Apache Spark

Adrián Arroyo Calle - https://adrianistan.eu

$ whoami

- Adrián Arroyo Calle
- Ingeniero Informático
- Backend Developer en Telefónica Innovación Digital
- https://adrianistan.eu
- Emacs >>> Vim
- Me encanta Prolog
- Pero uso Kotlin, Rust, JavaScript, Python, según el problema

¿Qué es Apache Spark?

- Apache Spark es un motor de computación distribuido en cluster diseñado para manejar grandes cantidades de datos.
- Es opensource y está programado en Scala, aunque se puede usar desde Java, Python y R también.
- Existe una versión comercial llamada Databricks.

¿Cuándo necesitamos Apacke Spark?

- Necesitamos realizar transformaciones a unos datos y...
  - Los datos son demasiado grandes, ocupan demasiado espacio, ...
  - Las transformaciones no necesitan ser “real time”
  - El cómputo se beneficia de la paralelización
- En general: procesamiento de datos (ETL), ciertos tipos de machine learning (ML), reportes, analíticas, etc
- De forma habitual trabajaremos con un clúster de Spark, al que se le van mandando jobs. Los jobs, si están programados usando las APIs de Spark, aprovecharán la potencia del clúster completo de forma eficiente.

¿Cómo puede funcionar Spark?

- MapReduce
  - En Spark hay dos tipos de operaciones fundamentales:
    - Map: Se aplica una función para cada dato, que devuelve otro dato
    - Reduce: Se aplica una función que N datos, los combina en un dato único
  - Los Map y Reduce se distribuyen por el clúster. Spark se encarga de enviar los datos de forma transparente entre los nodos
- RDD: Inmutabilidad y pereza
  - Los datos se cargan en un RDD. Sobre un RDD podemos hacer operaciones pero los RDD en sí son inmutables. Se nos genera otro RDD con el cambio. ¿Con el cambio? Realmente no, se almacena el cambio que hay que hacer pero son perezosos. Hasta que no necesitemos los datos, no se realizará la operación.
- HDFS
  - Aunque Spark funciona con muchas fuentes de datos, es habitual encontrarlo HDFS, un sistema de archivos (Hadoop FileSystem) optimizado para data lakes.


[🐮, 🥔, 🐔, 🌽].map(cook) ⇒ [🍔, 🍟, 🍗, 🍦]
[🍔, 🍟, 🍗, 🍦].filter(isVegetarian) ⇒ [🍟, 🍦]
[🍔, 🍟, 🍗, 🍦].reduce(eat) ⇒ 💩

¡Al lío!

https://spark.apache.org/downloads.html

Spark 3.5.0 prebuilt with Hadoop 3.0

https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020/

Races are won at the track. Championships are won at the factory.

Mercedes (2019)

spark-shell

- La primera forma que tenemos de usar Spark es mediante el Spark-Shell. Línea a línea podemos ir ejecutando código (en Scala 2)
- bin/spark-shell
  - `val df = spark.read.format(“csv”).option(“header”, true).load(“data/circuits.csv”)`
  - `df.show()`

```
scala> df.show()
+---------+-------------+--------------------+------------+---------+--------+---------+---+--------------------+
|circuitId|   circuitRef|                name|    location|  country|     lat|      lng|alt|                 url|
+---------+-------------+--------------------+------------+---------+--------+---------+---+--------------------+
|        1|  albert_park|Albert Park Grand...|   Melbourne|Australia|-37.8497|  144.968| 10|http://en.wikiped...|
|        2|       sepang|Sepang Internatio...|Kuala Lumpur| Malaysia| 2.76083|  101.738| 18|http://en.wikiped...|
|        3|      bahrain|Bahrain Internati...|      Sakhir|  Bahrain| 26.0325|  50.5106|  7|http://en.wikiped...|
|        4|    catalunya|Circuit de Barcel...|    Montmeló|    Spain|   41.57|  2.26111|109|http://en.wikiped...|
|        5|     istanbul|       Istanbul Park|    Istanbul|   Turkey| 40.9517|   29.405|130|http://en.wikiped...|
|        6|       monaco|  Circuit de Monaco| Monte-Carlo|   Monaco| 43.7347|  7.42056|  7|http://en.wikiped...|
|        7|   villeneuve|Circuit Gilles Vi...|    Montreal|   Canada|    45.5| -73.5228| 13|http://en.wikiped...|
|        8|  magny_cours|Circuit de Nevers...| Magny Cours|   France| 46.8642|  3.16361|228|http://en.wikiped...|
|        9|  silverstone| Silverstone Circuit| Silverstone|       UK| 52.0786| -1.01694|153|http://en.wikiped...|
|       10|hockenheimring|      Hockenheimring|  Hockenheim|  Germany| 49.3278|  8.56583|103|http://en.wikiped...|
|       11|  hungaroring|         Hungaroring|    Budapest|  Hungary| 47.5789|  19.2486|264|http://en.wikiped...|
|       12|     valencia|Valencia Street C...|    Valencia|    Spain| 39.4589|-0.331667|  4|http://en.wikiped...|
|       13|          spa|Circuit de Spa-Fr...|         Spa|  Belgium| 50.4372|  5.97139|401|http://en.wikiped...|
|       14|        monza|Autodromo Naziona...|       Monza|    Italy| 45.6156|  9.28111|162|http://en.wikiped...|
|       15|   marina_bay|Marina Bay Street...|  Marina Bay|Singapore|  1.2914|  103.864| 18|http://en.wikiped...|
|       16|         fuji|       Fuji Speedway|       Oyama|    Japan| 35.3717|  138.927|583|http://en.wikiped...|
|       17|     shanghai|Shanghai Internat...|    Shanghai|    China| 31.3389|   121.22|  5|http://en.wikiped...|
|       18|   interlagos|Autódromo José Ca...|   São Paulo|   Brazil|-23.7036| -46.6997|785|http://en.wikiped...|
|       19| indianapolis|Indianapolis Moto...|Indianapolis|      USA|  39.795| -86.2347|223|http://en.wikiped...|
|       20|  nurburgring|         Nürburgring|     Nürburg|  Germany| 50.3356|   6.9475|578|http://en.wikiped...|
+---------+-------------+--------------------+------------+---------+--------+---------+---+--------------------+
only showing top 20 rows
```

Spark SQL y DataFrames

- La estructura de datos fundamental de Spark es el RDD. Sin embargo, para la mayoría de tareas es de demasiado bajo nivel.
- Spark ofrece una API de DataFrames, que nos permite hacer operaciones trabajando con filas y columnas.
- La API es similar a SQL
- Pero por debajo usa todas las facilidades de Spark para que el cómputo sea distribuido en el clúster de forma óptima

- Para referirnos a una columna usaremos $”columna”, col(“columna”) o df(“columna”)

Ejemplos

- Generar dataframe nuevo seleccionando ciertas columnas
  - `df.select($“name”, $”country”).show()`
- Filtrar filas for valor de una columna
  - `df.select($”name”).where($”country” === “Spain”).show()`
- Agrupar y contar
  - `df.groupBy($”country”).count().show()`
- Ordenar
  - `df.orderBy(desc($”name”)).show()`
- Join
  - `df.join(OTRO_DF, CONDICION, TIPO)`

Desafío 1

Obtener una tabla con la temporada y el piloto que ganó esa temporada en el circuito de Mónaco.

Desafío 1 (solución)

Obtener una tabla con la temporada y el piloto que ganó esa temporada en el circuito de Mónaco.

Abrir DataFrames:

- `val circuits = spark.read.format(“csv”).option(“header”, true).load(“data/circuits.csv”)`
- `val races = spark.read.format(“csv”).option(“header”, true).load(“data/races.csv”)`
- `val drivers = spark.read.format(“csv”).option(“header”, true).load(“data/drivers.csv”)`
- `val results = spark.read.format(“csv”).option(“header”, true).load(“data/results.csv”)`

Dejamos solo las carreras que tuvieron lugar en Mónaco

- val racesInMonaco = races.join(circuits, “circuitId”, “inner”).where(circuits(“name”) === “Circuit de Monaco”)

Obtenemos los ganadores de Mónaco

- val monacoWinners = racesInMonaco.join(results, “raceId”, “inner”).where(results(“position”) === 1)

Obtenemos nombre, apellido y años de la victoria, ordenamos de forma descendente:

- monacoWinners.join(drivers, “driverId”, “inner”).select(drivers(“forename”), drivers(“surname”), races(“year”)).orderBy(desc(“year”))

```
scala> races.join(circuits, "circuitId", "inner").where(circuits("name") === "Circuit de Monaco").join(results, "raceId", "inner").where(results("position") === 1).join(drivers, "driverId", "inner").select(drivers("forename"), drivers("surname"), races("year")).orderBy(desc("year")).show()
+---------+-------------+----+
| forename|      surname|year|
+---------+-------------+----+
|      Max|   Verstappen|2023|
|   Sergio|        Pérez|2022|
|      Max|   Verstappen|2021|
|    Lewis|     Hamilton|2019|
|   Daniel|    Ricciardo|2018|
|Sebastian|       Vettel|2017|
|    Lewis|     Hamilton|2016|
|     Nico|      Rosberg|2015|
|     Nico|      Rosberg|2014|
|     Nico|      Rosberg|2013|
|     Mark|       Webber|2012|
|Sebastian|       Vettel|2011|
|     Mark|       Webber|2010|
|   Jenson|       Button|2009|
|    Lewis|     Hamilton|2008|
| Fernando|       Alonso|2007|
| Fernando|       Alonso|2006|
|     Kimi|    Räikkönen|2005|
|    Jarno|       Trulli|2004|
```

Particiones

- Para distribuir el trabajo, Spark debe saber cómo repartir los datos dentro de los nodos
- Hagamos una prueba empírica
  - `val df = spark.range(0, 100)`
  - `df.rdd.getNumPartitions`
- Por defecto, tantas particiones como núcleos nuestro ordenador
- Definamos una transformación muy compleja:

```
def superComputation(n: java.lang.Long) = { Thread.sleep(1000); n }
```

- Ejecutando
  - `df.map(superComputation).show()`
  - 5,9 segundos

- Cambiemos el particionado a uno peor
  - `val df = spark.range(0, 10).repartition(1)`
  - `df.map(superComputation).show()`
  - ¡Ahora son 13,4 segundos!

- En un DataFrame podemos reparticionar usando el método repartition. Podemos indicarle el número de particiones que queremos y/o la columna sobre la que hacer la partición
  - Idealmente se debe escoger una columna donde la mayoría de operaciones de un nodo tengan todos el mismo valor

Agregaciones

- Las agregaciones o reducciones son la principal diferencia entre una base de datos convencional.
- Las BBDD relacionales suelen estar optimizadas para OLTP. Las agregaciones (GROUP BY) se soportan pero no son eficientes.
- Spark está diseñado de modo OLAP, de modo que las agregaciones son operaciones muy eficientes
- Para agrupar en Spark usamos groupBy(). Dentro las columnas por las que agrupamos, seguido de agg y una función de agregación
  - `df.groupBy(“name”).agg(sum(“pedidos”) as “pedidos_total”)`

Desafío 2

¿Cuántas vueltas rápidas tiene cada piloto?

Desafío 2 (solución)

¿Cuántas vueltas rápidas tiene cada piloto?

```
val fastestLapDriversSum = results                                    : sql.DataFrame
  .where(results("rank") === 1)                                       : Dataset[Row]
  .groupBy(results("driverId"))                                       : RelationalGroupedDataset
  .agg(sum( columnName = "rank") as "fastest_laps")                   : sql.DataFrame
  .join(drivers,  usingColumn = "driverId",  joinType = "inner")      : sql.DataFrame
  .select( col = "forename",  cols = "surname", "fastest_laps")       : sql.DataFrame
  .orderBy(desc( columnName = "fastest_laps"))                        : Dataset[Row]

fastestLapDriversSum.write.format( source = "csv").option("header", true).save( path = "results/drivers-fastest-laps-sum")
```

Despliegue

- Spark Shell es muy bonito pero salvo para hacer pruebas, es poco práctico
- Podemos generar un JAR para enviar a Spark.
- Mínimo dos archivos
  - build.sbt
  - src/main/scala/Formula1.scala

build.sbt

```
name := “Formula 1 Spark”

version := “1.0”

scalaVersion := “2.12.18”

libraryDependencies += “org.apache.spark” %% “spark-sql” % “3.5.0”
```

src/main/scala/Formula1.scala

```
import org.apache.spark.sql.SparkSession


object Formula1 {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder.appName(“Formula 1”).getOrCreate()
    // Aquí ya tenemos acceso a la variable spark para leer DF
    spark.stop()
  }
}
```

Compilar y subir

Compilamos con:

```
$ sbt package
```

Y mandamos al cluster de Spark con spark-submit

```
$ bin/spark-submit --class “Formula1” --master local[4]
FICHERO.jar
```

map / withColumn / when / reduce

- Con map podemos cambiar por completo cada fila dentro de un DF. Podemos ejecutar cualquier código Scala
  - `circuits.map(row => (row.getString(0), row.getString(2).toUpperCase)).toDF("circuitId", "name")`
- Con withColumn podemos agregar /sustituir una columna
  - `circuits.withColumn("name_uppercase", upper(col("name")))`
  - Las funciones tienen que ser funciones de Spark o UDFs (más adelante)
- Una función de Spark muy interesante con withColumn es when
  - `circuits.withColumn("in_spain", when(col("country") === "Spain", true).otherwise(false))`
- Podemos usar reduce para convertir de N valores a 1 valor. Para ello se especifica como se combinan dos filas
- También existe la función Spark SQL reduce, que se puede aplicar en agregaciones.

UDF y UDAFs

- Spark SQL contiene una gran cantidad de funciones. Pero a veces nosotros queremos definir las nuestras. Se pueden programar en Scala y que sean accesibles tanto desde Scala como desde SQL. UDF si operan sobre columnas y UDAF si opera sobre agregaciones.
- Aunque existe ya la función upper, hagamos como si no existiese
  - `def upperCase(str: String): String = str.toUpperCase`
  - `val upperCaseUDF = udf(upperCase _)`
  - `circuits.select(upperCaseUDF(col("name")).as("name"))`

https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/functions$.html

¿Y los RDD?

- Si nuestros datos no siguen el formato tabular podemos usar RDD, los bloques básicos de Spark
- También si queremos controlar más al detalle el paralelismo
- No obstante, actualmente se recomienda usar DataFrames si es posible ya que contiene mayores optimizaciones

Python | Scala | Java

```scala
val count = sc.parallelize(1 to NUM_SAMPLES).filter { _ =>
  val x = math.random
  val y = math.random
  x*x + y*y < 1
}.count()
println(s"Pi is roughly ${4.0 * count / NUM_SAMPLES}")
```

Spark ML

- Se trata de una librería que implementa algoritmos de Machine Learning sobre Spark.
- Hay que tener en cuenta que Spark no accede a las GPUs, por lo que ciertos tipos de ML no son adecuados en Spark (redes neuronales por ejemplo)
- Pero con otros algoritmos su uso es ideal
  - Algoritmos de clústering
  - Regresiones, SVM, Random Forest, Bayes, …
  - Algoritmos de recomendación (ALS, …)
  - Extracción y transformaciones de features

Desafío 3

De los carreras de 2022. ¿Cuál fue el número de paradas promedio?

Desafío 3 (solución)

De los carreras de 2022. ¿Cuál fue el número de paradas promedio?

- val races = spark.read.format(“csv”).option(“header”, true).load(“data/races.csv”)
- val racesIn2022 = races.filter(races(“year”) === 2022)
- val pitStops = spark.read.format(“csv”).option(“header”, true).load(“data/pit_stops.csv”)
- val pitStopsRaces2022 = racesIn2022.join(pitStops, “raceId”, “inner”)
- val pitStopsPerDriver = pitStopsRaces202.groupBy(races(“name”), $”driverId”).agg(max(“stop”) as “stops”)
- pitStopsPerDriver.groupBy(“name”).agg(avg(“stops”)).show()

```
scala> pitStopsPerDriver.groupBy("name").agg(avg("stops")).show(25)
+--------------------+------------------+
|                name|       avg(stops)|
+--------------------+------------------+
|    Miami Grand Prix|1.3157894736842106|
|  Spanish Grand Prix|              2.65|
|Brazilian Grand Prix|2.4444444444444446|
|    Dutch Grand Prix|               3.6|
|Emilia Romagna Gr...|1.3888888888888888|
|Singapore Grand Prix|               1.4|
| Canadian Grand Prix|1.7777777777777777|
|  Italian Grand Prix|1.5555555555555556|
|Azerbaijan Grand ...|1.4210526315789473|
|  British Grand Prix| 2.764705882352941|
|   Monaco Grand Prix| 2.789473684210526|
| Austrian Grand Prix|               2.1|
|Saudi Arabian Gra...|1.1176470588235294|
|Australian Grand ...|1.2222222222222223|
|Hungarian Grand Prix|               2.1|
|  Bahrain Grand Prix|               2.9|
|   French Grand Prix|1.3157894736842106|
|Abu Dhabi Grand Prix|              1.55|
|Mexico City Grand...|              1.15|
|  Belgian Grand Prix| 2.111111111111111|
|United States Gra...| 1.894736842105263|
| Japanese Grand Prix|2.3333333333333335|
+--------------------+------------------+
```

Spark SQL

- Esta API se parece mucho a SQL. Tanto que si nos gusta, podemos escribir las queries en, ¡SQL!
- Ejemplo:
  - `df.createOrReplaceTempView("circuits")`
  - `spark.sql("SELECT name, country FROM circuits").show()`

Desafío 4

¿Cuántas horas ha corrido cada piloto de F1?

Desafío 4 (solución)

¿Cuántas horas ha corrido cada piloto de F1?

```scala
lapTimes.createOrReplaceTempView( viewName = "lap_times")
drivers.createOrReplaceTempView( viewName = "drivers")

def millisecondsToHours(n: Long): Long = {
  n / 3600000
}

val millisecondsToHoursUDF = udf(millisecondsToHours _)
spark.udf.register( name = "msToHours", millisecondsToHoursUDF)

val driversTime = spark.sql(
  """
    |SELECT forename, surname, msToHours(sum(milliseconds)) AS time
    |FROM lap_times
    |JOIN drivers ON lap_times.driverId = drivers.driverId
    |GROUP BY drivers.forename, drivers.surname
    |ORDER BY time DESC
    |""".stripMargin)

driversTime.write.format( source = "csv").option("header", true).save( path = "results/drivers-time")
```

FIN