Apache Spark

SGDI, Gorka Suárez García

Introducción

1. Iniciado por Matei Zaharia en 2009.
2. Busca ser un **framework** para **procesar información** sobre **clusters**.
3. No trabaja sobre disco como Hadoop, carga **todo en memoria** para ganar mayor rapidez de procesamiento.
4. Uno de los proyectos más activos en la Apache Software Foundation.

Introducción

5. Programado con Scala y Java.
6. Utiliza los **Resilient Distributed Datasets** (RDDs) como mecanismo principal de distribución y planificación del procesamiento de información en los algoritmos.
7. Es multiplataforma y permite ser usado desde Scala, Java y Python.

Instalación en Windows

Requisitos:

- Scala (2.11.4)
- Spark (1.2.0)
- Maven (3.2.3)
- SBT (0.13.7)

(Nota: Las versiones indicadas son las usadas en los ejemplos de la presentación.)

Instalación en Windows

Compilación:

```
CD C:\Spark
SET SCALA_HOME=C:\Scala
SET PATH=%PATH%;%SCALA_HOME%\bin;C:\Maven\bin
SET MAVEN_OPTS=-Xmx2G -XX:MaxPermSize=512M -XX:
ReservedCodeCacheSize=512M
mvn -DskipTests clean package
```

Instalación en Windows

Comprobación:

- Iniciar el shell de Spark:
  C:\Spark\bin\spark-shell
- Introducir la siguiente sentencia:
  sc.parallelize(1 to 1000).count()
- Y observar que el resultado es 1000.

El lenguaje Scala

1. Iniciado por Martin Odersky en 2001.
2. Es un lenguaje funcional, con orientación a objetos y tipado estático (los tipos se pueden inferir a la hora de compilar).
3. Funciona sobre la JVM, por lo que es compatible con Java y su entorno.

Nociones básicas de Scala

```
// Comentarios como en Java
val constante = 123
var variable = "123"
var lista = List(1, 2, 3)
var tupla = (1, 2, 3)
var nada = ()
var (x, y, z) = tupla
```

Nociones básicas de Scala

```
// Diversión con listas:
var a = 1 :: List(2, 3, 4, 5)
var b = 1 to 5
var c = 1 until 6
var d = 1 to 10 by 2
// Acceso al contenido:
var x = a(2)
```

Nociones básicas de Scala

Estructuras de control existentes:

- if-else (el else opcional, si no se indica se devuelve ())
- while
- do-while
- for (ejerce de for-each)
- break (para salir de un bucle)

Nociones básicas de Scala

```
// Funciones normales (los parámetros
// siempre necesitan un tipo):
def sum(a:Int, b:Int) = { a + b }

// Expresiones lambda:
var f = (a, b) => (a + b) / 2
```

Nociones básicas de Scala

- La tabulación es importa para empezar una nueva sentencia o continuar la anterior.
- Las {} sirven para hacer bloques de sentencias, se devuelve la última como valor final en las funciones.
- Con ; se puede tener varias sentencias en una sola línea.

Nociones básicas de Scala

```
// Encaje de patrones:
(xs zip ys) map { case (x,y) => x * y }

// El comodín:
(1 to 6).map(x => x * 2)
(1 to 6).map(_ * 2)
```

Lanzar una tarea (MyTest.scala)

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
object MyTest {
  def main(args: Array[String]) {
    val sc = new SparkContext(new SparkConf().setAppName("Test"))
    val result = sc.textFile("hamlet.txt").flatMap(_.split(" "))
                   .filter(_!="").map((_, 1)).reduceByKey(_+_)
                   .sortBy({case (k,v) => v}, false)
    println("\n\n\n" + result.collect().take(50).mkString("\n"))
  }
}
```

Lanzar una tarea (MyTest.sbt)

```
name := "My Test"

version := "1.0"

scalaVersion := "2.11.4"

libraryDependencies += "org.apache.spark" %%
"spark-core" % "1.2.0"
```

Lanzar una tarea (MyTest.bat)

```
SET PATH=%PATH%;C:\Spark\bin;C:\SBT\bin
CALL sbt package
COPY target\scala-2.11\*.jar /B MyTest.jar /B
CALL spark-submit --class "MyTest" --master
local[2] MyTest.jar
```

¿Qué hace la tarea?

```
// Crear el contexto para poder trabajar.
val sc = new SparkContext(new SparkConf().setAppName("Test"))
val result = sc.textFile("hamlet.txt") // Cargar fichero.
               .flatMap(_.split(" "))  // Partir las líneas.
               .filter(_!="")          // Quitar cadenas vacías.
               .map((_, 1))            // Generar las tuplas.
               .reduceByKey(_+_)       // Reducir sumando.
               .sortBy({case (k,v) => v}, false) // Ordenarlo.
// Mostrar los 50 primeros resultados.
println("\n\n\n" + result.collect().take(50).mkString("\n"))
```

Lanzar un fichero script

- Iniciar el shell de Spark: C:\Spark\bin\spark-shell
- Invocar al fichero con el script: :load ruta_fichero
- Mostraremos a continuación algunos ejemplos ya conocidos de tareas de map-reduce vistas en clase.

Contador de palabras

```
def wordCount(inputPath:String, outputPath:String) = {
  var r = sc.textFile(inputPath).flatMap(_.split(" "))
            .map((_, 1)).reduceByKey(_+_)
            .sortBy({case (k,v) => v}, false)
  var output = scala.tools.nsc.io.File(outputPath)
  output.writeAll(r.collect().mkString("\n"))
}
wordCount("input/4-Hamlet.txt", "output/WordCount.txt")
```

Weblog

```
def weblogExample(inputPath:String, outputPath:String) = {
  var r = sc.textFile(inputPath).map(_.split(" "))
            .filter(x => { x(x.length - 2) == "302" })
            .map(x => (x(1).drop(1).takeWhile(_!=':'), 1))
            .reduceByKey(_+_).sortBy{case (k,v) => k}
  var output = scala.tools.nsc.io.File(outputPath)
  output.writeAll(r.collect().mkString("\n"))
}
weblogExample("input/1-*", "output/Weblog.txt")
```

Temperatura

```
def temperatureExample(inputPath:String, outputPath:String) = {
  var r = sc.textFile(inputPath).map(_.split(","))
            .map(x=>(x(1)+" "+x(2),x(8).toFloat-x(12).toFloat))
            .groupByKey().map{case (k,v) => (k, (v.max, v.min))}
            .sortBy{case (k,v) => k}
  var output = scala.tools.nsc.io.File(outputPath)
  output.writeAll(r.collect().mkString("\n"))
}
temperatureExample("input/2-*", "output/Temperature.txt")
```

"Felicidad"

```
def happinessExample(inputPath:String, outputPath:String) = {
  var r = sc.textFile(inputPath).map(_.split("\t"))
            .filter(x => { x(2).toFloat < 2.0 && x(4) != "--" })
            .map(x => ("Palabras muy tristes", x(0)))
            .groupByKey().map{case (k,v)=>(k,v.mkString(", "))}
            .map{case (k,v) => k + ": " + v}
  var output = scala.tools.nsc.io.File(outputPath)
  output.writeAll(r.collect().mkString("\n"))
}
happinessExample("input/3-*", "output/Happiness.txt")
```

Índice de palabras (1ª parte)

```
def cleanPath(s:String) = {
  var r = s
  while(r.exists(_=='/')) r = r.dropWhile(_!='/').drop(1)
  r
}

def lineSplit(s:String) = {
    s.map(x => if (x.isLetterOrDigit) x else ' ')
     .split(" ").filter(!_.isEmpty)
}
```

Índice de palabras (2ª parte)

```
def indexExample(inputPath:String, outputPath:String) = {
  var r = sc.wholeTextFiles(inputPath)
            .map(x => (cleanPath(x._1),x._2))
            .flatMap{case (f,c)=>lineSplit(c).map(x=>((x,f),1))}
            .reduceByKey(_+_)
            .map{case ((w,f),v) => (w,(f,v))}
            .groupByKey()
            .filter(x => x._2 != null)
            .filter(x => !x._2.filter(_._2 > 20).isEmpty)
```

Índice de palabras (3ª parte)

```
            .map(x => (x._1,x._2.toList.sortBy(y => -y._2)))
            .sortBy({case (k,v) => v.head._2}, false)
  var output = scala.tools.nsc.io.File(outputPath)
  output.writeAll(r.collect().mkString("\n"))
}
indexExample("input/4-*", "output/Index.txt")
```

Conclusiones

1. Código más corto que en Java al usar Scala, pero en ocasiones más "feo" que en Python.
2. Una API bastante útil para las operaciones habituales que se esperaría en un map-reduce.
3. Permite mostrar datos por consola al ejecutarse las operaciones intermedias, lo cual ayuda enormemente a la hora de depurar.
4. Documentación suficiente, aunque mejorable en algunas ocasiones.

Referencias

1. spark.apache.org
2. en.wikipedia.org/wiki/Apache_Spark
3. www.scala-lang.org
4. www.artima.com/pins1ed
5. en.wikipedia.org/wiki/Scala_(programming_language)
6. www.scala-sbt.org

¿Preguntas?

La respuesta seguramente será 42.