD?

```
scala> println(auctionRdd.take(5).deep)
```

Resultado:

```
Array(Array(8213034705, 95, 2.927373, jake7870, 0, 95, 117.5, xbox, 3), Ar
13034705, 115, 2.943484, davidbresler2, 1, 95, 117.5, xbox, 3), Array(8213
, 100, 2.951285, gladimacowgirl, 58, 95, 117.5, xbox, 3), Array(8213034705
5, 2.998947, daysrus, 10, 95, 117.5, xbox, 3), Array(8213060420, 2, 0.0652
nnie4814, 5, 1, 120, xbox, 3))
```

   c. Cuál es el número total de pujas?

```
scala> println(auctionRdd.count())
```

Resultado:

```
10654
```

d. Número total de elementos diferentes que se han subastado ? (guardar resultado en una variable llamada totitems)

```
scala> val auctionItemRdd= auctionRdd.groupBy(x=>x(auctionid))

scala> val toitems= auctionItemRdd.count()

toitems: Long = 627
```

e. ¿Cuál es el número total de tipos de elementos que se han subastado? (guardar resultado en una variable llamada totitemtype)

```
scala> val auctionItemRdd= auctionRdd.groupBy(x=>x(itemtype))

scala> val totitemtype= auctionItemRdd.count()
```

Resultado:

```
totitemtype: Long = 3
```

f. ¿Cuál es el número total de ofertas por cada tipo de artículo? (guardar resultado en una variable llamada bids_itemtype)

```
scala> val auctionByItemRdd= auctionRdd.map(x=>(x(itemtype),1))

scala> val auctionCountItemRdd= auctionByItemRdd.reduceByKey((x,y)=>x+y)

scala> val bids_itemtype= auctionCountItemRdd.collect()
```

Resultado:

```
bids_itemtype: Array[(String, Int)] = Array((xbox,2784), (palm,5917), (cartier,1953))
```

g. ¿Cuál es el número total de ofertas por subasta? (Crear un RDD llamada bids_auctionRDD)

```
scala> val bids_auctionRDD=auctionRdd.map(x=>(x(auctionid),1)).reduceByKey((x,y)=>x+y);

scala> bids_auctionRDD.collect()
 Array[(String, Int)] = Array((3024504428,1), (3024707992,20), (3014792711,7), (3018904443,24), (3025035412,1), (3014835507,27), (3023275213,7), (3024895548,7),
526592,8), (1650483277,30), (1649858595,7), (1642243766,11), (3023748273,42), (8213472092,3), (8215125069,19), (3013951754,18), (1640550476,23), (3015958025,20),
5307344,17), (3014834982,20), (8215582227,16), (3014834745,21), (3016892738,19), (3023389524,16), (8212236671,43), (1642911743,3), (3021870696,9), (8212264580,22
920026227,20), (3024980402,26), (8212145833,26), (1639672910,4), (3025885755,7), (3014616784,23), (1650986455,5), (3018288277,32), (3023187210,11), (3016035790,1
3024307294,3), (3017950485,20), (1640179146,22), (3015710025,7), (3024799631,18), (8212602164,50), (8214418083,12), (3018131250,...
```

A partir de aquí, se recomienda utilizar: import java.lang.Math

h. En todos los artículos subastados, ¿Cuál es el número máximo de pujas?

```
scala> val max_bids=bids_auctionRDD.map(x=>x._2).reduce((x,y)=>Math.max(x,y))
```

Resultado:

```
max_bids: Int = 75
```

i. En todos los artículos subastados, ¿Cuál es el número mínimo de pujas?

```
scala> val min_bids=bids_auctionRDD.map(x=>x._2).reduce((x,y)=>Math.min(x,y))
```

Resultado:

```
min_bids: Int = 1
```

j. ¿Cuál es el número medio de pujas?

```
scala> val averagebids=tobids/toitems
averagebids: Long = 16
```