

Técnicas para Big Data

Clase 10 - Scala y Apache Spark

Hasta ahora

Conocemos técnicas para procesar consultas que nos pueden ayudar ante grandes cantidades de datos

Conocemos distintos motores que nos pueden ayudar ante distintos casos de uso

Conocemos técnicas distribuidas para procesar datos (Sharding, Map - Reduce)

Hoy

Apache Spark. Un *framework* para procesar datos de manera distribuida

Spark

Spark

Apache Spark es un motor para procesar grandes cantidades de datos

Está programado en Scala

Puede ser usado junto a Scala, Python, Java o R

Puede correr sobre *clusters* y distintos DBMS

Scala

Scala

Usaremos Scala como lenguaje para usar Apache Spark (y más adelante GraphX)

Es un lenguaje orientado a objetos y funcional

Corre sobre la JVM (al igual que Java)

Posee una consola interactiva (Scala REPL)

Scala

Ejemplos Scala REPL

Scala

Ejemplos Scala REPL

```
scala> println("Hello World!")  
Hello World!
```

```
scala> val x = 5  
x: Int = 5
```

```
scala> println(x * 2)  
10
```

Scala

Valores y variables

Scala

Valores y variables

- Para declarar valores inmutables usamos **val**
- Para declarar variables (que eventualmente pueden cambiar) usamos **var**

Scala

If - else

Scala

If - else

```
val x = 10;  
val y = 5;  
if (x < y) {  
    println("x es menor que y")  
} else {  
    println("x es mayor o igual que y")  
}
```

Scala

While

Scala

While

```
var x = 0;
while (x < 10) {
  println(s"El valor de x es $x")
  x += 1
}
```

Scala

s'' - string interpolation

Scala

s" - string interpolation

En el ejemplo anterior usamos *string interpolation*

```
scala> s"El valor de x es $x"
```

Se antepone una **s** al string y luego puedo hacer uso de la variable **x** dentro del string

Scala

Arreglos, tuplas y listas

Scala

Arreglos, tuplas y listas

```
val person = (1, "Cristian")  
println(s"El nombre de la persona es $person._2")
```

```
val numbers = Array(1, 2, 4, 8, 16)  
println(s"El tercer elemento es ${numbers(2)}")
```

```
val fruits = List("Apple", "Orange")  
val more_fruits = fruits :+ "Banana"  
println(more_fruits)
```

Scala

For y foreach

Scala

For y foreach

```
val fruits = Array("Apple", "Orange", "Banana", "Blueberry")
for (fruit <- fruits) {
  println(fruit)
}
```

// Imprime cada fruta junto al largo del string

```
fruits.foreach(fruit => println(s"$fruit ${fruit.size}"))
```

/ El output del foreach es*

Apple 5

Orange 6

Banana 6

Blueberry 9

**/*

Scala

Foreach, map y reduce

Scala

Foreach, map y reduce

Son funciones de los iterables

Notamos que **foreach** tomaba una función y la ejecutaba para cada elemento

map toma una función que convierte cada elemento en uno nuevo, retornando una nueva lista

reduce toma los elementos de a dos y los combina en uno único

Scala

Map y reduce

Scala

Map y reduce

```
val numbers = Array(1, 2, 3, 4, 5)

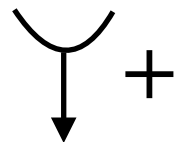
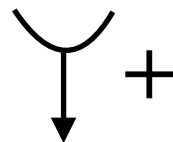
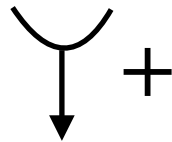
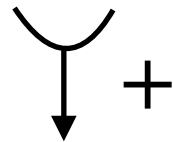
val squares = numbers.map(number => math.pow(number, 2))
squares.foreach(square => println(square))

val sum_squares = squares.reduce((a, b) => a + b)
println(s"La suma de todos los cuadrados es: $sum_squares")

/* El output del programa es:
  1.0
  4.0
  9.0
  16.0
  25.0
  La suma de todos los cuadrados es: 55.0
  */
```

Scala

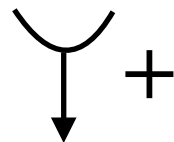
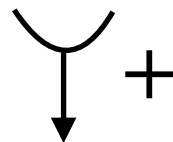
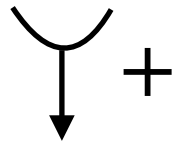
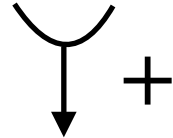
Explicación reduce



Scala

Explicación reduce

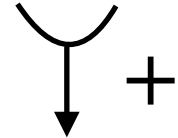
[1, 4, 9, 16, 25]



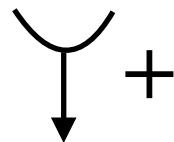
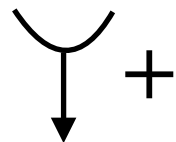
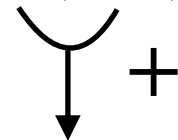
Scala

Explicación reduce

[1, 4, 9, 16, 25]



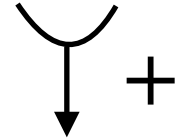
[5, 9, 16, 25]



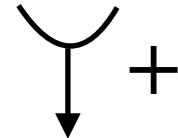
Scala

Explicación reduce

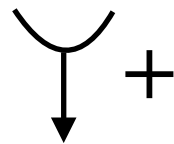
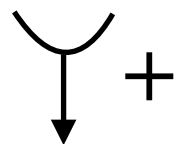
[1, 4, 9, 16, 25]



[5, 9, 16, 25]



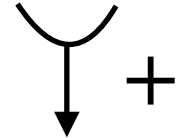
[14, 16, 25]



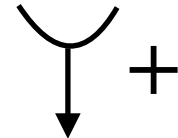
Scala

Explicación reduce

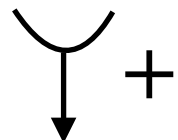
[1, 4, 9, 16, 25]



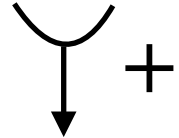
[5, 9, 16, 25]



[14, 16, 25]



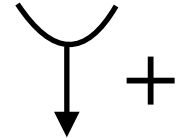
[30, 25]



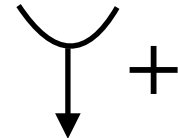
Scala

Explicación reduce

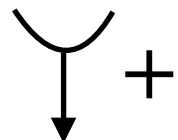
[1, 4, 9, 16, 25]



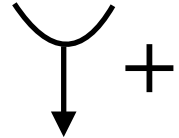
[5, 9, 16, 25]



[14, 16, 25]



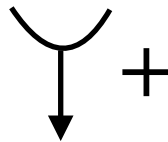
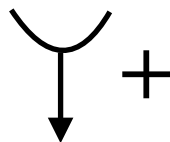
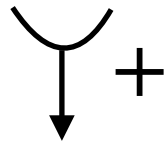
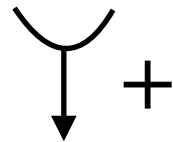
[30, 25]



[55]

Scala

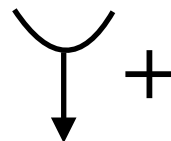
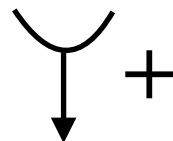
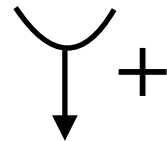
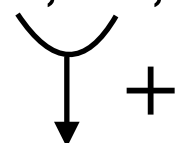
Explicación reduce



Scala

Explicación reduce

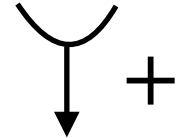
[1, 2, 3, 4, 5]



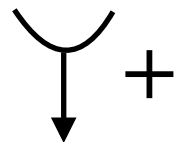
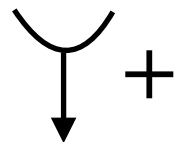
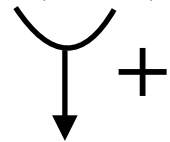
Scala

Explicación reduce

[1, 2, 3, 4, 5]



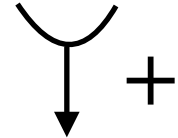
[3, 3, 4, 5]



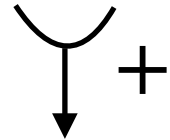
Scala

Explicación reduce

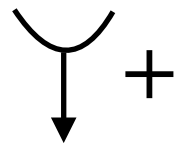
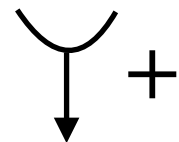
[1, 2, 3, 4, 5]



[3, 3, 4, 5]

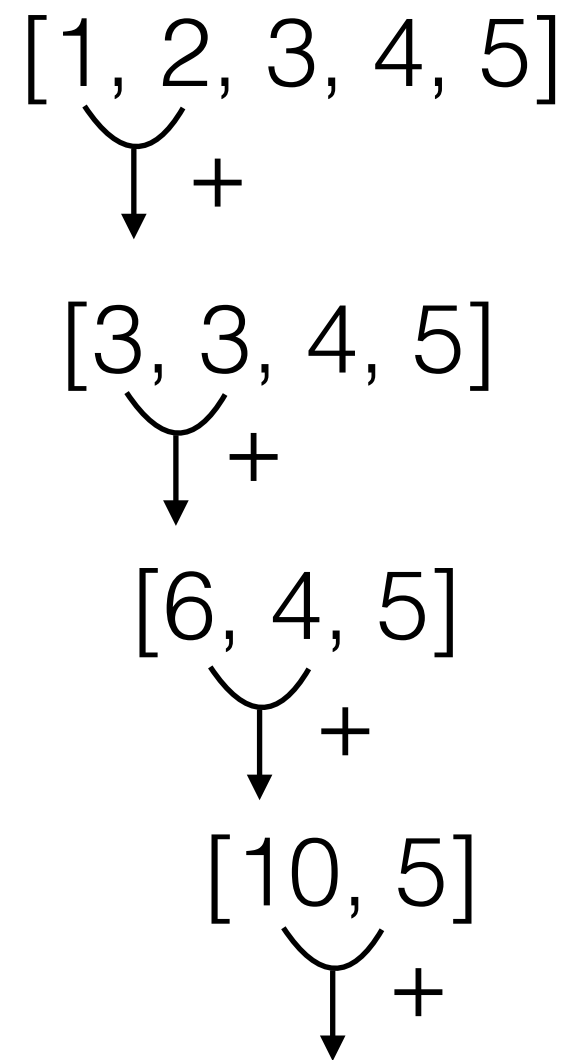


[6, 4, 5]



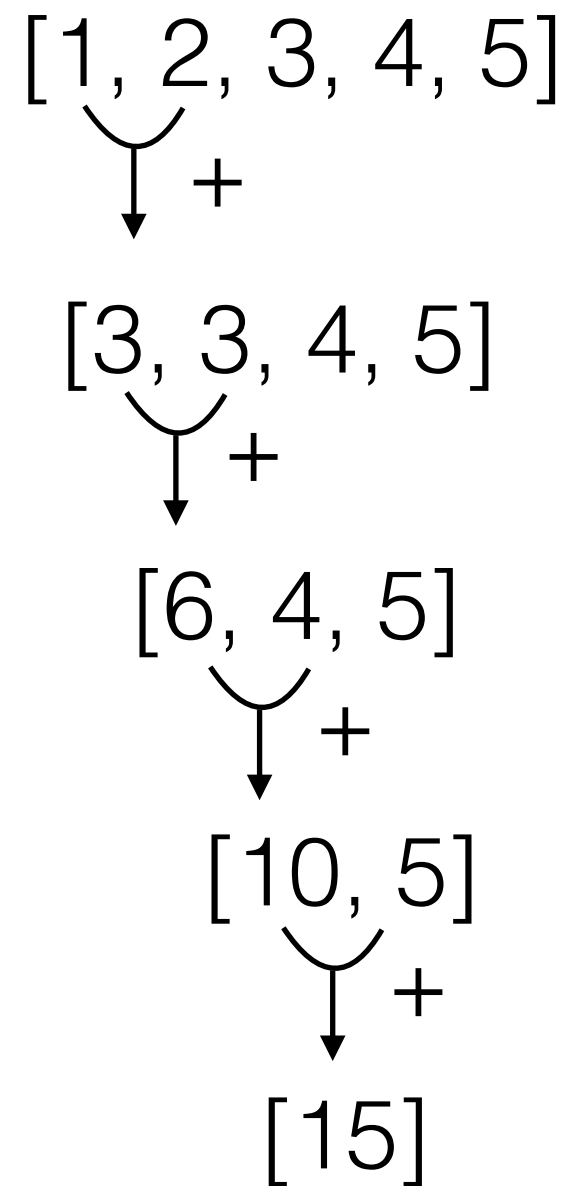
Scala

Explicación reduce



Scala

Explicación reduce



Scala

Classes

Scala

Classes

```
class Point(var x: Int, var y: Int) {  
    // Unit es analogo a void  
  
    def move(dx: Int, dy: Int): Unit = {  
        x = x + dx  
        y = y + dy  
    }  
  
    override def toString: String =  
        s"($x, $y)"  
}
```

Scala

Scala

¿Cómo corremos un programa de Scala?

Existen varias formas de hacer esto, pero en general es mejor usar un IDE, en este caso recomendamos IntelliJ (la versión *premium* utilizando el correo UAI); también podemos usar Eclipse

También podemos desarrollar por consola + editor de texto

SBT

SBT

Para correr Scala en general usamos SBT (similar a los virtual-env de Python)

Al instalar IntelliJ, debemos instalar el *plugin* de Scala; SBT viene incorporado

Nos ayuda a manejar y consolidar las librerías

Tutorial SBT IntelliJ: <https://docs.scala-lang.org/getting-started/intellij-track/building-a-scala-project-with-intellij-and-sbt.html>

Spark

Spark

Apache Spark tiene APIs en muchos lenguajes, pero los principales son Scala y Python

Ya que Spark está programado en Scala, es preferible usarlo con este lenguaje (no todos los módulos son accesibles desde PySpark)

Futuro tema de presentación: generar un tutorial completo de PySpark

Spark

Configuración

Spark

Configuración

La forma más sencilla de usar Spark es descargar la `spark-shell`

Nos dirigimos a la carpeta `bin` y corremos `./spark-shell`

Esta consola permite comandos en Scala y propios de Spark

Descarga: <https://spark.apache.org/downloads.html>

Spark

Contar palabras de un archivo

Spark

Contar palabras de un archivo

```
scala> sc
```

```
scala> val textFile = sc.textFile("/path/to/file.txt")
```

```
scala> val counts = textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey((a, b) => a + b)
```

```
scala> counts.saveAsTextFile("/path/to/output")
```

Spark

Contar palabras de un archivo

Spark

Contar palabras de un archivo

El primer comando comprueba nuestro contexto de Spark

El segundo comando carga el archivo al que le queremos contar las palabras

Spark

Contar palabras de un archivo

Spark

Contar palabras de un archivo

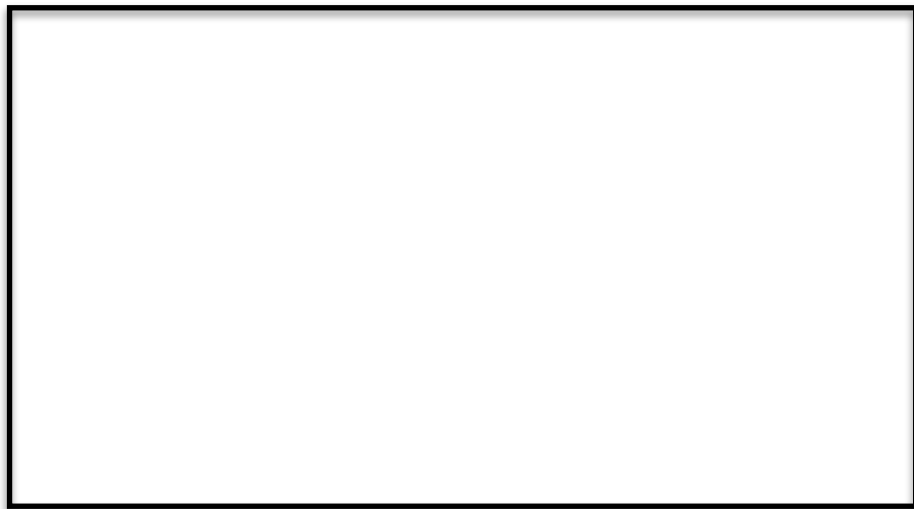
El comando:

```
textFile.flatMap(line => line.split(" "))
```

Separa el archivo en distintas lineas, y para cada linea separa cada palabra, retornando un arreglo con cada palabra del archivo

Spark

Contar palabras de un archivo



["Este", "es", "un",
"archivo", "de",
"ejemplo", "no",
"es", "un",
"archivo", "muy",
"extenso"]

Spark

Contar palabras de un archivo

Este es un archivo
de ejemplo no es
un archivo muy
extenso



["Este", "es", "un",
"archivo", "de",
"ejemplo", "no",
"es", "un",
"archivo", "muy",
"extenso"]

Spark

Contar palabras de un archivo

`["Este", "es",
"un", ...]` \longrightarrow `[("Este", 1),
("es", 1),
("un", 1), ...]`

Spark

Contar palabras de un archivo

El comando:

```
map(word => (word, 1))
```

transforma la lista en una lista de tuplas con las mismas palabras acompañadas de un 1

`["Este", "es", "un", ...]` \longrightarrow `[("Este", 1), ("es", 1), ("un", 1), ...]`

Spark

Contar palabras de un archivo

```
“Este”: [1]  
“es”: [1, 1]  
“un”: [1, 1]
```

...



```
(“Este”, 1)  
(“es”, 2)  
(“un”, 2)
```

...

Spark

Contar palabras de un archivo

El comando:

```
reduceByKey((a, b) => a + b)
```

Agrupar por cada palabra enviando los valores a un arreglo al que le haré **reduce**

```
"Este": [1]  
"es": [1, 1]  
"un": [1, 1]
```

...



```
("Este", 1)  
("es", 2)  
("un", 2)
```

...

Spark

Contar palabras de un archivo

Spark

Contar palabras de un archivo

El comando:

```
counts.saveAsTextFile("/path/to/output")
```

Crea una carpeta que contiene el output

Spark

Ejecutar un archivo en la consola

Spark

Ejecutar un archivo en la consola

Podemos correr un programa en la consola de Spark con el comando **:load**

```
:load /path/to/ExampleSpark.scala
```

Spark

Programando en Scala + SBT

Spark

Programando en Scala + SBT

Para hacer un programa de Scala + SBT que corra procedimientos de Apache Spark debemos añadir la versión correspondiente de Spark al archivo **build.sbt**

Spark

Programando en Scala + SBT

Para hacer un programa de Scala + SBT que corra procedimientos de Apache Spark debemos añadir la versión correspondiente de Spark al archivo **build.sbt**

```
// https://mvnrepository.com/artifact/org.apache.spark/  
spark-core  
libraryDependencies += "org.apache.spark" %% "spark-core"  
% "3.1.2"
```

Spark

Programando en Scala + SBT

Para hacer un programa de Scala + SBT que corra procedimientos de Apache Spark debemos añadir la versión correspondiente de Spark al archivo **build.sbt**

```
// https://mvnrepository.com/artifact/org.apache.spark/  
spark-core  
libraryDependencies += "org.apache.spark" %% "spark-core"  
% "3.1.2"
```

En este caso estamos cargando la versión 3.1.2 de Spark; ojo que esta versión es compatible con Scala 2.12

Spark

Contar palabras de un archivo

Spark

Contar palabras de un archivo

Ahora vamos a crear un programa en Scala que haga esto mismo:

Spark

Contar palabras de un archivo

Ahora vamos a crear un programa en Scala que haga esto mismo:

```
import org.apache.spark.{SparkConf, SparkContext}

object ExampleSpark extends App{

    val conf = new SparkConf().setAppName("TestApp").setMaster("local")
    val sc = new SparkContext(conf)
    val textFile = sc.textFile("/path/to/example.txt")
    val counts = textFile.flatMap(line => line.split(" ")).map(word => (word,
1)).reduceByKey((a, b) => a + b)
    counts.saveAsTextFile("/path/to/example-output")

}
```

Spark

Contar palabras de un archivo

Spark

Contar palabras de un archivo

Notamos que ahora tenemos que definir nuestro **SparkContext**

En el caso que nos queramos conectar a un *cluster* solo cambiamos las opciones de configuración!

Spark

Corriendo programas en un cluster

Spark

Corriendo programas en un cluster

Para montar un *cluster* local lo hacemos con el comando:

```
sbin/start-master.sh
```

Spark

Corriendo programas en un cluster

Para montar un *cluster* local lo hacemos con el comando:

```
sbin/start-master.sh
```

Luego para añadir un *worker*:

```
sbin/start-worker spark://127.0.0.1:7077
```

Spark

Corriendo programas en un cluster

Spark

Corriendo programas en un cluster

Ojo! archivo de configuración del master debe asignar la dirección a *localhost*

SPARK_MASTER_HOST=127.0.0.1

Y cuando dejemos de usarlos, debemos parar los servidores con los *scripts* **stop-master** y **stop-worker** (se usan de la misma forma)

Spark

Corriendo programas en un cluster

Spark

Corriendo programas en un cluster

También podemos añadir *workers* externos, señalando la IP y el puerto

```
sbin/start-worker spark://<master-  
ip>:<master-port>
```

Siempre y cuando tengamos configurados los servidores en una red (tarea no trivial)

Spark

Corriendo programas en un cluster

Spark

Corriendo programas en un cluster

Otros *backend* para Apache Spark:

- Mesos
- Yarn
- Kubernetes
- ...

Spark

Corriendo programas en un cluster

Spark

Corriendo programas en un cluster

Una vez configurado el *cluster*, podemos conectarnos desde la consola de Spark:

```
bin/spark-shell --master spark://  
127.0.0.1:7077
```

En donde la dirección IP + Puerto es la del *cluster* (en este caso es local)

También podemos subir archivos .jar con el comando `spark-submit.sh`

Spark

Spark

Para esta clase, vamos a usar el *approach* de abrir una consola local (sin conectarse a un *cluster*) y usar el comando **:load** para cargar archivos **.scala**

Spark

Spark

Vamos a ver los siguientes ejemplos:

- Analizar un LogFile
- Spark y SQL
- Ajustar un modelo de aprendizaje automático

Técnicas para Big Data

Clase 10 - Scala y Apache Spark