

## SPARK

Spark es un framework para programación distribuida sobre datos en paralelo

## RDD

Rdd : Resilient Distributed Dataset es un simple, RESILIENTE e INMUTABLE colección DISTRIBUIDA de objetos

RESILIENTE podemos recuperar los datos si se pierden

INMUTABLE no podemos realizar modificaciones sobre un mismo rdd

DISTRIBUIDO cada rdd es dividido en múltiples particiones automáticamente

DATASET conjunto de datos 2 tipos de fuentes : externas y datos en memoria

Un DStream es una secuencia de datos que llegan en un periodo de tiempo

Internamente cada DStream es representado por una secuencia de RDDs que llegan en cada paso, de ahí el nombre discretized

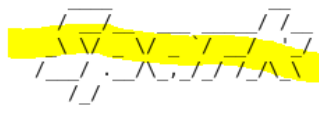
Los DStream pueden ser creados desde varios orígenes de datos como flume, kafka o hdfs

Ejercicio: Usando el Shell de Spark (modulo 1) El propósito de este ejercicio es trabajar con el Shell de Spark en Scala para leer un fichero en un RDD. Tareas a realizar

1. Arrancar el Shell de Spark para scala y familiarizarse con la información que aparece por pantalla (Infos, Warnings, versión de Scala y Spark, etc...). Tarda un poco en arrancar. a. spark-shell

```

[cloudera@quickstart ~]$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/avro/avro-tools-1.7.6-cdh5.13.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Welcome to

 version 1.6.0

Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_67)
Type in expressions to have them evaluated.
Type :help for more information.

```

2. Comprobar que se ha creado un contexto "sc" tal y como vimos en la documentación a. Escribir "sc" b. Deberéis ver por pantalla algo de este tipo:  
res0:org.apache.spark.SparkContext=  
org.apache.spark.SparkContext@"alfanum"

```

scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@1918675
scala>

```

3. Usando el comando de autocompletado sobre el SparkContext, podéis ver los métodos disponibles. La función de autocompletado consiste en presionar el tabulador después de escribir el objeto SparkContext seguido de un punto

```
scala>
scala> NADA
<init>
LEGACY_DRIVER_IDENTIFIER
RDD_SCOPE_NO_OVERRIDE_KEY
SPARK_JOB_GROUP_ID
StringToColumn
abs
add_months
array
asc
asin
atan2
base64
bitwiseNOT
booleanWritableConverter
bytesToBytesWritable
callUDF
cbrt
classOf
clone
col
collect_set
concat
conv
cos
count
crc32
cume_dist
current_timestamp
date_format
datediff
dayofyear
denseRank
desc
doubleToDoubleWritable
encode
equals
explode
DRIVER_IDENTIFIER
RDD_SCOPE_KEY
SPARK_JOB_DESCRIPTION
SPARK_JOB_INTERRUPT_ON_CANCEL
_sqlContext
acos
approxCountDistinct
array_contains
ascii
atan
avg
bin
boolToBoolWritable
broadcast
bytesWritableConverter
callUdf
ceil
clearActiveContext
coalesce
collect_list
column
concat_ws
corr
cosh
countDistinct
cumeDist
current_date
date_add
date_sub
dayOfMonth
decode
dense_rank
doubleRDDToDoubleRDDFunctions
doubleWritableConverter
eq
exp
expm1
```

SOLO TAB

4. Para salir del Shell, se puede escribir “exit” o presionar Cntrl+C saprkcontext

## Ejercicio: Comenzando con los RDDs (módulo 2)

2 El objetivo de este ejercicio es practicar con los RDDs a través del Shell de Spark, para lo que usaremos ficheros externos. Una práctica habitual en el análisis de datos en su etapa de pruebas es analizar ficheros pequeños que son subconjuntos similares a los datasets que se usarán en producción. En ocasiones, estos ficheros no se encuentran físicamente en ninguno de los nodos del cluster, por lo que es necesario importarlos de alguna manera. Una forma sencilla de hacer estas transferencias entre nuestro Host y la MV/Cluster es a través de herramientas como Winscp <https://winscp.net/eng/download.php> Otra opción es hacerlo como en los ejercicios pasados, a través de una carpeta compartida con la MV. A- Exploración de fichero plano 1 Tareas a realizar

1. Inicia el Shell de Spark si te saliste en el paso anterior.

2. Para este ejercicio vamos a trabajar con datos en local a. Para acceder al fichero en local, se coloca delante de la ruta la palabra "file:"

3. Crea una carpeta llamada BIT en "/home" de forma que se cree la ruta "/home/BIT" y copia dentro de ella todos los ficheros de datos necesarios para el curso: a. Copiar la carpeta data\_spark a la máquina virtual como en otras ocasiones y familiarizaos con su contenido

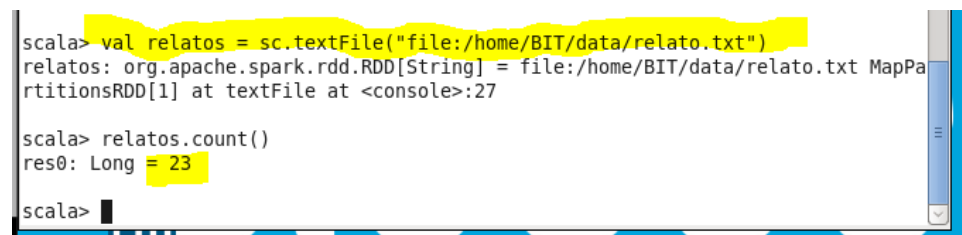
4. Dentro de data\_spark se encuentra el fichero "relato.txt". Copia este fichero en la máquina virtual en la siguiente ruta: a. "/home/BIT/data/relato.txt" a. "/home/BIT/data/relato.txt"

5. Visualiza el fichero con un editor de texto como "gedit" o "vi" o a través de la Shell con el comando "cat" print

6. Crea un RDD llamado "relato" que contenga el contenido del fichero utilizando el método "textFile" a.

7. Una vez hecho esto, observa que aún no se ha creado el RDD. Esto ocurrirá cuando ejecutemos una acción sobre el RDD

8. Cuenta el número de líneas del RDD y observa el resultado. Si el resultado es 23 es correcto. a.



```
scala> val relatos = sc.textFile("file:/home/BIT/data/relato.txt")
relatos: org.apache.spark.rdd.RDD[String] = file:/home/BIT/data/relato.txt MapPartitionsRDD[1] at textFile at <console>:27

scala> relatos.count()
res0: Long = 23

scala>
```

9. Ejecuta el método "collect()" sobre el RDD y observa el resultado. Recuerda lo que comentamos durante el curso sobre cuándo es recomendable el uso de este método. A.

```
scala> relatos.collect()
res1: Array[String] = Array(Two roads diverged in a yellow wood,, And sorry I could not travel both, And be one traveler, long I stood, And looked down one as far as I could, To where it bent in the undergrowth;; "", Then took the other, as just as fair,, And having perhaps the better claim,, Because it was grassy and wanted wear;; Though as for that the passing there, Had worn them really about the same,, "", And both that morning equally lay, In leaves no step had trodden black., Oh, I kept the first for another day!, Yet knowing how way leads on to way, I doubted if I should ever come back., "", I shall be telling this with a sigh, Somewhere ages and ages hence:, Two roads diverged in a wood, and I--, I took the one less traveled by,, And that has made all the difference.)
scala>
```

10. Observa el resto de métodos aplicables sobre el RDD como vimos en el ejercicio anterior.

11. Si tienes tiempo, investiga cómo usar la función “foreach”

para visualizar el contenido del RDD de una forma más cómoda de entender

```
scala> relatos.foreach(println)
Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;

Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,

And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.

I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I--
I took the one less traveled by,
And that has made all the difference.

scala>
```

### a. 3 B- Exploración de fichero plano

#### 2 Tareas a realizar

1. Copia la carpeta weblogs contenida en la carpeta de ejercicios de Spark a “/home/BIT/data/weblogs/” y revisa su contenido.

2. Escoge uno de los ficheros, ábrelo, y estudia cómo está estructurada cada una de sus líneas (datos que contiene, separadores (espacio), etc)

3. 116.180.70.237 es la IP, 128 el número de usuario y GET /KBD0C-00031.html HTTP/1.0 el artículo sobre el que recae la acción.

4. Crea una variable que contenga la ruta del fichero, por ejemplo  
file:/home/BIT/data/weblogs/2013-09-15.log a. Val  
ruta="file:/home/BIT/data/2013-09-15.log"

```
scala> val log="file:/home/BIT/data/weblogs/2013-09-15.log"
log: String = file:/home/BIT/data/weblogs/2013-09-15.log

scala> val logs=sc.textFile(log)
logs: org.apache.spark.rdd.RDD[String] = file:/home/BIT/data/weblogs/2013-09-15.
log MapPartitionsRDD[3] at textFile at <console>:29

scala>
```

5. Crea un RDD con el contenido del fichero llamada logs a. Val  
logs=sc.textFile(ruta)

```
scala> val log="file:/home/BIT/data/weblogs/2013-09-15.log"
log: String = file:/home/BIT/data/weblogs/2013-09-15.log

scala> val logs=sc.textFile(log)
logs: org.apache.spark.rdd.RDD[String] = file:/home/BIT/data/weblogs/2013-09-15.
log MapPartitionsRDD[3] at textFile at <console>:29

scala>
```

6. Crea un nuevo RDD, jpglogs, que contenga solo las líneas del RDD que contienen la cadena de caracteres ".jpg". Puedes usar el método contains() a.  
val jpglogs=logs.filter(x=>x.contains(".jpg"))

```
scala> val jpglogs=logs.filter(x=>x.contains(".jpg"))
jpglogs: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[4] at filter at <co
nsole>:31

scala> jpglogs.take(5)
res3: Array[String] = Array(233.19.62.103 - 16261 [15/Sep/2013:23:55:57 +0100] "
GET /titanic_1100.jpg HTTP/1.0" 200 16713 "http://www.loudacre.com" "Loudacre M
obile Browser Sorrento F10L", 136.98.55.4 - 22232 [15/Sep/2013:23:52:49 +0100] "
GET /ifruit_1.jpg HTTP/1.0" 200 7127 "http://www.loudacre.com" "Loudacre Mobile
Browser iFruit 2", 219.248.10.104 - 66652 [15/Sep/2013:23:49:59 +0100] "GET /if
ruit_4a.jpg HTTP/1.0" 200 1686 "http://www.loudacre.com" "Loudacre Mobile Brows
er Sorrento F20L", 47.148.67.112 - 40002 [15/Sep/2013:23:44:07 +0100] "GET /meet
oo_5.1.jpg HTTP/1.0" 200 6804 "http://www.loudacre.com" "Loudacre Mobile Browse
r Titanic 2200", 29.110.195.184 - 39859 [15/Sep/2013:23:42:32 +0100] "GET /titan
ic_2400.jpg HTTP/1.0" 200 4175 "http://www.loudacre.com" "Loudacre Mobile Br...

scala>
```

7. Imprime en pantalla las 5 primeras líneas de jpglogs a b. jpglogs.take(5)

```
scala> val jpglogs=logs.filter(x=>x.contains(".jpg"))
jpglogs: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[4] at filter at <console>:31

scala> jpglogs.take(5)
res3: Array[String] = Array(233.19.62.103 - 16261 [15/Sep/2013:23:55:57 +0100] "
GET /titanic_1100.jpg HTTP/1.0" 200 16713 "http://www.loudacre.com" "Loudacre M
obile Browser Sorrento F10L", 136.98.55.4 - 22232 [15/Sep/2013:23:52:49 +0100] "
GET /ifruit_1.jpg HTTP/1.0" 200 7127 "http://www.loudacre.com" "Loudacre Mobile
Browser iFruit 2", 219.248.10.104 - 66652 [15/Sep/2013:23:49:59 +0100] "GET /if
ruit_4a.jpg HTTP/1.0" 200 1686 "http://www.loudacre.com" "Loudacre Mobile Brows
er Sorrento F20L", 47.148.67.112 - 40002 [15/Sep/2013:23:44:07 +0100] "GET /meet
oo_5.1.jpg HTTP/1.0" 200 6804 "http://www.loudacre.com" "Loudacre Mobile Browse
r Titanic 2200", 29.110.195.184 - 39859 [15/Sep/2013:23:42:32 +0100] "GET /titan
ic_2400.jpg HTTP/1.0" 200 4175 "http://www.loudacre.com" "Loudacre Mobile Br...
scala>
```

8. Es posible anidar varios métodos en la misma línea. Crea una variable jpglogs2 que devuelva el número de líneas que contienen la cadena de caracteres “.jpg” a. `val jpglogs=logs.filter(x=>x.contains(".jpg")).count()`

9. Ahora vamos a comenzar a usar una de las funciones más importantes de Spark, la función “map()”. Para ello, coge el RDD logs y calcula la longitud de las 5 primeras líneas. Puedes usar la función “size()” o “length()” Recordad que la función map ejecuta una función sobre cada línea del RDD, no sobre el conjunto total del RDD. a. `Logs.map(x=>x.length).take(5)`

```
scala> logs.map(x=>x.length).take(5)
res4: Array[Int] = Array(141, 134, 140, 133, 143)

scala>
```

10. Imprime por pantalla cada una de las palabras que contiene cada una de las 5 primeras líneas del RDD logs. Puedes usar la función “split()” a. `Logs.map(x=>x.split(" ")).take(5)`

11. Mapea el contenido de logs a un RDD “logwords” de arrays de palabras de cada línea a. `Val logwords=log.map(line=>line.split(" "))`

```
scala> var logwords = logs.map(line => line.split(' '))
logwords: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[6] at map a
t <console>:31
```

12. Crea un nuevo RDD llamado “ips” a partir del RDD logs que contenga solamente las ips de cada línea (primer elemento de cada fila) a. `Val ips=logs.map(línea=>línea.split(" ")(0))`

```
scala> var ips = logs.map(line => line.split(' ')(0))
ips: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at map at <console>:31
scala>
```

13. Imprime por pantalla las 5 primeras líneas de ips a. `ips.take(5)`  
 116.180.70.237 - 128 [15/Sep/2013:23:59:53 +0100] "GET /KBDOC00031.html HTTP/1.0" 200 1388 "http://www.loudacre.com" "Loudacre CSR Browser" 4

```
scala> ips.take(5)
res7: Array[String] = Array(116.180.70.237, 116.180.70.237, 218.193.16.244, 218.193.16.244, 198.122.118.164)
scala>
```

14. Visualiza el contenido de ips con la función “`collect()`”. Verás que no es demasiado intuitivo. Prueba a usar el comando “`foreach`” a. `ips.collect()`

```
43.93.158.53
43.93.158.53
215.159.42.147
215.159.42.147
68.22.19.132
68.22.19.132
12.254.198.60
12.254.198.60
58.215.255.197
58.215.255.197
36.12.15.59
36.12.15.59
18.120.193.68
18.120.193.68
121.17.90.255
121.17.90.255
35.58.16.118
35.58.16.118
28.174.249.234
28.174.249.234
238.204.235.88
238.204.235.88
115.211.47.107
115.211.47.107
220.114.174.10
220.114.174.10
220.114.174.10
220.114.174.10
194.233.67.145
194.233.67.145
194.233.67.145
194.233.67.145
64.14.85.32
64.14.85.32
246.137.198.87
246.137.198.87
159.93.248.54
159.93.248.54
124.145.96.147
124.145.96.147
174.145.96.147
```

15. Crea un bucle “`for`” para visualizar el contenido de las 10 primeras líneas de ips. Ayuda: un bucle `for` tiene la siguiente sintaxis `for(x <- ips.take(10)){ print(x)}`

```
scala> for(x<-ips.take(10)){print(x)}
116.180.70.237116.180.70.237218.193.16.244218.193.16.244198.122.118.164198.122.118.164103.17.173.248103.17.173.248233.19.62.103233.19.62.103
scala>
```



16. Guarda el contenido de “ips” entero en un fichero de texto usando el método saveAsTextFile en la ruta “/home/cloudera/iplist” y observa su contenido. c. ips.saveAsTextFile("file:/home/cloudera/iplist")

## C- Exploración de un conjunto de ficheros planos en una carpeta

### Tareas a realizar

1. Crea un RDD que contenga solo las ips de todos los documentos contenidos en la ruta “/home/BIT/data/weblogs”. Guarda su contenido en la ruta “/home/cloudera/iplistw” y observa su contenido. a. Val ips=sc.textFile(“file:/home/BIT/data/weblogs”).map(li nea=línea.split(“ “)(0))

```
scala> sc.textFile("file:/home/BIT/data/weblogs/*").map(line => line.split(' ')(0)).saveAsTextFile("file:/home/cloudera/iplistw")
```

2. A partir del RDD logs, crea un RDD llamado “htmllogs” que contenga solo la ip seguida de cada ID de usuario de cada fichero html. El ID de usuario es el tercer campo de cada línea de cada log. Después imprime las 5 primeras líneas. Un ejemplo sería este:

a. Val htmllogs=logs.map((linea=>linea.split(“ “)(0)),(linea=>linea.split(“ “)(2))

b. Htmllogs.take(5) scala> for (x <- rdd.take()) { print(x) } 165.32.101.206/8100.219.90.44/102 182.4.148.56/173 5

```
scala> var htmllogs = logs.filter(_.contains(".html")).map(line => (line.split(' ')(0),line.split(' ')(2)))
htmllogs: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[3] at map at <console>:31

scala> htmllogs.take(5).foreach(t => println(t._1 + "/" + t._2))
116.180.70.237/128
218.193.16.244/94
198.122.118.164/131
103.17.173.248/85365
233.19.62.103/16261

scala>
```

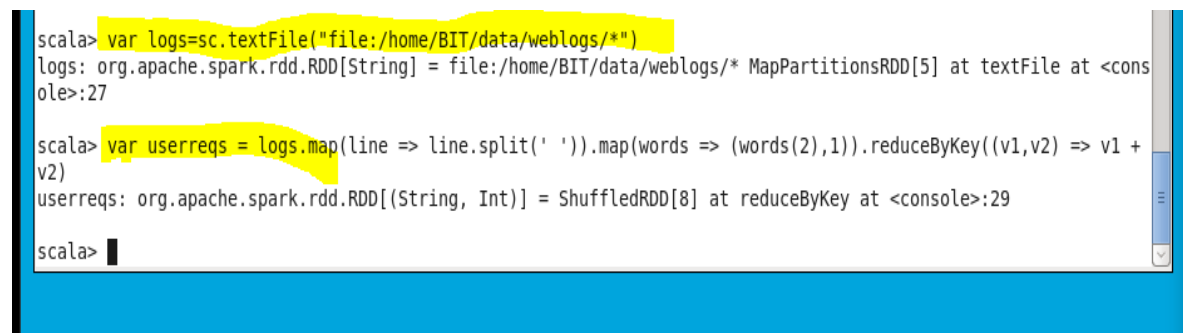
Ejercicio: Trabajando con PairRDDs (módulo 4)

**El objetivo de este ejercicio es familiarizarnos con el trabajo con pares RDD. A-Trabajo con todos los datos de la carpeta de logs: "/home/BIT/data/weblogs/\*" Tareas a realizar**

1. Usando MapReduce, cuenta el número de peticiones de cada usuario, es decir, las veces que cada usuario aparece en una línea de un log. Para ello

a. Usa un Map para crear un RDD que contenga el par (ID, 1), siendo la clave el ID y el Value el número 1. Recordad que el campo ID es el tercer elemento de cada línea. Los datos obtenidos tendrían que quedar de la siguiente manera `var logs=sc.textFile("file:/home/BIT/data/weblogs/*")` `var rdd = logs.map(line => line.split(' ')).map(words => (words(2),1))`

b. Usa un Reduce para sumar los valores correspondientes a cada userid. Los datos tendría que mostrarse de la siguiente manera: `Var red = rdd.reduceByKey((v1,v2)=> v1+v2)`



```
scala> var logs=sc.textFile("file:/home/BIT/data/weblogs/*")
logs: org.apache.spark.rdd.RDD[String] = file:/home/BIT/data/weblogs/* MapPartitionsRDD[5] at textFile at <console>:27

scala> var userreqs = logs.map(line => line.split(' ')).map(words => (words(2),1)).reduceByKey((v1,v2) => v1 + v2)
userreqs: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[8] at reduceByKey at <console>:29

scala>
```

2. Muestra los id de usuario y el número de accesos para los 10 usuarios con mayor número de accesos. Para ello:

a. Utiliza un map() para intercambiar la Clave por el Valor, de forma que quede algo así: (Si no se te ocurre cómo hacerlo, busca en internet) `Var newRed=red.map(campo=>campo.swap())`

b. Utiliza la función vista en teoría para ordenar un RDD. Ten en cuenta que queremos mostrar los datos en orden descendiente (De mayor a menor número de peticiones). Recuerda que el RDD debe estar en la misma forma que al inicio, es decir, con clave: userid y valor: nº de peticiones. El resultado debe ser:

`newRed.sortByKey(false).map(field=>field.swap).take(10).foreach(println) 6`

```
scala> var userreqs = logs.map(line => line.split(' ')).map(words => (words(2),1)).reduceByKey((v1,v2) => v1 + v2)
userreqs: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[8] at reduceByKey at <console>:29

scala> val swapped=userreqs.map(field => field.swap)
swapped: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[9] at map at <console>:31

scala> swapped.sortByKey(false).map(field => field.swap).take(10).foreach(println)
(193,1603)
(77,1547)
(119,1540)
(34,1526)
(182,1524)
(64,1508)
(189,1508)
(20,1502)
(173,1500)
(17,1500)

scala>
```

3. Crea un RDD donde la clave sea el userid y el valor sea una lista de ips a las que el userid se ha conectado (es decir, agrupar las IPs por userID). Ayúdate de la función `groupByKey()` para conseguirlo, de manera que el resultado final sea algo así: `Var rdd=logs.map(line => line.split(' ')).map(words => (words(2),0)).groupByKey().take(10)`

```
scala> var userips = logs.map(line => line.split(' ')).map(words => (words(2),words(0))).groupByKey()
userips: org.apache.spark.rdd.RDD[(String, Iterable[String])] = ShuffledRDD[16] at groupByKey at <console>:29

scala>

scala> userips.take(10)
res3: Array[(String, Iterable[String])] = Array((79844,CompactBuffer(136.132.254.160, 136.132.254.160, 53.251.68.51, 53.251.68.51)), (16669,CompactBuffer(23.137.191.64, 23.137.191.64)), (99640,CompactBuffer(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, 142.96.254.175, 142.96.254.175, 35.107.69.206, 35.107.69.206, 208.153.228.87, 208.153.228.87, 67.92.22.4, 67.92.22.4, 15.209.169.137, 15.209.169.137, 15.209.169.137, 15.209.169.137, 51.239.242.13, 51.239.242.13, 51.239.242.13, 18.76.240.35, 18.76.240.35, 107.125.111.173, 107.125.111.173, 245.141.100.16, 245.141.100.16, 215.150.177.226, 215.150.177.226, 215.150.177.226, 215.150.177.226, 39.175.103.131, 39.175.103.131, 102.17...), (102.17...))

scala>
```

Si te sobra tiempo prueba a mostrar el RDD resultante por pantalla de forma que tenga una estructura como la siguiente:

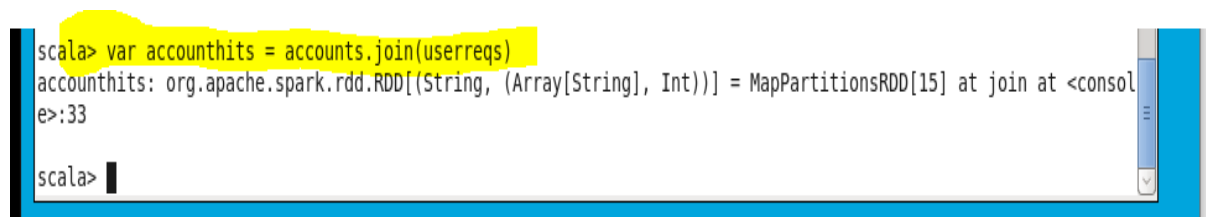
**B- Trabajo con todos los datos de la carpeta de logs:**  
**“/home/BIT/data/accounts.csv” Tareas a realizar**

1. Abre el fichero accounts.csv con el editor de texto que prefieras y estudia su contenido. Verás que el primer campo es el id del usuario, que corresponde con el id del usuario de 7 los logs del servidor web. El resto de campos corresponden con fecha, nombre, apellido, dirección, etc.

2. Haz un JOIN entre los datos de logs del ejercicio pasado y los datos de accounts.csv, de manera que se obtenga un conjunto de datos en el que la clave sea el userid y como valor tenga la información del usuario seguido del número de visitas de cada usuario. Los pasos a ejecutar son:

a. Haz un map() de los datos de accounts.csv de forma que la Clave sea el userid y el Valor sea toda la línea, incluido el userid. Obtendríamos algo de este tipo

b. Haz un JOIN del RDD que acabas de crear con el que creaste en el paso anterior que contenía (userid, nº visitas), de manera que quede algo como esto:  
i.

A screenshot of a Scala REPL (Read-Eval-Print Loop) window. The first line shows the command `scala> var accounthits = accounts.join(userregs)` with a yellow highlight. The second line shows the result: `accounthits: org.apache.spark.rdd.RDD[(String, (Array[String], Int))] = MapPartitionsRDD[15] at join at <console>:33`. The third line shows the prompt `scala>`  with a cursor.

```
scala> var accounthits = accounts.join(userregs)
accounthits: org.apache.spark.rdd.RDD[(String, (Array[String], Int))] = MapPartitionsRDD[15] at join at <console>:33
scala>
```

c. Crea un RDD a partir del RDD anterior, que contenga el userid, número de visitas, nombre y apellido de las 5 primeras líneas, para obtener una estructura como la siguiente (muestra más líneas de las requeridas): i. (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, ...]) 8

### C- Trabajo con más métodos sobre pares RDD Tareas a realizar

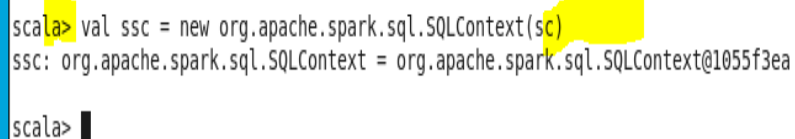
1. Usa `keyBy` para crear un RDD con los datos de las cuentas, pero con el código postal como clave (noveno campo del fichero `accounts.CSV`). Puedes buscar información sobre este método en la API online de Spark a.

2. Crea un RDD de pares con el código postal como la clave y una lista de nombres (Apellido, Nombre) de ese código postal como el valor. Sus lugares son el 5º y el 4º respectivamente. a. Si tienes tiempo, estudia la función `"mapValues()"` e intenta utilizarla para cumplir con el propósito de este ejercicio. b.

3. Ordena los datos por código postal y luego, para los primeros 5 códigos postales, muestra el código y la lista de nombres cuyas cuentas están en ese código postal. La salida sería parecida a esta: --- 85003 Jenkins,Thad Rick,Edward Lindsay,Ivy ... --- 85004 Morris,Eric Reiser,Hazel Gregg,Alicia PrestonElizabeth 9 10

Ejercicio: SparkSQL (JSON) (modulo 5.1) EL objetivo de este ejercicio es familiarizarnos con el uso de la herramienta SQL de Spark. Tareas a realizar

1. Crea un nuevo contexto `SQLContext` a.

A screenshot of a Scala REPL window. The first line shows the command `scala> val ssc = new org.apache.spark.sql.SQLContext(sc)` with a yellow highlight on the `scala>` prompt. The second line shows the response `ssc: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@1055f3ea`. The third line shows the prompt `scala>` with a cursor.

```
scala> val ssc = new org.apache.spark.sql.SQLContext(sc)
ssc: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@1055f3ea
scala>
```

2. Importa los implicits que permiten convertir RDDs en DataFrames a.

```
scala> import sqlContext.implicits._
import sqlContext.implicits._

scala> |
```

3. Carga el dataset “zips.json” que se encuentra en la carpeta de ejercicios de Spark en “/home/BIT/data/zips.json” que contiene datos de códigos postales de Estados Unidos. Puedes usar el comando “ssc.load(“ruta\_fichero”, “formato”)”. Tiene este aspecto:

```
scala> var zips = ssc.load("file:/home/cloudera/Desktop/data_spark/zips.json", "json")
warning: there were 1 deprecation warning(s); re-run with -deprecation for details
zips: org.apache.spark.sql.DataFrame = [_id: string, city: string, loc: array<double>, pop: bigint, state: string]

scala> |
```

4. Visualiza los datos con el comando “show()”. Tienes que ver una tabla con 5 columnas con un subconjunto de los datos del fichero. Puedes ver que el código postal es “\_id”, la ciudad es “city”, la ubicación “loc”, la población “pop” y el estado “state”. A.

```
scala> zips.show()
+-----+-----+-----+-----+-----+
|_id|city|loc|pop|state|
+-----+-----+-----+-----+
|01001|AGAWAM|[-72.622739, 42.0...|15338|MA|
|01002|CUSHMAN|[-72.51565, 42.37...|36963|MA|
|01005|BARRE|[-72.108354, 42.4...|4546|MA|
|01007|BELCHERTOWN|[-72.410953, 42.2...|10579|MA|
|01008|BLANDFORD|[-72.936114, 42.1...|1240|MA|
|01010|BRIMFIELD|[-72.188455, 42.1...|3706|MA|
|01011|CHESTER|[-72.988761, 42.2...|1688|MA|
|01012|CHESTERFIELD|[-72.833309, 42.3...|177|MA|
|01013|CHICOPEE|[-72.607962, 42.1...|23396|MA|
|01020|CHICOPEE|[-72.576142, 42.1...|31495|MA|
|01022|WESTOVER AFB|[-72.558657, 42.1...|1764|MA|
|01026|CUMMINGTON|[-72.905767, 42.4...|1484|MA|
|01027|MOUNT TOM|[-72.679921, 42.2...|16864|MA|
|01028|EAST LONGMEADOW|[-72.505565, 42.0...|13367|MA|
|01030|FEEDING HILLS|[-72.675077, 42.0...|11985|MA|
|01031|GILBERTVILLE|[-72.198585, 42.3...|2385|MA|
|01032|GOSHEN|[-72.844092, 42.4...|122|MA|
|01033|GRANBY|[-72.520001, 42.2...|5526|MA|
|01034|TOLLAND|[-72.908793, 42.0...|1652|MA|
|01035|HADLEY|[-72.571499, 42.3...|4231|MA|
+-----+-----+-----+-----+
only showing top 20 rows
```

5. Obtén los códigos postales cuya población es superior a 10000 usando la api de DataFrames a.

```
scala> zips.filter(zips("pop") > 10000).collect()
res4: Array[org.apache.spark.sql.Row] = Array([01001,AGAWAM,WrappedArray(-72.622739, 42.070206),15338,MA], [01002,CUSHMAN,WrappedArray(-72.51565, 42.377017),36963,MA], [01007,BELCHERTOWN,WrappedArray(-72.410953, 42.275103),10579,MA], [01013,CHICOPEE,WrappedArray(-72.607962, 42.162046),23396,MA], [01020,CHICOPEE,WrappedArray(-72.576142, 42.176443),31495,MA], [01027,MOUNT TOM,WrappedArray(-72.679921, 42.264319),16864,MA], [01028,EAST LONGMEAD,WrappedArray(-72.505565, 42.067203),13367,MA], [01030,FEEDING HILLS,WrappedArray(-72.675077, 42.07182),11985,MA], [01040,HOLYOKE,WrappedArray(-72.626193, 42.202007),43704,MA], [01056,LUDLOW,WrappedArray(-72.471012, 42.172823),18820,MA], [01060,FLORENCE,WrappedArray(-72.654245, 42.324662),27939,MA], [01075,SOUTH HADLEY,WrappedArray(-72.581137, 42.23...
scala>
```

6. Guarda esta tabla en un fichero temporal para poder ejecutar SQL contra ella.

a.

```
scala> zips.registerTempTable("zips")

scala>
```

7. Realiza la misma consulta que en el punto 5, pero esta vez usando SQL a.

```
scala> ssc.sql("SELECT * FROM zips WHERE pop > 10000").collect()
res6: Array[org.apache.spark.sql.Row] = Array([01001,AGAWAM,WrappedArray(-72.622739, 42.070206),15338,MA], [01002,CUSHMAN,WrappedArray(-72.51565, 42.377017),36963,MA], [01007,BELCHERTOWN,WrappedArray(-72.410953, 42.275103),10579,MA], [01013,CHICOPEE,WrappedArray(-72.607962, 42.162046),23396,MA], [01020,CHICOPEE,WrappedArray(-72.576142, 42.176443),31495,MA], [01027,MOUNT TOM,WrappedArray(-72.679921, 42.264319),16864,MA], [01028,EAST LONGMEAD,WrappedArray(-72.505565, 42.067203),13367,MA], [01030,FEEDING HILLS,WrappedArray(-72.675077, 42.07182),11985,MA], [01040,HOLYOKE,WrappedArray(-72.626193, 42.202007),43704,MA], [01056,LUDLOW,WrappedArray(-72.471012, 42.172823),18820,MA], [01060,FLORENCE,WrappedArray(-72.654245, 42.324662),27939,MA], [01075,SOUTH HADLEY,WrappedArray(-72.581137, 42.23...
scala>
```

8. Usando SQL, obtén la ciudad con más de 100 códigos postales

```
scala> ssc.sql("SELECT city FROM zips GROUP by city having count(*)>100").show()
+-----+
|  city |
+-----+
|HOUSTON|
+-----+
```

9. Usando SQL, obtén la población del estado de Wisconsin (WI) 10. Usando SQL, obtén los 5 estados más poblados { "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338, "state" : "MA", "\_id" : "01001" } // [CA,29760021] // [NY,17990455] // [TX,16986510] // [FL,12937926] // [PA,11881643] 13

```
scala> ssc.sql("SELECT state, SUM(pop) AS POPULATION FROM zips GROUP BY state ORDER by SUM(pop) DESC").show()
+-----+
|state|POPULATION|
+-----+
|CA|29760021|
|NY|17990455|
|TX|16986510|
|FL|12937926|
|PA|11881643|
|IL|11430602|
|OH|10847115|
|MI|9295297|
|NJ|7730188|
|NC|6628637|
|GA|6478216|
|VA|6187358|
|MA|6016425|
|IN|5544159|
|MO|5114343|
|WI|4891769|
|TN|4876457|
|WA|4866692|
|MD|4781468|
|MN|4375099|
+-----+
only showing top 20 rows

scala> █
```

**Ejercicio: SparkSQL (hive) (módulo 5.2)** El objetivo de este ejercicio es familiarizarnos con el uso de SparkSQL para acceder a tablas en Hive, dado que es una herramienta ampliamente extendida en entornos analíticos. Tareas a realizar

1. Abrir una tercera terminal y logaros como root, con pass cloudera. Ejecutar este comando: "cp /usr/lib/hive/conf/hive-site.xml /usr/lib/spark/conf/". Para más información podéis consultar esta web "<https://community.cloudera.com/t5/Advanced-Analytics-Apache-Spark/how-toaccess-the-hive-tables-from-spark-shell/td-p/36609> "

2. En un terminal, arrancar el Shell de Hive y echar un vistazo a las bases de datos y tablas que hay en cada una de ellas.

3. En otro terminal aparte, arrancar el Shell de Spark, y a través de SparkSQL crear una base de datos y una tabla con dos o tres columnas. Si no creamos la base de datos, las tablas que creamos se guardarán dentro de la bbdd "default" de Hive. Antes de esto, crear una variable de contexto para trabajar con Spark a través de Hive. bbdd: hivespark tabla: empleados columnas: id INT, name STRING, age INT config table: FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' a. b. c.



4. Crear un fichero “/home/cloudera/empleado.txt” que contenga los siguientes datos de esta manera dispuestos

5. Aprovechando la estructura de la tabla que hemos creado antes, usando SparkSQL, subid los datos del fichero “/home/cloudera/empleado.txt” a la tabla hive, usando como la sintaxis de HiveQL como vimos en el curso de Hive (LOAD DATA LOCAL INPATH). a. 14

6. Ejecutad cualquier consulta en los terminales de Hive y Spark para comprobar que todo funciona y se devuelven los mismos datos. En el terminal de Spark usad el comando “show()” para mostrar los datos. a. b. c.

**Ejercicio: SparkSQL (DataFrames) (módulo 5.3) El objetivo de este ejercicio es familiarizarnos un poco más con la API de DataFrames.**

1. Creamos un contexto SQL a.

```
scala> var ssc = new org.apache.spark.sql.SQLContext(sc)
ssc: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@261820a8

scala> import sqlContext.implicits._
import sqlContext.implicits._

scala> import org.apache.spark.sql.Row
import org.apache.spark.sql.Row

scala> import org.apache.spark.sql.types.{StructType, StructField, StringType}
import org.apache.spark.sql.types.{StructType, StructField, StringType}

scala> █
```

2. Importa los implicits que permiten convertir RDDs en DataFrames y Row a. import sqlContext.implicits.\_ b. import org.apache.spark.sql.Row c. import org.apache.spark.sql.types{StructType, StructField, StringType}

```
scala> var ssc = new org.apache.spark.sql.SQLContext(sc)
ssc: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@261820a8

scala> import sqlContext.implicits._
import sqlContext.implicits._

scala> import org.apache.spark.sql.Row
import org.apache.spark.sql.Row

scala> import org.apache.spark.sql.types.{StructType, StructField, StringType}
import org.apache.spark.sql.types.{StructType, StructField, StringType}

scala> █
```

3. Creamos una variable con la ruta al fichero “/home/BIT/data/DataSetPartidos.txt”. las líneas tienen el siguiente formato # idPartido::temporada::jornada::EquipoLocal::EquipoVisitante::golesLocal::golesVisitante::fecha::timestamp a.

```
scala> val ruta_datos="file:/home/cloudera/Desktop/DataSetPartidos.txt"
ruta_datos: String = file:/home/cloudera/Desktop/DataSetPartidos.txt

scala>
```

4. Leemos el contenido del archivo en una variable a.

```
scala> val datos = sc.textFile(ruta_datos)
datos: org.apache.spark.rdd.RDD[String] = file:/home/cloudera/Desktop/DataSetPartidos.txt MapPartition
SRDD[11] at textFile at <console>:36

scala>
```

5. Creamos una variable que contenga el esquema de los datos a.

```
scala> val schemaString = "idPartido::temporada::jornada::EquipoLocal::EquipoVisitante::golesLocal::golesVisitante::fecha::ti
<console>:1: error: unclosed string literal
    val schemaString = "idPartido::temporada::jornada::EquipoLocal::EquipoVisitante::golesLocal::golesVisitante::fecha::ti
                        ^

scala> val schemaString = "idPartido::temporada::jornada::EquipoLocal::EquipoVisitante::golesLocal::golesVisitante::fecha::ti
mestamp"
schemaString: String = idPartido::temporada::jornada::EquipoLocal::EquipoVisitante::golesLocal::golesVisitante::fecha::timest
amp

scala>
```

6. Generamos el esquema basado en la variable que contiene el esquema de los datos que acabamos de crear a.

```
scala> val schema = StructType(schemaString.split("::").map(fieldName => StructField(fieldName, StringType, true)))
schema: org.apache.spark.sql.types.StructType = StructType(StructField(idPartido,StringType,true), StructField(temporada,StringType,true), StructField(jornada,StringType,true), StructField(EquipoLocal,StringType,true), StructField(EquipoVisitante,StringType,true), StructField(golesLocal,StringType,true), StructField(golesVisitante,StringType,true), StructField(fecha,StringType,true), StructField(timestamp,StringType,true))

scala>
```

7. Convertimos las filas de nuestro RDD a Rows a.

```
scala> val rowRDD = datos.map(_._split(":")).map(p => Row(p(0), p(1),p(2),p(3),p(4),p(5),p(6),p(7),p(8).trim))
rowRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[13] at map at <console>:38

scala> █
```

## 8. Aplicamos el Schema al RDD a.

```
scala> val partidosDataFrame = sqlContext.createDataFrame(rowRDD, schema)
partidosDataFrame: org.apache.spark.sql.DataFrame = [idPartido: string, temporada: string, jornada: string, EquipoLocal: string, EquipoVisitante: string, golesLocal: string, golesVisitante: string, fecha: string, timestamp: string]

scala> █
```

## 9. Registramos el DataFrame como una Tabla a.

```
scala> partidosDataFrame.registerTempTable("partidos");

scala> val results = sqlContext.sql("SELECT temporada, jornada FROM partidos")
results: org.apache.spark.sql.DataFrame = [temporada: string, jornada: string]

scala> results.show()
+-----+-----+
|temporada|jornada|
+-----+-----+
| 1970-71|      1|
| 1970-71|      1|
| 1970-71|      1|
| 1970-71|      1|
| 1970-71|      1|
| 1970-71|      1|
| 1970-71|      1|
| 1970-71|      1|
| 1970-71|      1|
| 1970-71|      2|
| 1970-71|      2|
| 1970-71|      2|
| 1970-71|      2|
| 1970-71|      2|
| 1970-71|      2|
| 1970-71|      2|
| 1970-71|      2|
| 1970-71|      3|
| 1970-71|      3|
| 1970-71|      3|
+-----+-----+
only showing top 20 rows

scala> █
```

## 10. Ya estamos listos para hacer consultas sobre el DF con el siguiente formato a. b.



12. Ejercicio: ¿Cuál es el record de goles como visitante en una temporada del Oviedo? a. b.

```
scala> val recordOviedo = sqlContext.sql("select sum(golesVisitante) as goles, temporada from partidos where equipoVisitante='Real Oviedo' group by temporada order by goles desc")
recordOviedo: org.apache.spark.sql.DataFrame = [goles: double, temporada: string]

scala> recordOviedo.take(1)
res19: Array[org.apache.spark.sql.Row] = Array([25.0,1995-96])

scala> █
```

13. ¿Quién ha estado más temporadas en 1 Division Sporting u Oviedo? a. b.

```
scala> val temporadasOviedo = sqlContext.sql("select count(distinct(temporada)) from partidos where equipoLocal='Real Oviedo' or equipoVisitante='Real Oviedo' ")
temporadasOviedo: org.apache.spark.sql.DataFrame = [_c0: bigint]

scala> val temporadasSporting = sqlContext.sql("select count(distinct(temporada)) from partidos where equipoLocal='Sporting de Gijon' or equipoVisitante='Sporting de Gijon' ")
temporadasSporting: org.apache.spark.sql.DataFrame = [_c0: bigint]

scala> █
```

**Ejercicio: Spark Streaming I (modulo 6.1)** El objetivo de este ejercicio es el de iniciarnos en el uso de Spark Streaming y observar sus cualidades. Para ello generaremos un script en un terminal que contará las palabras que introduzcamos en otra terminal a modo de streaming simulado. Tareas a realizar 19

1. Visita en la web la documentación de Spark <https://spark.apache.org/docs/1.5.2/streaming-programming-guide.html> y familiarízate con el ejercicio. El objetivo es hacer lo mismo que pone en la web en el apartado “A Quick Example”

2. Tomate un tiempo para navegar por la web y explorar todo lo que puede ofrecerte. Cuando lo consideres, comienza el ejercicio:

3. Abre un terminal nuevo y escribe el siguiente comando: “nc -lkv 4444”, que hace que todo lo que escribas se envíe al puerto 4444

4. Inicia un nuevo terminal y arranca el Shell de Spark en modo local con al menos 2 threads, necesarios para ejecutar este ejercicio: “spark-shell --master local[2]”


5. Por otro lado, accede al fichero “/usr/lib/spark/conf/log4j.properties”, y edítalo para poner el nivel de log a ERROR, de modo que en tu Shell puedas ver con claridad el streaming de palabras contadas devuelto por tu script.

6. Importa las clases necesarias para trabajar con Spark Streaming

a. import org.apache.spark.streaming.StreamingContext

b. import org.apache.spark.streaming.StreamingContext.\_

c. import org.apache.spark.streaming.Seconds



```
scala> import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext

scala> import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.StreamingContext._

scala> import org.apache.spark.streaming.Seconds
import org.apache.spark.streaming.Seconds

scala> var ssc = new StreamingContext(sc,Seconds(5))
ssc: org.apache.spark.streaming.StreamingContext = org.apache.spark.streaming.St
reamingContext@940fa7d

scala> var mystream = ssc.socketTextStream("localhost",4444)
mystream: org.apache.spark.streaming.dstream.ReceiverInputDStream[String] = org.
apache.spark.streaming.dstream.SocketInputDStream@28f5f4f3

scala>
```

7. Crea un SparkContext con una duración de 5 segundos a. var ssc = new StreamingContext(sc,Seconds(5))

```
scala> import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext

scala> import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.StreamingContext._

scala> import org.apache.spark.streaming.Seconds
import org.apache.spark.streaming.Seconds

scala> var ssc = new StreamingContext(sc,Seconds(5))
ssc: org.apache.spark.streaming.StreamingContext = org.apache.spark.streaming.St
reamingContext@940fa7d

scala> var mystream = ssc.socketTextStream("localhost",4444)
mystream: org.apache.spark.streaming.dstream.ReceiverInputDStream[String] = org.
apache.spark.streaming.dstream.SocketInputDStream@28f5f4f3

scala> █
```

8. Crea un DStream para leer texto del puerto que pusiste en el comando “nc”, especificando el hostname de nuestra máquina, que es “quickstart.cloudera” a. `var mystream = ssc.socketTextStream("localhost",4444)`

```
scala> import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext

scala> import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.StreamingContext._

scala> import org.apache.spark.streaming.Seconds
import org.apache.spark.streaming.Seconds

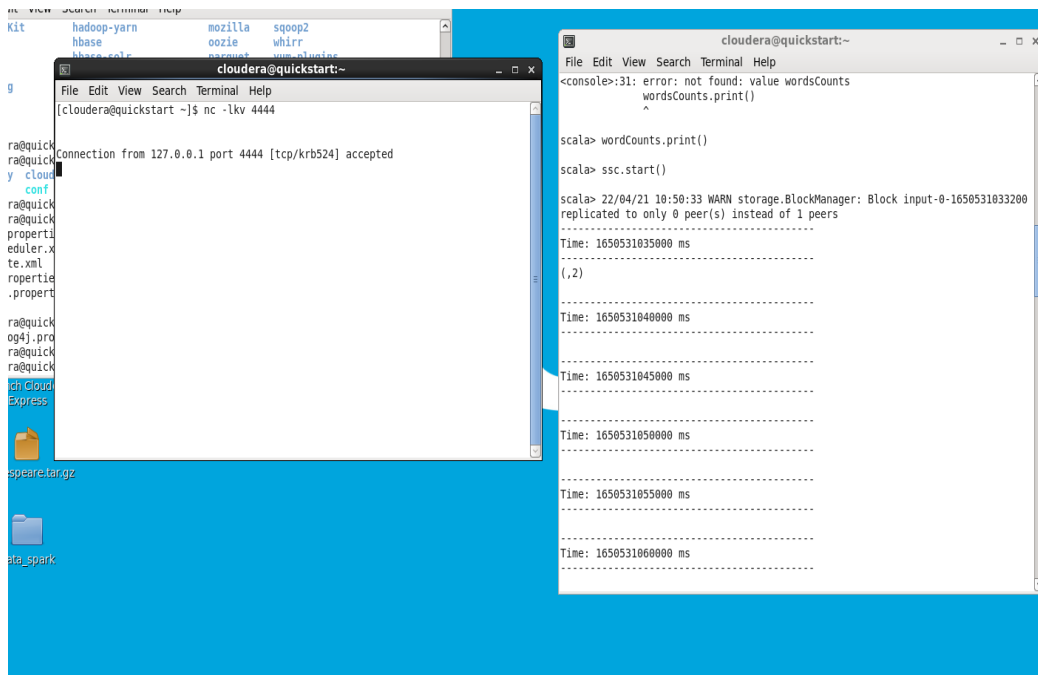
scala> var ssc = new StreamingContext(sc,Seconds(5))
ssc: org.apache.spark.streaming.StreamingContext = org.apache.spark.streaming.St
reamingContext@940fa7d

scala> var mystream = ssc.socketTextStream("localhost",4444)
mystream: org.apache.spark.streaming.dstream.ReceiverInputDStream[String] = org.
apache.spark.streaming.dstream.SocketInputDStream@28f5f4f3

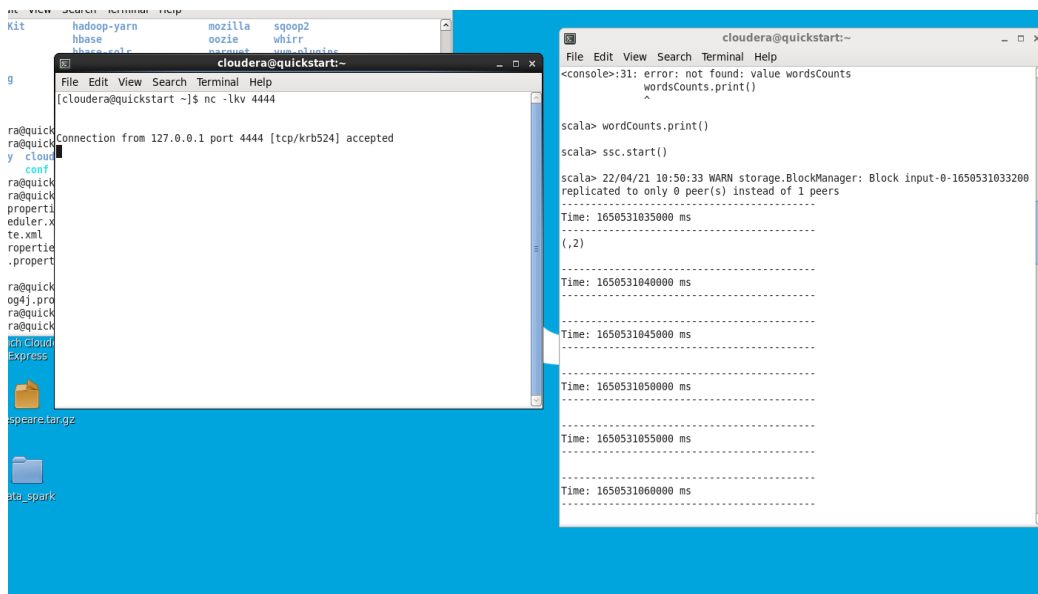
scala> █
```

9. Crea un MapReduce, como vimos en los apuntes, para contar el número de palabras que aparecen en cada Stream a. `var words = mystream.flatMap(line => line.split("\\W"))` b. `var wordCounts = words.map(x => (x, 1)).reduceByKey((x,y) => x+y)`

10. Imprime por pantalla los resultados de cada batch a. `wordCounts.print()`



11. Arranca el Streaming Context y llama a awaitTermination para esperar a que la tarea termine a. ssc.start() b. ssc.awaitTermination() 20



12. Deberías ver algo de este tipo:

13. Una vez hayas acabado, sal del Shell y del terminal donde ejecutaste el comando “nc” haciendo un CNTRL+C

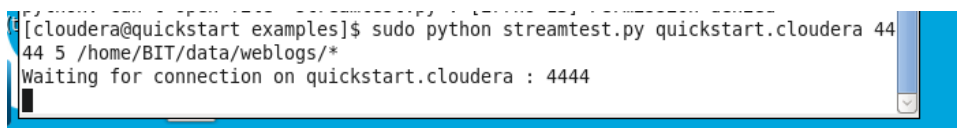
14. Para ejecutarlo desde un script: spark-shell --master local[2] -i prueba.scala



## Ejercicio: Spark Streaming II (modulo 6.2)

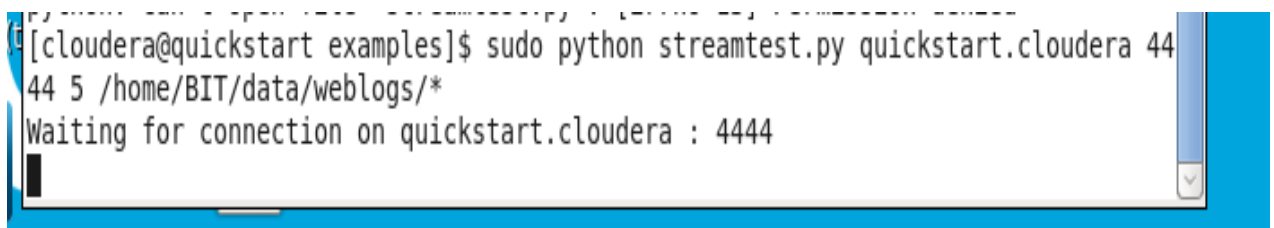
El objetivo de este ejercicio es seguir familiarizándonos con el concepto de DStream. Para ello vamos a simular accesos a web (los de la carpeta weblogs) y vamos a trabajar con ellos como si estuvieran ocurriendo en tiempo real, creando una aplicación Spark Streaming que capture aquellos que cumplan una característica específica que indicaremos más abajo. Para ayudarnos de ello, se ha creado un script en Python que podéis encontrar en la carpeta de ejercicios de Spark. La primera tarea es copiar este script en la ruta “/home/BIT/examples/streamtest.py” (o en otra que tú elijas) que lo que hace es leer los ficheros contenidos en weblogs y simular un streaming de ellos (parecido a lo que hicimos con el comando “nc” en el ejercicio pasado, pero de manera automática y sobre un conjunto de ficheros) Tareas a realizar

1. Abre un nuevo terminal, sitúate en la ruta “/home/BIT/examples” y ejecuta el script Python mencionado arriba de la siguiente manera



```
[cloudera@quickstart examples]$ sudo python streamtest.py quickstart.cloudera 44
44 5 /home/BIT/data/weblogs/*
Waiting for connection on quickstart.cloudera : 4444
```

2. Copia el fichero “StreamingLogs.scalaspark” situado en la carpeta de ejercicios de Spark en “/home/BIT/stubs/StreamingLogs.scalaspark” y familiarízate con el contenido. El objetivo es ejecutar en el Shell cada una de las líneas que contiene más las que vamos a programar para este ejercicio.



```
[cloudera@quickstart examples]$ sudo python streamtest.py quickstart.cloudera 44
44 5 /home/BIT/data/weblogs/*
Waiting for connection on quickstart.cloudera : 4444
```

3. Abre un nuevo terminal y arranca el Shell de Spark con al menos dos threads, como en el caso anterior (tal y como pone en el fichero). A continuación ejecuta los imports y crea un nuevo StreamingContext con intervalos de un segundo.

```
scala> import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext

scala> import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.StreamingContext._

scala> import org.apache.spark.streaming.Seconds
import org.apache.spark.streaming.Seconds

scala> var ssc=new StreamingContext(sc, seconds(1))
<console>:32: error: not found: value seconds
      var ssc=new StreamingContext(sc, seconds(1))
                                   ^
scala> var ssc=new StreamingContext(sc, Seconds(1))
ssc: org.apache.spark.streaming.StreamingContext = org.apache.spark.streaming.St
reamingContext@33369f6e

scala> █
```

4. Crea un DStream de logs cuyo host es “quickstart.cloudera” (también podéis usar “localhost”) y cuyo puerto es “4444” (lo mismo que hemos indicado por parámetro al script Python anterior) a.

```
scala> val dstream=ssc.socketTextStream("quickstart.cloudera",4444)
dstream: org.apache.spark.streaming.dstream.ReceiverInputDStream[String] = org.a
pache.spark.streaming.dstream.SocketInputDStream@38bd7174

scala> █
```

5. Filtra las líneas del Stream que contengan la cadena de caracteres “KBDOC”

```
scala> val lineas = dstream.filter(x=>x.contains("KBDOC"))
lineas: org.apache.spark.streaming.dstream.DStream[String] = org.apache.spark.st
reaming.dstream.FilteredDStream@7822210c

scala> █
```

6. a.

```
scala> val numero = lineas.foreachRDD(rdd => rdd.count())
numero: Unit = ()

scala> █
```

7. Para cada RDD, imprime el número de líneas que contienen la cadena de caracteres indicada. Para ello, puedes usar la función “foreachRDD()”. a.

8. Guarda el resultado del filtrado en un fichero de texto en sistema de archivos local (créate una ruta en /home/cloudera/...)de la máquina virtual, no en hdfs y revisa el resultado al acabar el ejercicio. a.

9. Para arrancar el ejercicio ejecuta los comandos `start()` y `awaitTermination()` a.  
EXTRA 22

10. Si te ha sobrado tiempo, prueba a ampliar el ejercicio de la siguiente manera

11. Cada dos segundos, muestra el número de peticiones de KBDQC tomando ventanas de 10 segundos. Ayúdate de la función “`countByWindow`” y recuerda el uso del checkpoint necesario para utilizar funciones de ventana. La idea es replicar el código hecho hasta ahora y añadir las líneas que hacen falta para cumplir con el objetivo marcado en este punto. a.

12. Inicia el `StreamingContext` y espera a que termine.