Máster en Big Data

Tecnologías de Almacenamiento

7. Hands-On: Desarrollo Apache Spark

Presentado por: **José David Angulo, Arnau Recio y Albert Ripoll**

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 Developer_Hadoop y ejecutaremos el Spark Shell ubicado en */home/training/spark-1.3.1/bin*

Ejecutamos el Spark Shell llamado "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-Apache Spark* a partir de la página 64. Sigue los pasos que se describen a continuación:

3.1. 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)

En consola ejecutamos el siguiente código.

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

3.2. ¿Cuantos elementos contiene el dataset cargado?

En consola ejecutamos el siguiente código.

```
val countRDD = auctionRDD.count()
```

Y nos devuelve: Long=10654 debajo de todo de la imagen siguiente.

3.3. Aplicar una transformación de filtrado para los resultados que contengan la palabra "xbox" (p. 73) y aplica una acción para contar el número de resultados que aparecen

Si aplicamos la siguiente transformación

```
val xboxRDD = auctionRDD.filter(line=>line.contains("xbox"))
```

No sucede nada debido al concepto de *Lazy Evaluated*: No se realiza ninguna transformación hasta que no se realice antes una acción. Por lo tanto, con esa transformación no sucede nada. Para eso hay que realizar una acción posteriormente como collect(), count() o take().

Nosotros usamos la transformación count() posterior a la transformación.

```
xboxRDD.count()
```

La respuesta es que hay 2784 resultados con la palabra "xbox"

4. Carga e Inspección de datos

Primero se recomienda mapear las variables (columnes de el csv):

```
val auctionid = 0 (Identificador de la subhasta)
val bid = 1 (Cantidad de dinero pujado)
val bidtime = 2 (Momento de la apuesta)
val bidder = 3 (Persona que apostó)
val bidderrate = 4
val openbid = 5
val price = 6
val itemtype = 7 (Tipo de items. Hay 3. xbox y otros dos)
val daystolive = 8
```

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

4.1. 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(","))

```
val auctionRDD =
sc.textFile("/home/training/training_materials/developer/data/auctiondata.csv").map(_.split(","))
```

4.2. Que transformaciones y/o acciones se deben utilizar en cada uno de los casos?

a. ¿Como ver el primer elemento del inputRdd?

`auctionRDD.first()` → Retorna una array con el primer resultado. Sería equivalente a auctionRDD.take(1)

Respuesta:

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

b. ¿Como ver los 5 primeros elementos del RDD?

`auctionRDD.take(5)` → Retorna una array con los cinco primeros resultados.

Respuesta:

Array(Array(8213034705, 95, 2.927373, jake7870, 0, 95, 117.5, xbox, 3), Array(8213034705, 115, 2.943484, davidbresler2, 1, 95, 117.5, xbox, 3), Array(8213034705, 100, 2.951285, gladimacowgirl, 58, 95, 117.5, xbox, 3), Array(8213034705, 117.5, 2.998947, daysrus, 10, 95, 117.5, xbox, 3), Array(8213060420, 2, 0.065266, donnie4814, 5, 1, 120, xbox, 3))

c. ¿Cuál es el número total de pujas?

auctionRDD.count() → Entendiendo que cada línea es una puja, devuelve que hay 10654 pujas

d. ¿Cuál es el número total de elementos diferentes que se han subastado? (guardar resultado en una variable llamada totitems)

Ejecutamos:

```
val UniqueRDD = auctionRDD.map(x=>x(auctionid)).distinct
```

```
UniqueRDD.count()
```

Coge la primera posición de todas les columnas, o sea el identificador de la subasta (AuctionID)

Respuesta:

627

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

Ejecutamos lo siguiente para que cuente *count()* los valores únicos *distinct()* columna 7 que se llama "ítemtype"

```
Val totitemtype = auctionRDD.map(x=>x(itemtype)).distinct().count()
```

Y devuelve:

Long = 3

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

```
val maxRDD = auctionRDD.map(x=>(x(auctionid),1)).reduceByKey((x,y)=>x+y).map(x=>x._2).max()
```

Resultado:
Int = 75

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

• Forma 1:

```
val maxRDD = auctionRDD.map(x=>(x(auctionid),1)).reduceByKey((x,y)=>x+y).map(x=>x._2).max()
```

resultado: Int = 75

• Forma 2:

2.1) Creamos la variable sumaRDD con la array con todos los auctionid y la cantidad de sumas en ese.

```
val sumaRDD = auctionRDD.map(x=>(x(auctionid),1).reduceByKey((x,y)=>x+y).collect()
```

```
sumaRDD: Array[(String, Int)] = Array((3024504428,1), (3024707992,20), (3014792711,7), (3018904443,24), (3025035412,1), (3014835507,27), (30
023748273,42), (8213472092,3), (8215125069,19), (3013951754,18), (1640550476,23), (3015958025,20), (3025307344,17), (3014834982,20), (821558
21870696,9), (8212264580,22), (3020026227,20), (3024980402,26), (8212145833,26), (1639672910,4), (3025885755,7), (3014616784,23), (165098645
9146,22), (3015710025,7), (3024799631,18), (8212602164,50), (8214418083,12), (30181312...
```

2.2) Maximizamos por el segundo valor de la tupla:

```
val maxRDD = sumaRDD.maxBy(_._2)
```

```
scala> val maxRDD = sumaRDD.maxBy(_._2)
maxRDD: (String, Int) = (8214355679,75)
```

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

• Forma 1:

```
val minRDD = auctionRDD.map(x=>(x(auctionid),1)).reduceByKey((x,y)=>x+y).map(x=>x._2).min()
```

resultado: Int = 1

• Forma 2:

2.1) Creamos la variable sumaRDD con la array con todos los auctionid y la cantidad de sumas en ese.

```
val sumaRDD = auctionRDD.map(x=>(x(auctionid),1).reduceByKey((x,y)=>x+y).collect()
```

```
sumaRDD: Array[(String, Int)] = Array((3024504428,1), (3024707992,20), (3014792711,7), (3018904443,24), (3025035412,1), (3014835507,27), (30
023748273,42), (8213472092,3), (8215125069,19), (3013951754,18), (1640550476,23), (3015958025,20), (3025307344,17), (3014834982,20), (821558
21870696,9), (8212264580,22), (3020026227,20), (3024980402,26), (8212145833,26), (1639672910,4), (3025885755,7), (3014616784,23), (165098645
9146,22), (3015710025,7), (3024799631,18), (8212602164,50), (8214418083,12), (30181312...
```

2.2) Minimizamos por el segundo valor de la tupla:

```
val minRDD = sumaRDD.minBy(_._2)
```

```
scala> val minRDD = sumaRDD.minBy(_._2)
minRDD: (String, Int) = (3024504428,1)
```

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

• Tenemos la variable sumaRDD que es una array con todos los auctionid y la cantidad de sumas en ese.

```
sumaRDD: Array[(String, Int)] = Array((3024504428,1), (3024707992,20), (3014792711,7), (3018904443,24), (3025035412,1), (3014835507,27), (30
023748273,42), (8213472092,3), (8215125069,19), (3013951754,18), (1640550476,23), (3015958025,20), (3025307344,17), (3014834982,20), (821558
21870696,9), (8212264580,22), (3020026227,20), (3024980402,26), (8212145833,26), (1639672910,4), (3025885755,7), (3014616784,23), (165098645
9146,22), (3015710025,7), (3024799631,18), (8212602164,50), (8214418083,12), (30181312...
```

• Creamos una variable entera sumValues que es la suma del valor (el segundo _._2 valor de la tupla) de sumaRDD

```
val sumValues: Int = sumaRDD.map(_._2).sum
```

```
scala> val sumValues: Int = sumaRDD.map(_._2).sum
sumValues: Int = 10654
```

• Calculamos en formato double (punto flotante de doble precisión) el valor medio. Para hacerlo dividimos la suma anterior pasada a double y la longitud de la array sumaRDD.

val averageValue: Double = sumValues.toDouble / sumaRDD.length

```
scala> val averageValue: Double = sumValues.toDouble / sumaRDD.length
averageValue: Double = 16.992025518341308
```