

Procesamiento de datos con Spark

© EDICIONES ROBLE, S.L.

< S.L.

Indice

Procesamiento de datos con Spark	3
I. Introducción	3
II. Objetivos	3
III. Spark	3
3.1. Historia	4
3.1.1. Antecesores	6
3.2. Componentes	7
3.2.1. Spark Core	8
3.2.2. Spark SQL	9
3.2.3. Spark Streaming	12
3.2.4. Spark MLlib	19
3.2.5. Graph X	21
3.2.6. Spark R	23
3.3. Futuros componentes	24
3.4. Spark Shell	25
3.5. Descarga	26
3.6. Compilación	28
3.7. Configuración	29
3.8. Conceptos básicos de Spark	31
3.8.1. DAG (grafo acíclico dirigido)	32
3.8.2. RDD	33
3.9. Ejemplos	35
3.9.1. Funcionamiento interno	36
3.9.2. Ejemplo Java: word count	40
3.9.3. Word count Scala	41
3.9.4. Word count: Pyspark	44
3.9.5. Ejemplo de Spark SQL	48
3.9.6. Ejemplo de Spark Streaming	51
IV. Lectura recomendada	54
V. Resumen	55
Ejercicios	56
Ejercicio	56
Solución	57
Recursos	62
Enlaces de Interés	62
Bibliografía	62
Glosario.	62

Procesamiento de datos con Spark

I. Introducción

En esta unidad, nos adentraremos en una herramienta que ha tomado mucha fuerza en los últimos años y por la que tanto las empresas como las comunidades libres están haciendo grandes apuestas. Spark da una “vuelta de tuerca” a su antecesor y nos ofrece un *framework* de desarrollo de aplicaciones paralelas que elimina alguna de las limitaciones que tenía Hadoop y ofrece un rendimiento bastante superior en tareas de *machine learning*. Además, nos ofrece componentes para trabajar en *streaming*, uso de primitivas en memoria principal, etc.

II. Objetivos



Tras el estudio de esta unidad, los alumnos habrán alcanzado los siguientes objetivos:

- Conocer el origen y la historia de Spark.
- Conocer su ecosistema.
- Aprender a instalarlo.
- Características de Spark: DAG y RDD.
- Ejemplo de análisis de *log*.
- Ejemplo de *word count*.
- Ejemplos de las distintas API.
- Integraciones de Spark.

III. Spark

Spark es un *framework* para computar datos en clúster desarrollado por AMPLab en la Universidad de California, Berkeley, y que fue donado a Apache Software Foundation. A diferencia del sistema de dos fases Map Reduce empleado por Hadoop, Spark emplea multietapas con primitivas en memoria que obtienen un rendimiento hasta cien veces superior.



Anotación: Características

- Modo de funcionamiento:
 - Spark standalone.
 - Modo pseudodistribuido.
 - Clúster: Hadoop Yarn o Apache Mesos, nativo de spark.
- Sistema de archivos distribuidos: HDFS, Hive, HBase, Cassandra, Amazon S3, Swift, Kudu, Tachyon, etc.
- Facilidad de uso: Java, Scala, Python, R.
- Diseñado para algoritmos de aprendizaje máquina o *machine learning*.
- Combina SQL, *streaming* y análisis complejos.
- Gran número de contribuidores: proyecto Big Data más activo de *software libre*.

3.1. Historia

1

En el siguiente *time line* podemos apreciar la evolución y la aparición de tecnologías como GFS, Map Reduce, Hadoop y, posteriormente, Spark.

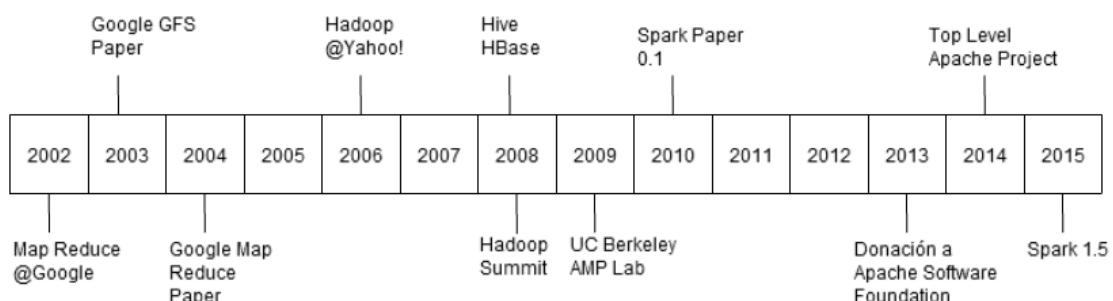


Figura 1. *Time line* Apache Spark. Fuente: Fundación Apache.

2

Map Reduce no funciona correctamente fuera del entorno *batch* general, por lo que surgieron soluciones en forma de sistemas especializados:

Procesamiento genérico en batch **Sistemas especializados: iterativos, interactivos, streaming, grafos...**



Figura 2. De Map Reduce a sistemas especializados.

Fuente: elaboración propia.

Algunos ejemplos de ello son sistemas de procesamiento en *streaming* como Storm, capacidades de consulta SQL en tiempo real como Drill e Impala, etc.

3

Una de las principales diferencias entre Spark y Hadoop es que Spark trabaja en memoria principal. Hadoop en cada fase map y reduce realizaba un acceso a disco mientras que en Spark esto no es necesario. Esto hace a Spark mas rápido en determinadas circunstancias.

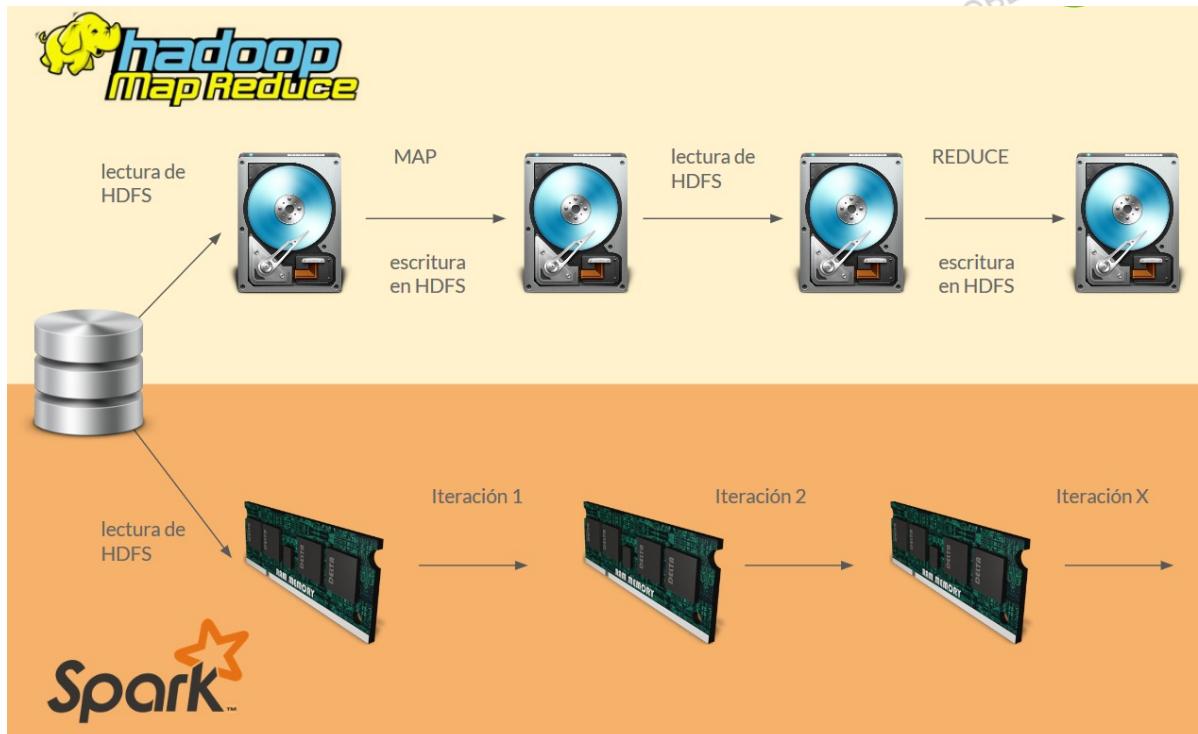


Figura 3. Map Reduce vs. Spark DAG. Fuente: Elaboración propia.

3.1.1. Antecesores

Spark se basa en dos herramientas: Scala, un lenguaje de programación, y en un *framework* de desarrollo de aplicaciones distribuidas llamado akka. A continuación podemos ver algunas de sus características:



Scala:

- Orientado a objetos y a programación funcional.
- Lenguaje de alto nivel para las JVM.
- Interopera con Java.



akka:

- *Framework open-source* de propósito general.
- Simplifica la construcción de aplicaciones, distribuidas sobre Java o Scala.
- Escrito en Scala.
- Usa un modelo de actores para abstraer todo el código relacionado con hilos.
- Proporciona una interfaz simple y útil para implementar un sistema escalable y tolerante a fallos fácilmente.



Spark:

- Se construye sobre akka.
- *Framework* para procesar datos en *batch* usando una versión generalizada del algoritmo Map Reduce.
- Un buen ejemplo de Apache Spark es el cálculo de algunas métricas de almacenamiento de datos para obtener una mejor visión de los mismos. Los datos se cargan y se procesan a petición.
- Apache Spark Streaming es capaz de realizar acciones similares y funciones en casi tiempo real para pequeños procesos de datos, del mismo modo que se podría hacer sobre datos ya almacenados.

3.2. Componentes

Spark ofrece distintos componentes para hacer frente a distintas necesidades en nuestros proyectos como consultas SQL, *machine learning* y procesamiento de grafos:

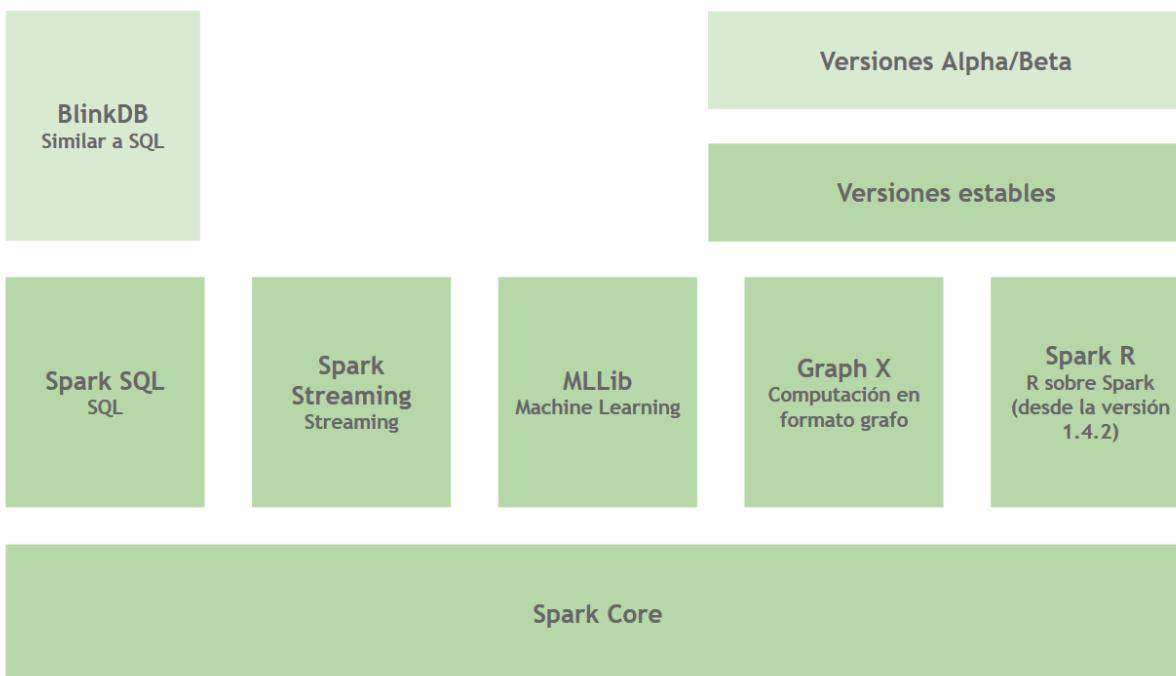


Figura 4. Arquitectura Spark. Fuente: Fundación Apache

A continuación, haremos un repaso de cada uno de los componentes principales, así como de sus características más reseñables.

3.2.1. Spark Core

	<ul style="list-style-type: none"> • Gestor distribuido de tareas, planificación y funcionalidades básicas I/O. • RDD (<i>Resilient Distributed Datasets</i>), una colección de datos particionados entre máquinas. • Pueden crearse RDD por dataset referenciando a sistemas de almacenamiento externo o aplicando transformaciones (map, filter, reduce o join) sobre RDD existentes. • Los RDD se obtienen mediante el empleo de API en Java¹, Python², Scala³, y R⁴. • Simplifica la programación ya que se las aplicaciones pueden manipular los RDD de modo similar a las colecciones de datos.
---	--



¹Wikipedia: "Java". [En línea] URL disponible en: <https://en.wikipedia.org/wiki/Java>

²Wikipedia: "Python (programming language)". [En línea] URL disponible en: [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))

³Wikipedia: "Scala (programming language)". [En línea] URL disponible en: [https://en.wikipedia.org/wiki/Scala_\(programming_language\)](https://en.wikipedia.org/wiki/Scala_(programming_language))

⁴Wikipedia: "R (programming language)". [En línea] URL disponible en: [https://en.wikipedia.org/wiki/R_\(programming_language\)](https://en.wikipedia.org/wiki/R_(programming_language))

Spark Core es un motor de ejecución general para la plataforma de Spark que permite que otros componentes se construyan sobre él.

Sus componentes y funcionalidades básicas son:

- Programación de tareas.
- Gestión de memoria.
- Recuperación frente a fallos.
- Interacción con sistemas de almacenamiento.
- Sencillo de desarrollar: permite API nativas con Java, Scala, Python, R y SQL.

Los *Resilient Distributed Datasets* (RDD) son el primer nivel de abstracción con el que Spark representa una colección de elementos distribuidos sobre múltiples nodos y que pueden ser manipulados en paralelo.

El programador puede definir el número de particiones asociadas a un RDD.

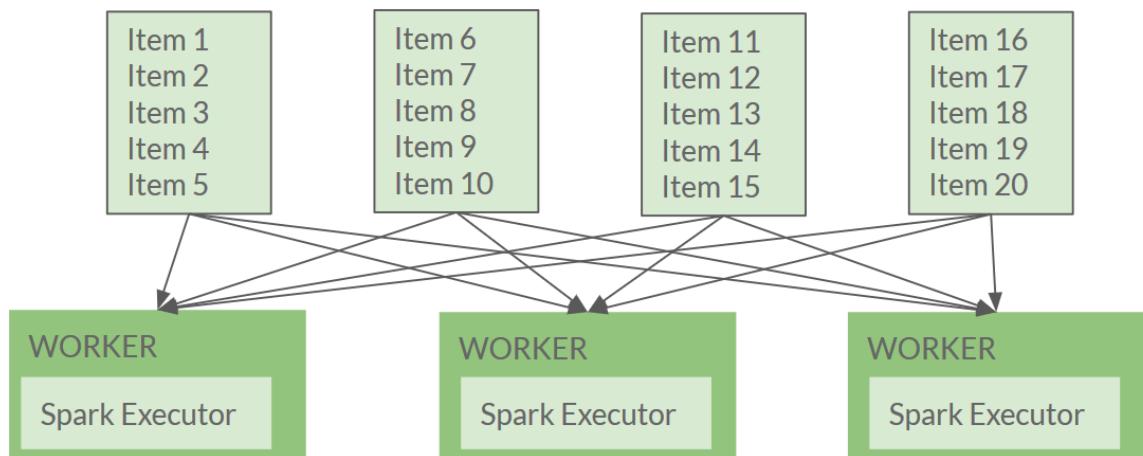


Figura 5. Procesamiento RDD. Fuente: Fundación Apache.

3.2.2. Spark SQL



- Introduce una nueva abstracción mediante los llamados DataFrames, que proporciona soporte a datos estructurados y semiestructurados.
- Spark SQL proporciona un lenguaje de dominio específico para manipular los DataFrames en Scala, Java, o Python.
- También proporciona un lenguaje de soporte en SQL, con una interfaz de línea de comandos y un servidor ODBC⁵/JDBC⁶ server.
- Hasta la versión 1.3 los DataFrames se llamaban SchemaRDDs.



⁵Wikipedia: “Open Database Connectivity”. [En línea] URL disponible en: https://en.wikipedia.org/wiki/Open_Database_Connectivity

⁶Wikipedia: “Java Database Connectivity”. [En línea] URL disponible en: https://en.wikipedia.org/wiki/Java_Database_Connectivity

Spark SQL es una interfaz que permite trabajar con datos estructurados y semiestructurados (datos que tienen un esquema, es decir, un conjunto de campos definidos).

Características:

Integrado

Permite mezclar queries SQL con programas Spark.

```
context = HiveContext(sc)

results = context.sql( "SELECT * FROM people")

names = results.map(lambda p: p.name)
```

Acceso uniforme a los datos

Permite conectar con cualquier origen de datos del mismo modo incluyendo conectores con AVRO, Hive, Parquet, ORC, JSON y JDBC.

```
context.jsonFile("s3n://...").registerTempTable("json")

results = context.sql("""SELECT * FROM people JOIN json ...""")
```

Compatibilidad con Hive

Se pueden ejecutar queries de Hive directamente sin realizar modificaciones.

```
import org.apache.spark.sql.hive.HiveContext  
  
val sc = new SparkContext(...)  
  
val hiveCtx = new HiveContext(sc)
```

Estándar de conectividad

Con JDBC y ODBC.

Rendimiento y escalabilidad: incluye un optimizador basado en coste, almacenamiento en modo columna y un generador de código para optimizar las consultas.

Carga de datos y ejecución de queries

La carga de datos y ejecución de queries devuelve esquemas RDD (Data Frames)

- Un SchemaRDD es similar a las tablas en los sistemas de datos tradicionales.
- Un SchemaRDD es un RDD compuesto por filas de objetos con información adicional de cada tipo en cada columna.

```
> printSchema(people)  
root  
|-- age: long (nullable = true)  
|-- name: string (nullable = true)  
>
```



```
// Incluimos import
import org.apache.spark.sql.hive.HiveContext

// Creamos el contexto
val sc = new SparkContext(...)

val hiveCtx = new HiveContext(sc)

// Recuperamos los datos de un fichero de entrada
val input = hiveCtx.jsonFile ("/ejemplosSpark/tweets.json")

// Registrarmos los datos dentro del schema RDD
input.registerTempTable("tweets")

// Seleccionamos los tweets basandonos en el nùmero de retweets

val topTweets = hiveCtx.sql("SELECT text, retweetCount FROM tweets ORDER
BY retweetCount LIMIT 10")
```

3.2.3. Spark Streaming



- Emplea la capacidad de planificación de tareas de Spark Core para realizar tareas analíticas en *streaming*.
- Puede ingerir datos de miniprocesos *batch* y realizar transformaciones mediante RDD sobre los datos de dichos minibatches.
- Este diseño permite que el código diseñado para un proceso *batch* analítico pueda utilizarse en un proceso analítico en *streaming*, sobre una única máquina.

Spark Streaming es una interfaz que permite construir aplicaciones de *streaming* tolerantes a fallos de un modo sencillo.

Características:

Fácil de usar

Permite construir aplicaciones basándose en operadores incluidos en su API, proporcionando soporte para Java, Scala y Python.

```
TwitterUtils.createStream(...)

.filter(_.getText.contains("Spark"))

.countByWindow(Seconds(5))
```

Resistente frente a fallos

Recupera trabajos perdidos y su estado.

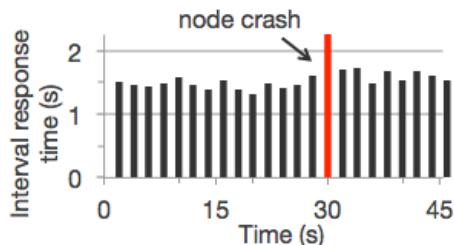


Figura 6. Fallo de un nodo.

Fuente: Fundación Apache.

Integración con Spark

Combina integración en modo *batch* o consultas interactivas.

```
// Buscar palabras con mayor frecuencia que los datos históricos.
```

```
stream.join(historicCounts).filter {

    case (word, (curCount, oldCount)) =>

        curCount > oldCount
}
```

Orígenes de datos

Los datos pueden ser ingeridos desde distintos orígenes de datos y sus resultados pueden mostrarse en diferentes formatos:



Figura 7. Fuentes de datos. *Fuente:* Fundación Apache.

También se pueden emplear como orígenes de datos los algoritmos de aprendizaje máquina y los algoritmos de procesamiento de grafos.

Spark Streaming extiende la API del *core* para permitir alto rendimiento y procesamiento tolerante frente a fallos de los datos de entrada.



Figura 8. Procesamiento *batch*. *Fuente:* Fundación Apache.

Gestión batch

La computación en *streaming* se trata como un conjunto de computaciones *batch*.

En cada intervalo de tiempo:

- Se recogen los datos.
- Al final del intervalo se crea un RDD.
- Los datos se procesan en tareas Spark.

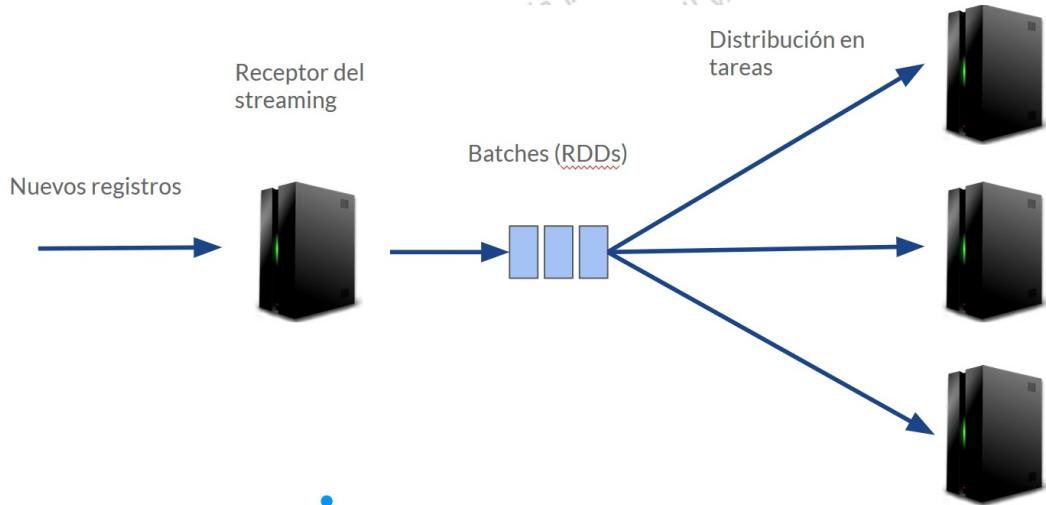


Figura 9. Distribución de pequeñas tareas *batch*.
Fuente: Elaboración propia.

Los registros son procesados en *batches* con tareas pequeñas donde cada *batch* es un RDD.

Cuellos de botella

En los sistemas tradicionales de procesamiento en *streaming* se generan cuellos de botella, dado que la planificación de trabajos es estática y no tiene en cuenta la carga y el uso de recursos de cada nodo.

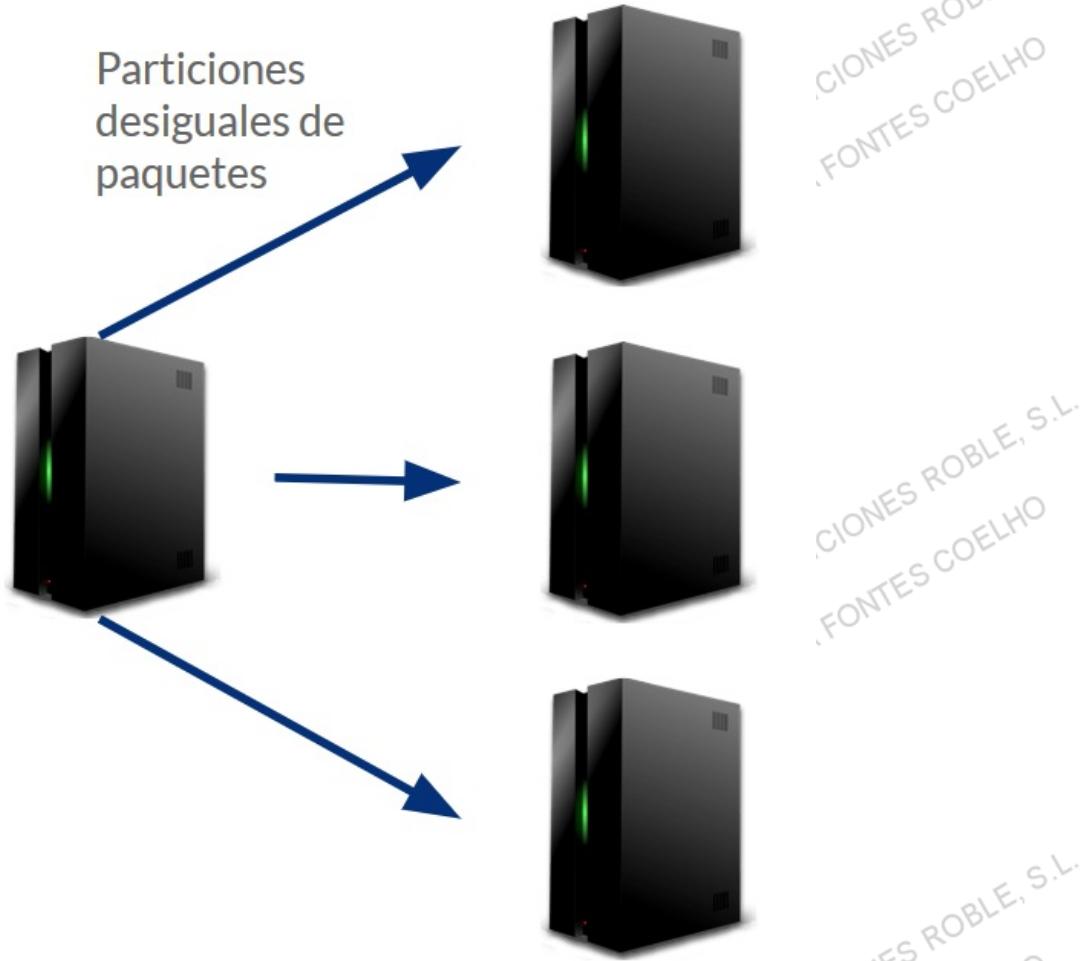


Figura 10. Planificaciones estáticas. *Fuente:* Elaboración propia.

Modificación de tamaños de tareas

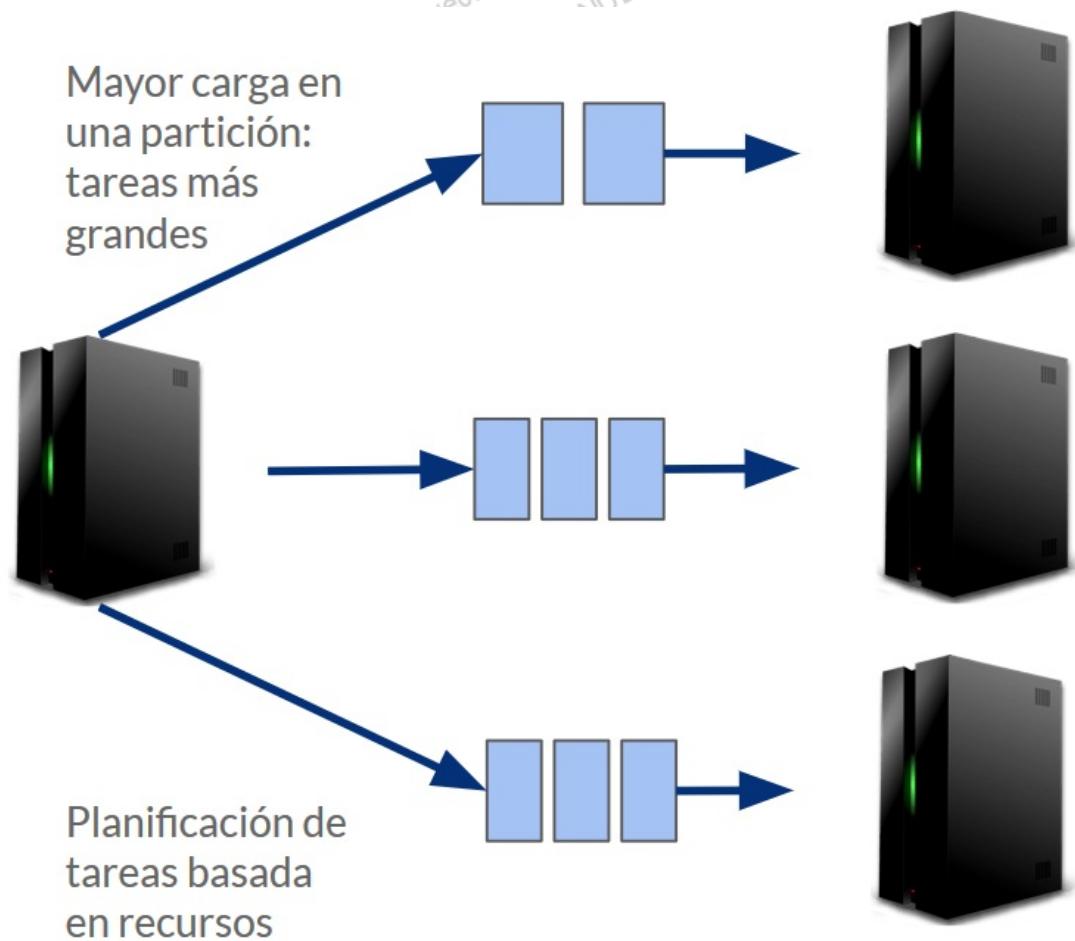


Figura 11. Planificación dinámica. *Fuente:* Elaboración propia.

Spark utiliza planificaciones dinámicas de tareas de distintos tamaños que tienen en cuenta el uso de recursos de los nodos que realizan los trabajos.

Recuperación ante fallos

En los sistemas tradicionales de procesamiento en *streaming*, la caída de un nodo exige que su carga la asuma otro nodo del clúster. Esto genera sobrecargas y cuellos de botella.

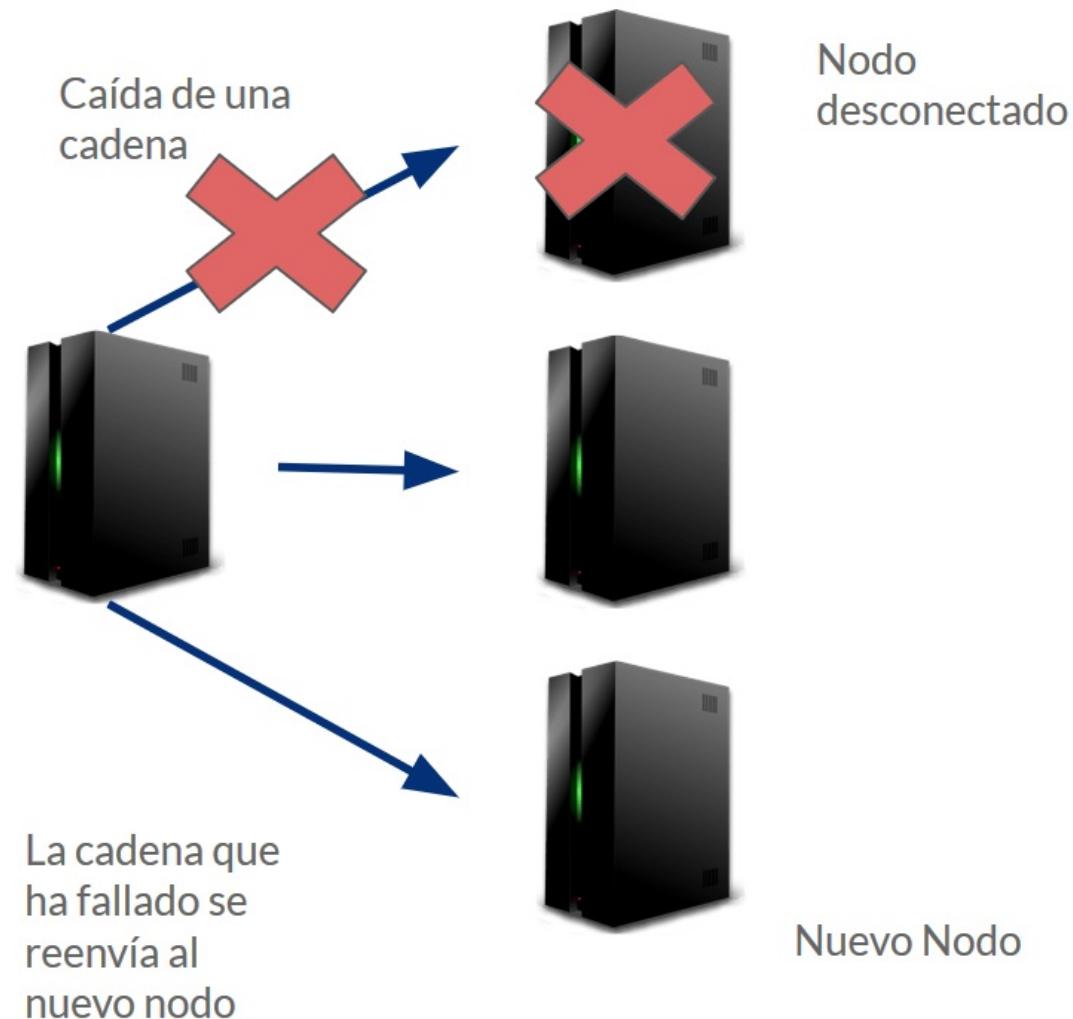


Figura 12. Fallo de un nodo en sistemas tradicionales. *Fuente:* Elaboración propia.

No obstante, Spark Streaming reparte el trabajo que tenía asignado el nodo que falló entre los nodos disponibles atendiendo a políticas basadas en el uso de recursos.

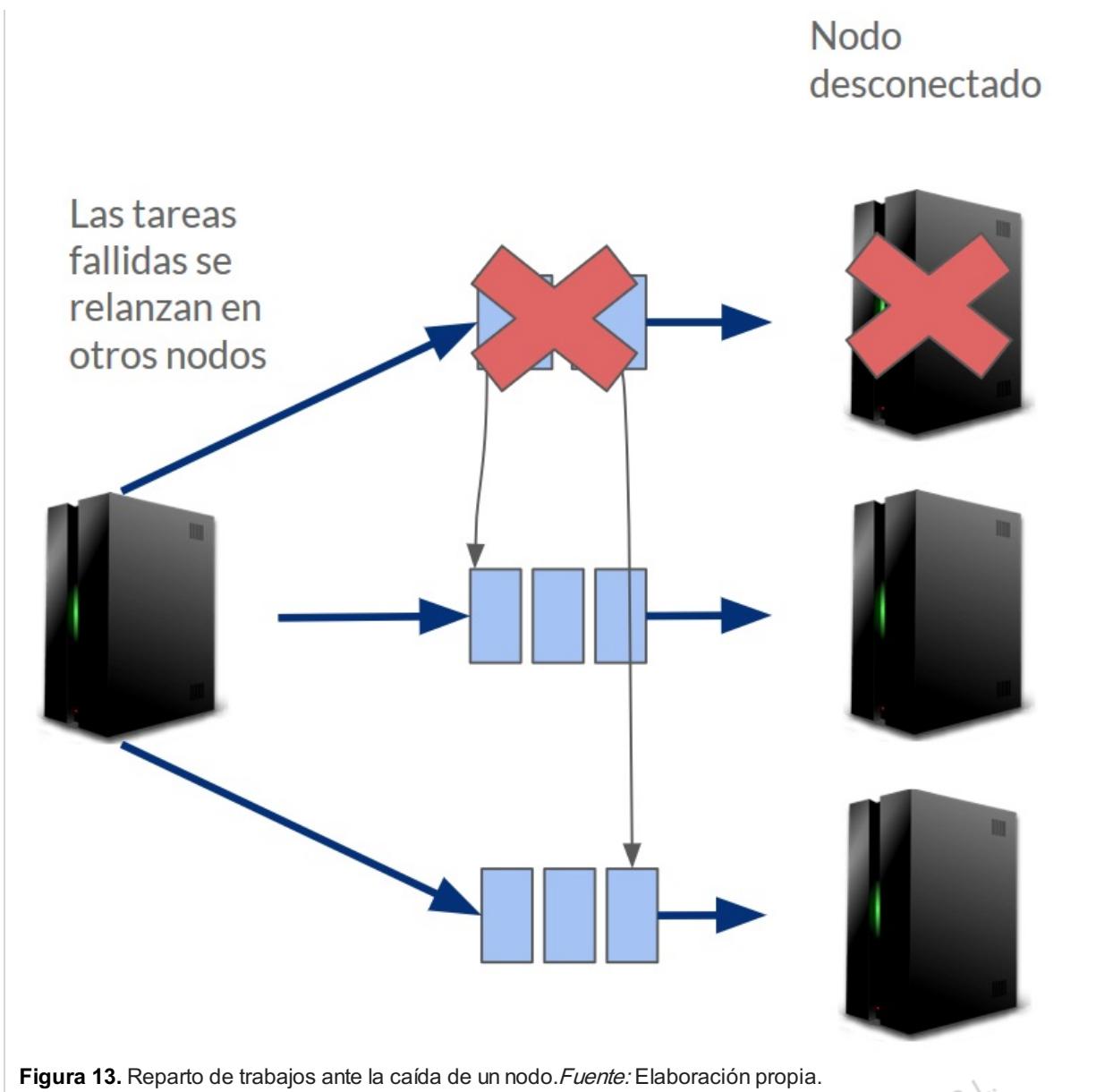


Figura 13. Reparto de trabajos ante la caída de un nodo. *Fuente:* Elaboración propia.

3.2.4. Spark MLlib

 MLlib	<ul style="list-style-type: none"> • <i>Framework</i> distribuido de aprendizaje automático construido sobre Spark Core. • Gracias a su arquitectura Spark permite un rendimiento nueve veces superior a las implementaciones basadas en disco que empleaba. <ul style="list-style-type: none"> ◦ Apache Mahout (de acuerdo con los estudios realizados por los desarrolladores de MLlib frente a las implementaciones de Alternating Least Squares (ALS) y antes de que Mahout se integrara en la interfaz de Spark). ◦ Vowpal Wabbit, librería de <i>machine learning</i> de Yahoo que gestiona Microsoft Research. • Muchos algoritmos estadísticos y de aprendizaje máquina han sido implementados e incluidos en MLlib.
-----------	--

Spark MLlib es una librería escalable de aprendizaje de lenguaje máquina.

Sus características son:

Fácil de usar

- Permite construir aplicaciones basándose en operadores incluidos en su API.
- Proporciona soporte para Java, Scala, Python y Spark R.
- Permite emplear como orígenes de datos cualquier origen de Hadoop (HDFS, HBase o sistema de archivos local), haciendo sencilla su integración con los flujos de trabajo de Hadoop.

Rendimiento

Posee algoritmos de alta calidad cien veces más rápidos que Map Reduce.

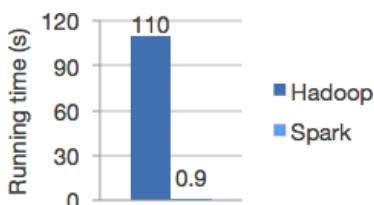


Figura 14. Rendimiento de Spark con respecto a Map Reduce.

Fuente: <https://bbvaopen4u.com>

Fácil de desplegar

Puede correr sobre clústers de Hadoop 2.X existentes y sobre sus datos, sin realizar ningún tipo de preinstalación. También instalarse en un standalone en EC2 o en Mesos.

Librería de algoritmos

Posee una amplia librería de algoritmos y utilidades

- Regresión logística y SVM (*linear support vector machine*).
- Algoritmos de clasificación y árboles de regresión.
- *Random Forest* y *gradient-boosted trees*.
- Recomendación mediante ALS (*alternating least squares*).
- Clustering mediante k-means, mezclas gaussianas (GMM).
- Modelado tópico mediante LDA (*latent dirichlet allocation*).
- Descomposición de valores singulares (SVD) y descomposición QR.
- Análisis de componentes principales (PCA).
- Regresiones lineales con L1, L2 y regulación elástica en red.
- Regresión isotónica.
- Multinomial/binomial bayesano ingenuo.
- Minería conjunta de elementos a través de crecimiento-FP y reglas de asociación.
- Minería de modelos secuenciales a través de PrefixSpan.
- Resúmenes estadísticos y pruebas de hipótesis.
- Transformaciones de funciones.

- Modelos de evaluación y tuning de hiperparámetros.

```
points = spark.textFile("hdfs://...").map(parsePoint)

model = KMeans.train(points, k=10)

// Evaluamos el modelo

val test_points = spark.textFile("hdfs://...")

test_points.map( t=> model.predict (t)).collect().foreach(println)
```

3.2.5. Graph X



- Iniciado como proyecto de investigación de la UC Berkeley's AMPLab and Databricks como Spark.
- *Framework* de procesamiento de grafos construido sobre Spark.
- Proporciona una API para expresar computación gráfica que permite modelar abstracciones Pregel.
- También proporciona un modelo de ejecución optimizado para esta abstracción.

Abstracción de Pregel

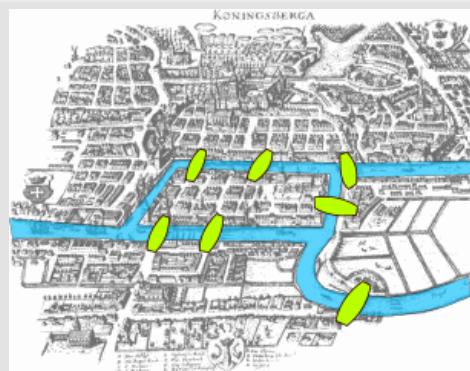


Figura 15. Puentes de Königsberg. Fuente: Wikipedia.

Sienta las bases de la teoría de grafos ampliamente utilizada en el mundo de las matemáticas y la computación.

La abstracción de Pregel es una abstracción del mapa de la ciudad de Königsberg que sirvió para enunciar el conocido problema matemático.



Anotación: Problema de los puentes de Königsberg

- Resuelto por Leonhard Euler en 1736.
- Dio origen a la teoría de grafos.
- Königsberg, ciudad de Prusia Oriental y luego de Alemania que desde 1945 se convertiría en la ciudad rusa de Kaliningrado.
- Esta ciudad está atravesada por el río Pregel que divide el terreno en cuatro regiones distintas, que entonces estaban unidas mediante siete puentes.
- El problema fue formulado en el siglo XVIII y consistía en encontrar un recorrido para cruzar a pie toda la ciudad, pasando solo una vez por cada uno de los puentes y regresando al mismo punto de inicio.

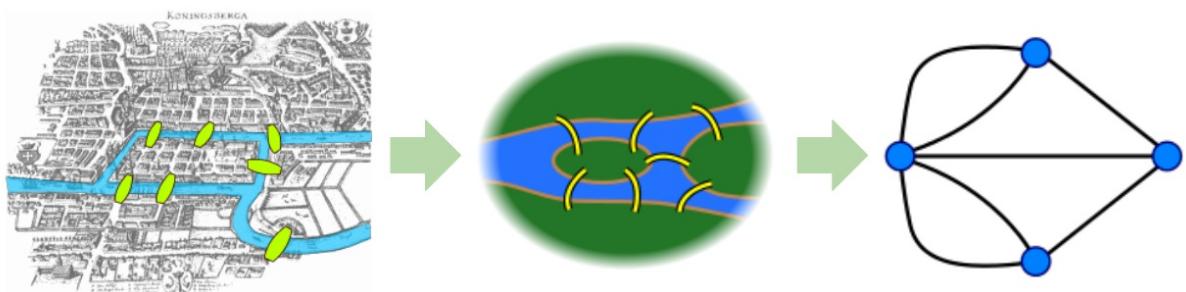


Figura 16. Teoría de grafos. *Fuente:* Wikipedia

Spark GraphX es una API para grafos y su procesamiento en paralelo.

Sus características son:

Flexibilidad

- Permite trabajar con colecciones y con grafos.
- GraphX unifica los ETL, el análisis exploratorio y la computación interactiva gráfica en un único sistema.
- Se pueden visualizar los datos como grafos o como colecciones, transformar los grafos y unirlos con RDD de forma eficiente, y escribir tus propios algoritmos iterativos usando la API de Pregel.

Velocidad

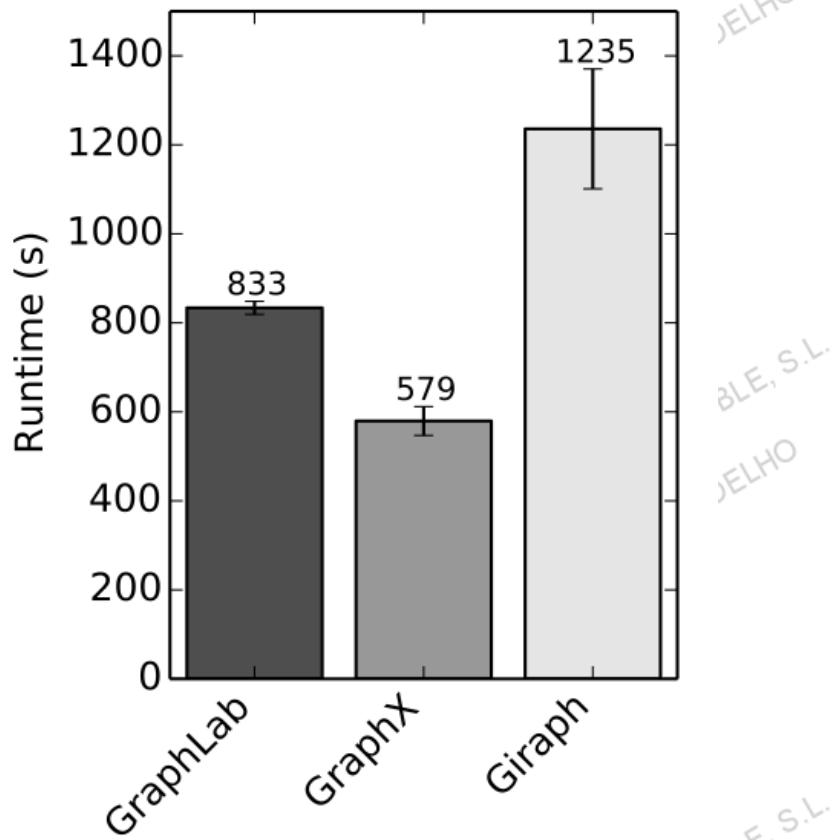
- Rendimiento similar a los sistemas de procesamiento de grafos especializados más rápidos.
- Mantiene la flexibilidad de Spark, la tolerancia frente a fallos y la facilidad de uso.

Algoritmos

La API incluye una gran variedad de algoritmos basados en grafos como:

- PageRank.
- Connected components.
- Label propagation.
- SVD++.
- Strongly connected components.
- Triangle count.

Figura 17. Comparativa con sistemas especializados.



Fuente: Fundación Apache.

3.2.6. Spark R

	<ul style="list-style-type: none">• Es un paquete de R que proporciona un <i>frontend</i> ligero para usar Apache Spark desde R.• En Spark 1.5.2, SparkR proporciona una implementación del <i>data frame</i>.• Permite soportar operaciones sobre grandes <i>datasets</i> como:<ul style="list-style-type: none">• Selección.• Filtrado.• Agregación.• Soporta el aprendizaje máquina distribuido empleando MLlib.
---	--

3.3. Futuros componentes

	
<p><u>BlinkDB:</u></p> <p>Motor de queries que permite ejecutar de forma masiva y en paralelo queries SQL sobre grandes volúmenes de datos.</p> <p>Permite a los usuarios gestionar los tiempos de respuesta y crear queries interactivas:</p> <ul style="list-style-type: none">• Queries sobre muestras de datos.• Representando los resultados con barras de errores significativos.	

Figura 18. Ejemplos de queries.

<pre>SELECT avg(sessionTime) FROM Table WHERE city='San Francisco' WITHIN 2 SECONDS</pre>	<pre>SELECT avg(sessionTime) FROM Table WHERE city='San Francisco' ERROR 0.1 CONFIDENCE 95.0%</pre>
Queries with Time Bounds	Queries with Error Bounds
<i>Fuente: http://blinkdb.org/</i>	

3.4. Spark Shell

La instalación de Spark también incluye varias *shells* para interactuar con los datos:

Python: ./bin/pyspark

```
lines = sc.textFile("README.md") # Creamos un RDD llamado lines
lines.count() # devuelve el número de elementos incluido en el RDD
```

Scala: ./bin/spark-shell

```
val lines = sc.textFile("README.md") # Creamos un RDD llamado lines
lines.count() # devuelve el número de elementos incluido en el RDD
```

SparkR: ./bin/sparkR

Desde la versión 1.4⁷ se ha incorporado el paquete de R que proporciona un *frontend* ligero para usar Apache Spark desde R.

SparkR proporciona una implementación del *dataframe* que permite soportar operaciones como selección, filtrado y agregación, pero sobre grandes *datasets*. SparkR también soporta el aprendizaje máquina distribuido empleando MLlib.

```
sc <- sparkR.init()
rdd <- parallelize(sc, 1:10)
count(rdd) # 10
```

⁷Databricks: "Announcing SparkR: R on Apache Spark" [En línea] URL disponible en: <https://databricks.com/blog/2015/06/09/announcing-sparkr-r-on-spark.html>

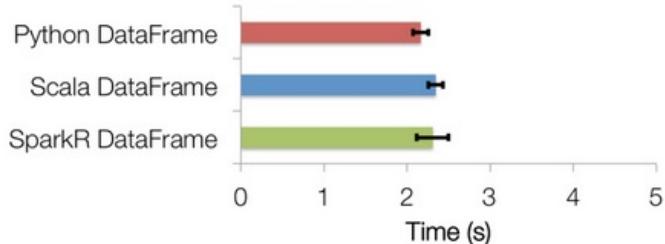


Figura 19. Comparativa de tiempos de ejecución de distintas *shells* disponibles. Fuente: Fundación Apache.

3.5. Descarga

Para trabajar con Spark, lo primero que debemos hacer es descargar el paquete desde su web.



Página web de Apache Spark, <http://spark.apache.org/downloads.html>



APACHE
Spark™ *Lightning-fast cluster computing*

Apache Software Foundation ▾

Latest News

- Spark 2.2.1 released (Dec 01, 2017)
- Spark 2.1.2 released (Oct 09, 2017)
- Spark Summit Europe (October 24-26th, 2017, Dublin, Ireland) agenda posted (Aug 28, 2017)
- Spark 2.2.0 released (Jul 11, 2017)

[Archive](#)

Download Apache Spark™

1. Choose a Spark release: **2.2.1 (Dec 01 2017)** ▾

2. Choose a package type: **Pre-built for Apache Hadoop 2.7 and later** ▾

3. Download Spark: **spark-2.2.1-bin-hadoop2.7.tgz**

4. Verify this release using the [2.2.1 signatures and checksums](#) and [project release KEYS](#).

Note: Starting version 2.0, Spark is built with Scala 2.11 by default. Scala 2.10 users should download the Spark source package and build with Scala 2.10 support.

Link with Spark

Spark artifacts are [hosted in Maven Central](#). You can add a Maven dependency with the following coordinates:

```
groupId: org.apache.spark
artifactId: spark-core_2.11
version: 2.2.0
```

Download Spark

Installing with PyPi

PySpark is now available in pypi. To install just run `pip install pyspark`.

We suggest the following mirror site for your download:
<http://apache.rediris.es/spark/spark-2.2.1/spark-2.2.1-bin-hadoop2.7.tgz>
 Other mirror sites are suggested below. Please use the backup mirrors only to download PGP and MD5 signatures to verify your downloads or if no other mirrors are working.

[Google Custom](#)
[The Apache Way](#)
[Contribute](#)
[ASF Sponsors](#)

HTTP

<http://apache.rediris.es/spark/spark-2.2.1/spark-2.2.1-bin-hadoop2.7.tgz>
<http://apache.uvigo.es/spark/spark-2.2.1/spark-2.2.1-bin-hadoop2.7.tgz>
<http://ftp.cixug.es/apache/spark/spark-2.2.1/spark-2.2.1-bin-hadoop2.7.tgz>

1

wget <https://archive.apache.org/dist/spark/spark-2.2.1/spark-2.2.1-bin-hadoop2.7.tgz>

```
bigdata@bigdata:~$ wget http://apache.rediris.es/spark/spark-2.2.1/spark-2.2.1-bin-hadoop2.7.tgz
--2017-12-08 20:32:16-- http://apache.rediris.es/spark/spark-2.2.1/spark-2.2.1-bin-hadoop2.7.tgz
Resolviendo apache.rediris.es (apache.rediris.es)... 130.206.13.2
Conectando con apache.rediris.es (apache.rediris.es)[130.206.13.2]:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 200934340 (192M) [application/x-gzip]
Guardando como: "spark-2.2.1-bin-hadoop2.7.tgz"

spark-2.2.1-bin-hadoop2. 26%[=====>] 50,36M 15,4MB/s eta 10s
```

27/62

2

```
tar -xvf spark-2.2.1-bin-hadoop2.7.tgz
```

```
bigdata@bigdata:~$ tar -xvf spark-2.2.1-bin-hadoop2.7.tgz
```

3

```
mv spark-2.2.1-bin-hadoop2.7 /home/bigdata/spark
```

```
bigdata@bigdata:~$ mv spark-2.2.1-bin-hadoop2.7 /home/bigdata/spark
```

4

```
sudo nano ~/.bashrc
```

```
bigdata@bigdata:~$ sudo nano ~/.bashrc
```

5

```
# SPARK VARIABLES START
```

```
export SPARK_HOME=/home/bigdata/spark
```

```
export PATH=$PATH:$SPARK_HOME/bin
```

```
# SPARK VARIABLES END
```

```
# SPARK VARIABLES START
export SPARK_HOME=/home/bigdata/spark
export PATH=$PATH:$SPARK_HOME/bin
# SPARK VARIABLES END
```

6

```
source ~/.bashrc
```

```
bigdata@bigdata:~$ source ~/.bashrc
```

3.6. Compilación

Ahora, se detallan los pasos para descargar el código de Spark, así como su posterior compilación, aunque para el seguimiento de los ejemplos esto no es necesario.

Download Apache Spark™

1. Choose a Spark release:
2. Choose a package type:
3. Download Spark: [spark-2.2.1-bin-hadoop2.7.tgz](#)
4. Verify this release using the [2.2.1 signatures and checksums](#) and [project release KEYS](#).

Note: Starting version 2.0, Spark is built with Scala 2.11 by default. Scala 2.10 users should download the Spark source package and build [with Scala 2.10 support](#).

```
bigdata@bigdata:~/spark$ ./sbt/sbt compile
NOTE: The sbt/sbt script has been relocated to build/sbt.
      Please update references to point to the new location.

      Invoking 'build/sbt compile' now ...

Using /usr/lib/jvm/java-7-openjdk-amd64 as default JAVA_HOME.
Note, this will be overridden by -java-home if it is set.
Attempting to fetch sbt
Launching sbt from build/sbt-launch-0.13.7.jar
Getting org.scala-sbt sbt 0.13.7 ...
```

3.7. Configuración

A continuación, se detallan los pasos para configurar Spark:

1

```
cd spark
```

```
cd conf
```

```
ls
```

```
bigdata@bigdata:~/spark/conf$ ls
docker.properties.template    metrics.properties.template    spark-env.sh.template
fairscheduler.xml.template   slaves.template
log4j.properties.template    spark-defaults.conf.template
bigdata@bigdata:~/spark/conf$
```

2

```
cp spark-env.sh.template spark-env.sh
```

```
bigdata@bigdata:~/spark/conf$ cp spark-env.sh.template spark-env.sh
bigdata@bigdata:~/spark/conf$
```

3

```
sudo nano spark-env.sh
```

```
bigdata@bigdata:~/spark/conf$ sudo nano spark-env.sh
```

4

```
SPARK_DIST_CLASSPATH=$(/home/bigdata/hadoop/bin/hadoop classpath)
```

```
export
```

```
SPARK_DIST_CLASSPATH="$SPARK_DIST_CLASSPATH:/home/bigdata/hadoop/share/hadoop/
```

```
#!/usr/bin/env bash
SPARK_DIST_CLASSPATH=$( /home/bigdata/hadoop/bin/hadoop classpath)
export SPARK_DIST_CLASSPATH="$SPARK_DIST_CLASSPATH:/home/bigdata/hadoop/share/hadoop/tools/lib/*"
```

5

```
sudo cp log4j.properties.template log4j.properties
```

```
bigdata@bigdata:~/spark/conf$ sudo cp log4j.properties.template log4j.properties
```

```
sudo nano log4j.properties
```

```
bigdata@bigdata:~/spark/conf$ sudo nano log4j.properties
```

6

Modificamos el archivo log4j.properties, cambiando:

log4j.rootCategory=INFO, console

por:

log4j.rootCategory=ERROR, console

```
# Set everything to be logged to the console
log4j.rootCategory=ERROR, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n

# Settings to quiet third party logs that are too verbose
log4j.logger.org.spark-project.jetty=WARN
log4j.logger.org.spark-project.jetty.util.component.AbstractLifeCycle=ERROR
log4j.logger.org.apache.spark.repl.SparkIMain$exprTyper=INFO
log4j.logger.org.apache.spark.repl.SparkILoop$SparkILoopInterpreter=INFO
log4j.logger.org.apache.parquet=ERROR
log4j.logger.parquet=ERROR

# SPARK-9183: Settings to avoid annoying messages when looking up nonexistent UDFs in SparkSQL with Hive support
log4j.logger.org.apache.hadoop.hive.metastore.RetryingHMSHandler=FATAL
log4j.logger.org.apache.hadoop.hive.ql.exec.FunctionRegistry=ERROR
```

spark-shell

```
bigdata@bigdata:~/spark/conf$ spark-shell
```

scala> exit

scala>:q

Welcome to



version 2.2.1

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_151)
 Type in expressions to have them evaluated.
 Type :help for more information.

scala> ■

pyspark

NOTA: Para que funcione Pyspark, debemos cambiar la version de Python para que detecte el entorno de nuestra version de Python en Spark, que está configurado para la 2.7. Por lo que debemos descargarnos la version anterior de python

sudo apt install python

```
bigdata@bigdata:~/spark/conf$ pyspark
```

>>>exit()

3.8. Conceptos básicos de Spark

1

Apache Spark combina:

- Un sistema de computación distribuida a través de clústers de ordenadores.
- Una manera sencilla y elegante de escribir programas.

2

Map Reduce permite:

- Trabajar con grandes conjuntos de datos.
- Un modelo relativamente simple para escribir programas.
 - Ejecución paralela.
 - Cientos y miles de máquinas.
- Una relación lineal de escalabilidad.
 - Si los datos crecen, es posible añadir más máquinas y tardar lo mismo.

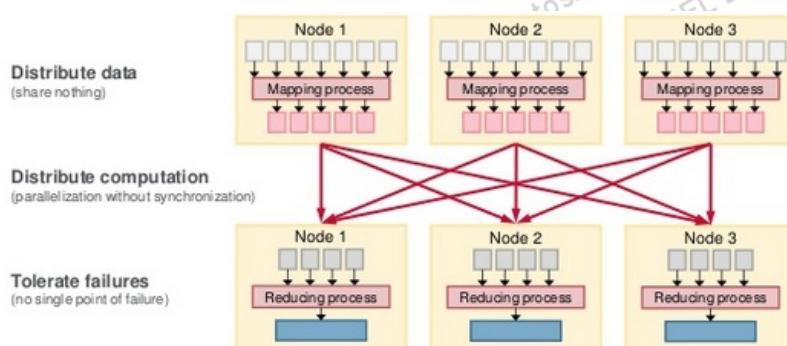


Figura 20. Modelo de ejecución paralela. *Fuente:* Fundación Apache.

Spark:

- Mantiene la escalabilidad lineal y la tolerancia a fallos.
- Amplía sus bondades gracias a DAG y RDD.

3.8.1. DAG (grafo acíclico dirigido)

Una de las limitaciones principales de Hadoop era el propio paradigma Map Reduce. Este paradigma nos limita a procesamientos que nos encajen en esas dos fases tan definidas, map y reduce. Spark en este caso, rompe ese corsé de dos fases que nos imponía Hadoop y utiliza grafos acíclicos dirigidos (DAG) con n etapas para realizar el procesamiento sobre el clúster.

- Grafo dirigido que no tiene ciclos.
- Para cada nodo del grafo no hay un camino directo que comience y finalice en dicho nodo.
- Un vértice se conecta a otro, pero nunca a sí mismo.

Spark soporta el flujo de datos acíclico.

- Cada tarea de Spark crea un DAG de etapas de trabajo para que se ejecuten en un determinado clúster.
- Map Reduce crea un DAG con dos estados predefinidos (map y reduce), los grafos DAG creados por Spark pueden tener cualquier número de etapas.

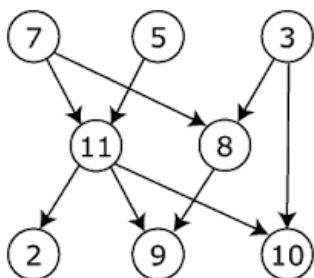


Figura 21. DAG.

Fuente: Wikipedia.

Spark con DAG es **más rápido** que MapReduce:

- **No tiene que escribir en disco** los resultados obtenidos en las etapas intermedias del grafo. Map Reduce, sin embargo, debe escribir en disco los resultados entre las etapas map y reduce.
- Gracias a una completa API, es posible **programar complejos hilos de ejecución** paralelos en unas pocas líneas de código.

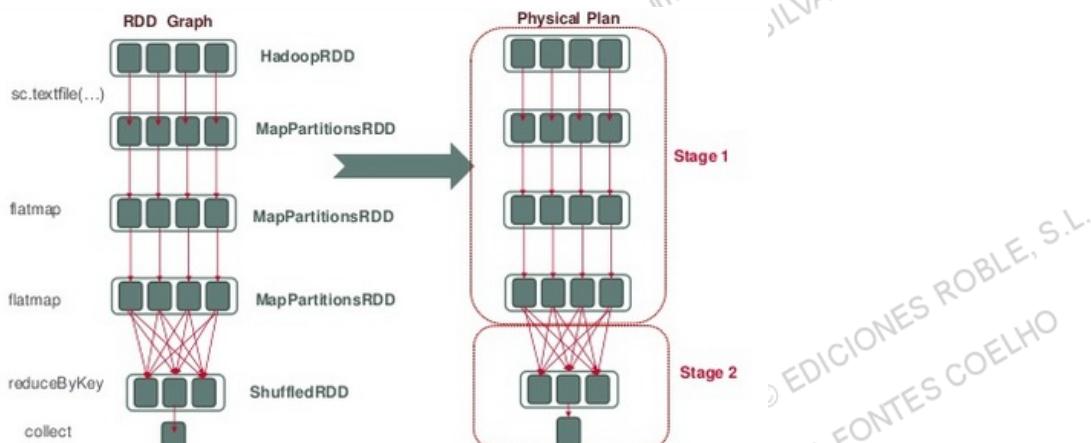


Figura 22. Planificador DAG. Fuente: Fundación Apache

3.8.2. RDD

Nivel de abstracción que Spark utiliza para manejar las infomación. Una colección distribuida inmutable de objetos. Cada conjunto de datos en un RDD se divide en particiones lógicas, que se pueden procesar en diferentes nodos del clúster. Los RDD pueden contener cualquier tipo de objetos de Python, Java o Scala, incluidas las clases definidas por el usuario.

Apache Spark mejora con respecto a los demás sistemas en cuanto a la computación en memoria.

Un RDD es una colección distribuida de elementos.

Los RDD permiten a los programadores realizar operaciones sobre grandes cantidades de datos en clústers de una manera rápida y tolerante a fallos.

1

Surgen debido a:

- Herramientas existentes inefficientes a la hora de ejecutar algoritmos iterativos y procesos de minería de datos.
- Datos en memoria mejora el rendimiento considerablemente.

2

Una vez que los datos se han leído como objetos RDD en Spark, pueden realizarse diversas operaciones mediante sus API. Los dos tipos de operaciones que se pueden realizar son:

- **Transformaciones (p. ej., map, filter).** Tras aplicar una transformación, obtenemos un nuevo y modificado RDD basado en el original.
- **Acciones (p. ej., count, reduce).** Una acción consiste simplemente en aplicar una operación sobre un RDD y obtener un valor como resultado, que dependerá del tipo de operación.

Características de los RDD

- **Inmutable.** Los RDD son estructuras de datos que apuntan a un origen de datos pero sus datos son de solo lectura y no pueden ser actualizados.
- **Transformaciones.** Nos permiten crear nuevos RDD partiendo de un origen.
- **Lazy evaluation.** Las transformaciones son perezosas, es decir, no se computan inmediatamente. Se ejecutarán cuando una acción corre sobre ellas.
- **Recuperación frente a fallos.**
- **Caché en memoria o disco.** Dado que las tareas de Spark pueden necesitar realizar diversas acciones o transformaciones sobre un conjunto de datos en particular, es altamente recomendable y beneficioso en cuanto a eficiencia el almacenar RDD en memoria para un rápido acceso a los mismos. Mediante la función cache() se almacenan los datos en memoria para que no sea necesario acceder a ellos en disco. El almacenamiento de los datos en memoria caché hace que los algoritmos de *machine learning* ejecutados que realizan varias iteraciones sobre el conjunto de datos de entrenamiento sea más eficiente. Además, se pueden almacenar versiones transformadas de dichos datos.
- **Control de persistencia y particionado.**
- **Transformaciones.** Crean nuevos RDD a partir de uno existente (*lazy execution*).
- **Acciones.** Disparan la ejecución de etapas de un RDD.

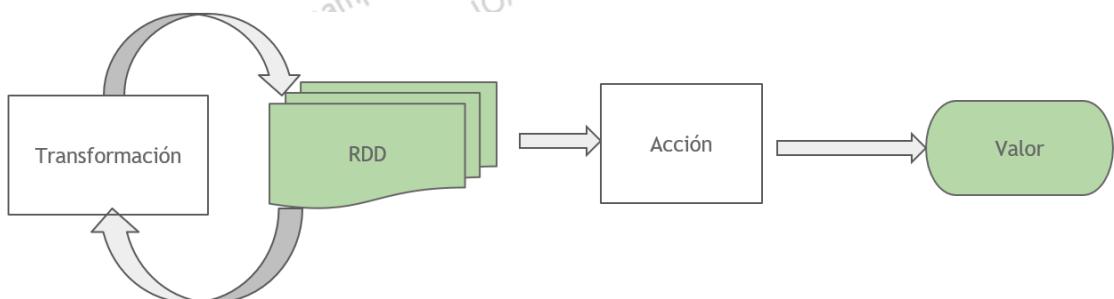


Figura 23. Transformaciones y acciones en Spark.

Fuente: Fundación Apache.

```
// Transformaciones de RDDs

val errores = lineas.filter(_.startsWith("ERROR"))

val mensajes = errores.map(_.split("\t")).map(r => r(1))

mensajes.cache()

// Acciones

mensajes.filter(_.contains("mysql")).count()
```

Formas de crear un RDD en Spark

Paralelizando una colección		Referenciando un conjunto de datos de una fuente externa	
Scala	scala> val data = Array (1, 2, 3, 4, 5) scala> val distData = sc.parallelize (data)	Scala	scala> val distFile = sc.textFile("data.txt") distFile: RDD[String] = MappedRDD@1d4cee08
Python	>>> data = [1, 2, 3, 4, 5] >>> distData = sc.parallelize(data)	Python	>>> distFile = sc.textFile("data.txt")
Java	List<Integer> data = Arrays.asList(1, 2, 3, 4, 5); JavaRDD<Integer> distData = sc.parallelize(data)	Java	JavaRDD<String> distFile = sc.textFile("data.txt");

Tabla 1. Creación de RDD en distintos lenguajes.
Fuente: Fundación Apache.

Tabla 2. Referencia de datos externos. *Fuente:* Fundación Apache.

3.9. Ejemplos

En este ejemplo sencillo veremos cómo se aplican en Spark las distintas transformaciones hasta llegar a la acción que nos dará el resultado. Se trata de un ejemplo en el que cargaremos información desde un *log* contenido en HDFS en un RDD, le aplicaremos varias transformaciones hasta llegar al RDD deseado y, mediante una acción, devolveremos un resultado.



Ejemplo: análisis de logs

```
// Cargaremos los mensajes de error desde un fichero de log en la memoria  
// y después buscaremos patrones entre sus datos  
  
// Creamos nuestro RDD  
  
val lines= sc.textFile ("hdfs://.....")  
  
// Obtenemos nuevos RDD transformados  
  
val errors= lines.filter(_.startsWith("ERROR"))  
  
val messages= errors.map(_.split("\t")).map( r => r(1))  
  
messages.cache()  
  
// Acción 1: identificar errores mysql  
  
messages.filter(_.contains("mysql")).count()  
  
// Acción 2: identificar errores php  
  
messages.filter(_.contains("php")).count()
```

3.9.1. Funcionamiento interno

A continuación, se detalla el funcionamiento interno de Spark.

1

Lo primero que haremos será crear un *context* que nos permita trabajar con HDFS recuperando información desde este.

```
val lines= sc.textFile ("hdfs://.....")
```

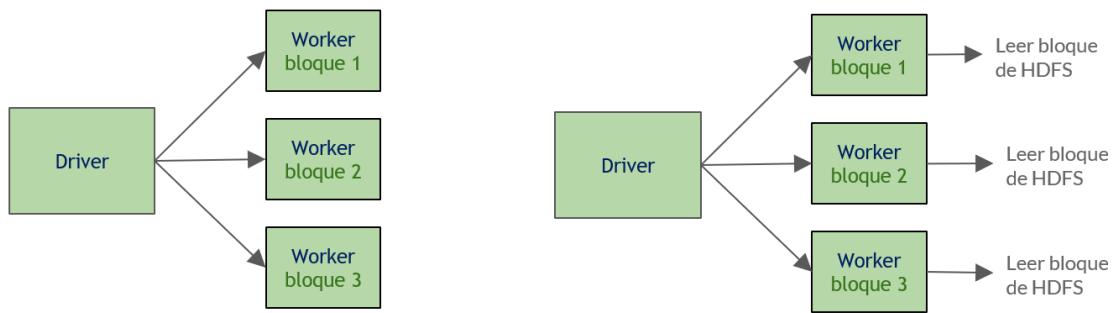


Figura 24. Paso 1: carga de datos desde HDFS.
Fuente: Fundación Apache.

2

Después, realizaremos una serie de transformaciones generando nuevos RDD con la información del original filtrada y parseada.

```
val errors= lines.filter(_.startsWith("ERROR"))
val messages= errors.map(_.split("\t")).map( r => r(1))
```



Figura 25. Paso 2: transformaciones. Fuente: Fundación Apache.

3

Ahora, cachearemos el RDD "messages":

```
messages.cache()
```

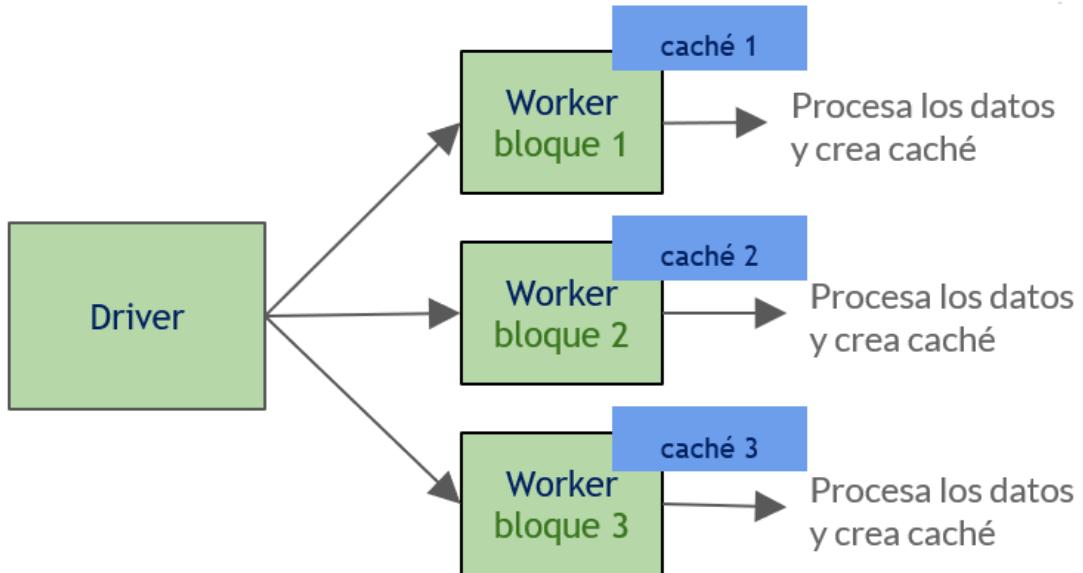


Figura 26. Paso 3: cacheo. *Fuente:* Fundación Apache.

4

Y ejecutaremos un *count* sobre el RDD que obtuvimos en el paso anterior generando un valor final:

```
messages.filter(_.contains("mysql")).count()
```

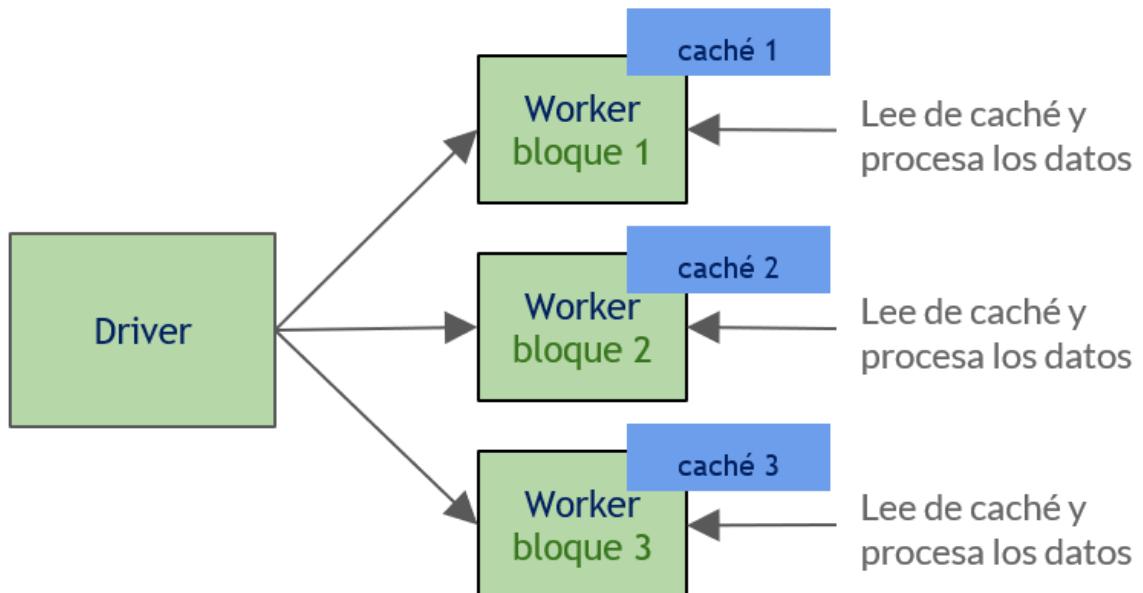


Figura 27. Paso 4: lectura de caché. *Fuente:* Fundación Apache.

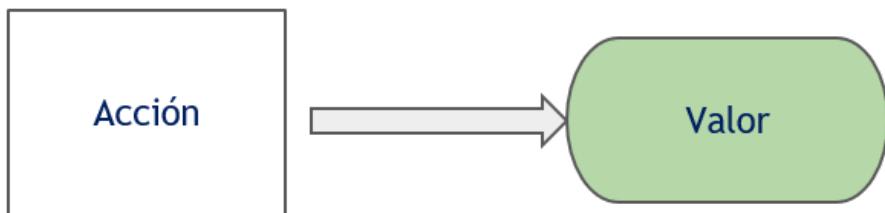


Figura 28. Paso 5: acción y devolución del valor. *Fuente:* Fundación Apache.

Ejecutaremos un *count* sobre el RDD que obtuvimos en el paso anterior generando un valor final:

```
messages.filter(_.contains("php")).count()
```

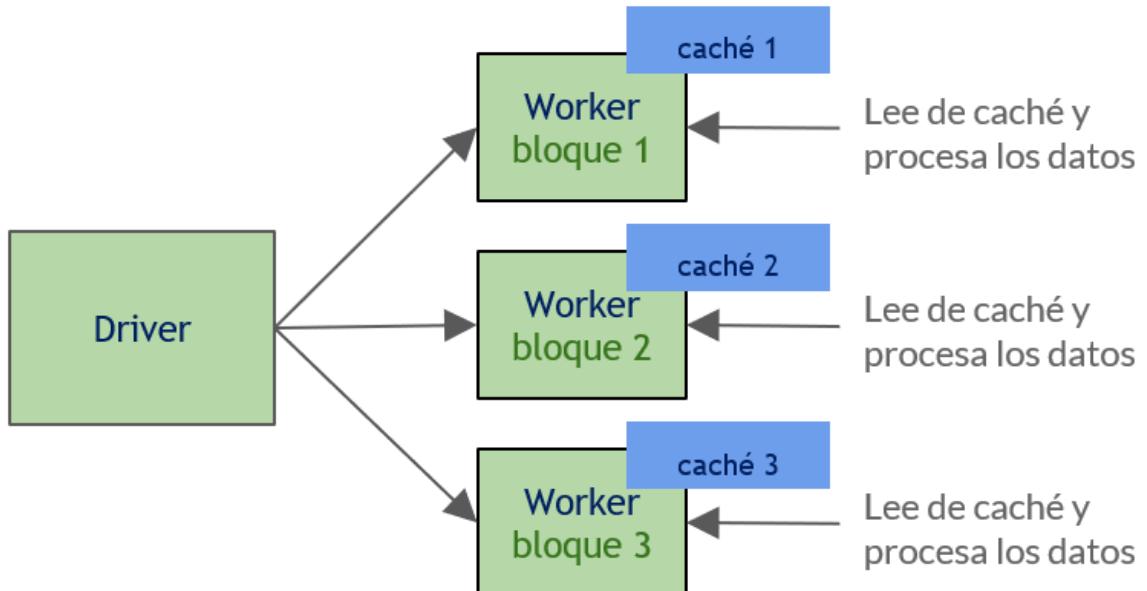


Figura 29. Paso 4: lectura de caché. *Fuente:* Fundación Apache.

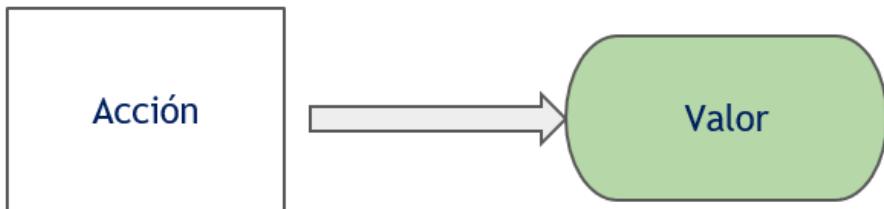


Figura 30. Paso 7: acción y devolución del valor. *Fuente:* Fundación Apache.

3.9.2. Ejemplo Java: *word count*

Aquí tenemos el ejemplo del *word count* en Java que utilizábamos en Hadoop.

```

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context
            ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class IntSumReducer
        extends Reducer<Text,IntWritable,Text,IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values,
            Context context
            ) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }

        public static void main(String[] args) throws Exception {
            Configuration conf = new Configuration();
            Job job = Job.getInstance(conf, "word count");
            job.setJarByClass(WordCount.class);
            job.setMapperClass(TokenizerMapper.class);
            job.setCombinerClass(IntSumReducer.class);
            job.setReducerClass(IntSumReducer.class);
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(IntWritable.class);
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));
            System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
    }
}

```

A continuación, veremos cómo podemos realizar este ejemplo en Spark.

3.9.3. Word count Scala

Ahora, utilizaremos el archivo `readme.md` que trae la instalación de Spark para ver algunos ejemplos de tratamientos de cadenas con Spark. En este caso con Scala.

1

```

cd $SPARK_HOME

cat README.md

bigdata@bigdata:~/spark/conf$ cd $SPARK_HOME
bigdata@bigdata:~/spark$ cat README.md
# Apache Spark

Spark is a fast and general cluster computing system for Big Data. It provides
high-level APIs in Scala, Java, Python, and R, and an optimized engine that
supports general computation graphs for data analysis. It also supports a
rich set of higher-level tools including Spark SQL for SQL and DataFrames,
MLlib for machine learning, GraphX for graph processing,
and Spark Streaming for stream processing.

<http://spark.apache.org/>

## Online Documentation

```

2

jps

```
$ start-dfs.sh
```

```
bigdata@bigdata:~/spark$ jps
32418 Jps
bigdata@bigdata:~/spark$ start-dfs.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/bigdata/hadoop/logs/hadoop-bigdata-namenode-bigdata.out
localhost: starting datanode, logging to /home/bigdata/hadoop/logs/hadoop-bigdata-datanode-bigdata.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/bigdata/hadoop/logs/hadoop-bigdata-secondarynamenode-bigdata.out
bigdata@bigdata:~/spark$
```

3

```
hdfs dfs -mkdir /spark  
hdfs dfs -put README.md /spark
```

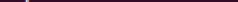
```
bigdata@bigdata:~/spark$ hdfs dfs -mkdir /spark  
bigdata@bigdata:~/spark$ hdfs dfs -put README.md /spark
```

4

spark-shell

```
bigdata@bigdata:~/spark$ spark-shell
```

Welcome to

 version 2.2.1

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.

Type :help for more information.

| scala>

Spark, como otros *frameworks*, nos ofrece una interfaz web donde podemos consultar métricas sobre los trabajos realizados en el clúster. En este caso la interfaz web de Spark está disponible en la máquina virtual a través del enlace:

1

localhost:4040/jobs

```
15/11/18 23:28:20 INFO repl.SparkILoop: Created sql context..  
SQL context available as sqlContext.
```

```
scala> █
```



Jobs

Stages

Storage

Environment

Executors

Spark shell application UI

↳

Spark Jobs (?)

User: bigdata
Total Uptime: 59 s
Scheduling Mode: FIFO

▶ Event Timeline

```
15/11/18 23:28:20 INFO repl.SparkILoop: Created sql context..  
SQL context available as sqlContext.
```

```
scala> █
```

2

Creación de un *context* y carga del archivo README.md:

```
scala> val textFile = sc.textFile("/spark/README.md")
```

```
scala> val textFile = sc.textFile("/spark/README.md")  
textFile: org.apache.spark.rdd.RDD[String] = /spark/README.md MapPartitionsRDD[1] at textFile at <console>:24
```

3

Contamos las líneas del archivo:

```
scala> textFile.count()
```

```
scala> textFile.count()  
res0: Long = 103
```

```
scala> █
```

4

Recuperamos la primera línea:

```
scala> textFile.first()
```

```
scala> textFile.first()
res1: String = # Apache Spark
```

5

Líneas que contienen la palabra Spark:

```
scala> val linesWithSpark = textFile.filter(line => line.contains("Spark"))
```

```
scala> val linesWithSpark = textFile.filter(line => line.contains("Spark"))
linesWithSpark: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:26
```

```
scala> textFile.filter(line => line.contains("Spark")).count()
```

```
scala> textFile.filter(line => line.contains("Spark")).count()
res2: Long = 20
```

```
scala> exit
```

```
scala> :q
```

3.9.4. Word count: Pyspark

A continuación, practicaremos el mismo ejemplo, en este caso con Python.

1

```
$ pyspark
```

bigdata@bigdata:~/spark\$ pyspark

Welcome to

 version 2.2.1

Using Python version 2.7.13 (default, Nov 23 2017 15:37:09)
SparkSession available as 'spark'.

>>>

2

```
>>> textFile = sc.textFile("/spark/README.md")
```

```
>>> textFile = sc.textFile("/spark/README.md")
```

3

```
>>> textFile.count() # Contar el número de elementos en el RDD
```

```
>>> textFile.count()  
103
```

4

```
>>> textFile.first() # Obtener el primer elemento del RDD
```

```
>>> textFile.first()  
u'# Apache Spark'
```

5

```
>>> linesWithSpark = textFile.filter(lambda line: "Spark" in line) # Crear un nuevo RDD como resultado de un filtrado
```

```
>>> linesWithSpark = textFile.filter(lambda line: "Spark" in line)
>>> 
```

6

```
>>> textFile.filter(lambda line: "Spark" in line).count() # Contar el nº de líneas que contienen la palabra Spark
```

```
>>> textFile.filter(lambda line: "Spark" in line).count()  
20
```

7

```
>>> # Obtener la línea con más palabras  
>>> textFile.map(lambda line: len(line.split())).reduce(lambda a, b: a if (a > b) else b)
```

```
>>> textFile.map(lambda line: len(line.split())).reduce(lambda a, b: a if (a > b) else b)  
22
```

8

```
>>> wordCounts = textFile.flatMap(lambda line: line.split()).map(lambda word: (word,  
1)).reduceByKey(lambda a, b: a+b)
```

```
>>> wordCounts.collect()
```

Procesamiento de datos con Spark

```
>>> wordCounts = textFile.flatMap(lambda line: line.split()).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)
>>> wordCounts.collect()
[(u'project.', 1), (u'help', 1), (u'when', 1), (u'Hadoop', 3), (u'MLLib', 1), (u'"local"', 1), (u'./dev/run-tests', 1), (u'including', 4), (u'graph', 1), (u'computation', 1), (u'file', 1), (u'high-level', 1), (u'find', 1), (u'web', 1), (u'shell', 2), (u'cluster', 2), (u'also', 4), (u'using:', 1), (u'Big', 1), (u'guidance', 2), (u'run:', 1), (u'scala', 1), (u'Running', 1), (u'should', 2), (u'environment', 1), (u'to', 17), (u'only', 1), (u'module', 1), (u'given.', 1), (u'rich', 1), (u'directory', 1), (u'Apache', 1), (u'Interactive', 2), (u'sc.parallelize(range(1000)).count()', 1), (u'Building', 1), (u'do', 2), (u'guide', 1), (u'return', 2), (u'which', 2), (u'than', 1), (u'Programs', 1), (u'Many', 1), (u'Try', 1), (u'built', 1), (u'YARN', 1), (u'R', 1), (u'using', 5), (u'Example', 1), (u'scala>', 1), (u'Once', 1), (u'DskipTests', 1), (u'Spark'][http://spark.apache.org/docs/latest/building-spark.html].', 1), (u'and', 9), (u'Because', 1), (u'cluster', 1), (u'name', 1), (u'-T', 1), (u'Testing', 1), (u'optimized', 1), (u'streaming', 1), (u'bin/pyspark', 1), (u'SQL', 2), (u'through', 1), (u'GraphX', 1), (u'them', 1), (u'guide][http://spark.apache.org/contributing.html]', 1), (u'[run', 1), (u'analysis', 1), (u'development', 1), (u'abbreviated', 1), (u'set', 2), (u'For', 3), (u'Scala', 2), (u'##', 9), (u'the', 24), (u'thread', 1), (u'library', 1), (u'see', 3), (u'individual', 1), (u'examples', 2), (u'MASTER', 1), (u'runs.', 1), (u'[Apache', 1), (u'Pi', 1), (u'instructions.', 1), (u'More', 1), (u'Python', 2), (u'#', 1), (u'processing', 1), (u'for', 12), (u'veral', 1), (u'review', 1), (u'its', 1), (u'contributing', 1), (u'This', 2), (u'Developer', 1), (u'version', 1), (u'provides', 1), (u'print', 1), (u'get', 1), (u'Configuration', 1), (u'supports', 2), (u'command', 2), (u'[params]', 1), (u'refer', 2), (u'available', 1), (u'be', 2), (u'Guide][http://spark.apache.org/docs/latest/configuration.html]', 1), (u'run', 7), (u'bin/run-example', 2), (u'Versions', 1), (u'Parallel', 1), (u'Hadoop', 2), (u'Documentation', 1), (u'use', 3), (u'downloaded', 1), (u'distributions.', 1), (u'Spark.', 1), (u'example', 1), (u'by', 1), (u'package', 1), (u'Maven'][http://maven.apache.org/].', 1), (u'Building', 1), (u'thread', 1), (u'package', 1), (u'of', 5), (u'changed', 1), (u'programming', 1), (u'Spark', 16), (u'against', 1), (u'site', 1), (u'Maven', 1), (u'3'][https://cwiki.apache.org/confluence/display/MAVEN/Parallel-builds-in-Maven+3].', 1), (u'or', 3), (u'comes', 1), (u'first', 1), (u'info', 1), (u'contains', 1), (u'can', 7), (u'overview', 1), (u'package.', 1), (u'Please', 4), (u'one', 3), (u'Contributing', 1), (u'You', 1), (u'Online', 1), (u'tools', 1), (u'your', 1), (u'page][http://spark.apache.org/documentation.html].', 1), (u'threads.', 1), (u'Tests', 1), (u'fast', 1), (u'from', 1), (u'[project', 1), (u'APIs', 1), (u'>', 1), (u'SparkPi', 2), (u'locally', 2), (u'system', 1), (u'submit', 1), (u'examples', 2), (u'systems.', 1), (u'start', 1), (u'IDE', 1), (u'params', 1), (u'build/mvn', 1), (u'way', 1), (u'basic', 1), (u'README', 1), (u'<http://spark.apache.org/>', 1), (u'It', 2), (u'graphs', 1), (u'more', 1), (u'engine', 1), (u'project', 1), (u'option', 1), (u'on', 7), (u'started', 1), (u'Note', 1), (u'N', 1), (u'usage', 1), (u'versions', 1), (u'DataFrames', 1), (u'particular', 2), (u'instance', 1), (u'bin/spark-shell', 1), (u'general', 3), (u'with', 4), (u'easiest', 1), (u'protocols', 1), (u'must', 1), (u'And', 1), (u'builds', 1), (u'developing', 1), (u'this', 1), (u'setup', 1), (u'shell!', 2), (u'will', 1), (u'./bin/run-example', 1), (u'following', 2), (u'Hadoop-supported', 1), (u'distribution', 1), (u'Maven', 1), (u'example', 3), (u're', 1), (u'detailed', 2), (u>Data', 1), (u'mesos://', 1), (u'stream', 1), (u'computing', 1), (u'URL', 1), (u'is', 6), (u'in', 6), (u'higher-level', 1), (u'tests', 2), (u'1000:', 2), (u'an', 4), (u'sample', 1), (u'To', 2), (u'tests][http://spark.apache.org/developer-tools.html#Individual-tests].', 1), (u'tips', 1), (u'at', 2), (u'have', 1), (u'1000.count()', 1), (u'[Specifying', 1), (u'[building', 1), (u'You', 4), (u'configure', 1), (u'information', 1), (u'different', 1), (u'Tools'][http://spark.apache.org/developer-tools.html].', 1), (u'MASTER=spark://host:7077', 1), (u'no', 1), (u'not', 1), (u'Java', 1), (u'that', 2), (u'storage', 1), (u'documentation', 1), (u'same', 1), (u'machine', 1), (u'how', 3), (u'need', 1), (u'other', 1), (u'build', 4), (u'prefer', 1), (u'online', 1), (u'you', 4), (u'if', 4), (u'[Contribution', 1), (u'A', 1), (u'About', 1), (u'HDFS', 1), (u'[configuration', 1), (u'sc.parallelize(1', 1), (u'locally', 1), (u'Hive', 2), (u'Useful', 1), (u'running', 1), (u'uses', 1), (u'a', 8), (u'version][http://spark.apache.org/docs/latest/building-spark.html#specifying-the-hadoop-version]', 1), (u'variable', 1), (u'The', 1), (u'data', 1), (u'class', 2), (u'built', 1), (u'building', 2), (u'yarn', 1), (u'Python', 2), (u'Thriftserver', 1), (u'processing', 1), (u'programs', 2), (u'requires', 1), (u'documentation', 3), (u'pre-built', 1), (u'Alternatively', 1), (u'programs', 1), (u'local[N]', 1), (u'Spark][#building-spark].', 1), (u'clean', 1), (u'<class>', 1), (u'spark://', 1), (u'learning', 1), (u'core', 1), (u'talk', 1), (u'latest', 1)]
```

```
>>> wordCounts = textFile.flatMap(lambda line: line.split()).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)
>>> wordCounts.collect()
[(u'project.', 1), (u'help', 1), (u'when', 1), (u'Hadoop', 3), (u'MLLib', 1), (u'"local"', 1), (u'./dev/run-tests', 1), (u'including', 4), (u'graph', 1), (u'computation', 1), (u'file', 1), (u'high-level', 1), (u'find', 1), (u'web', 1), (u'shell', 2), (u'cluster', 2), (u'also', 4), (u'using:', 1), (u'Big', 1), (u'guidance', 2), (u'run:', 1), (u'scala', 1), (u'Running', 1), (u'should', 2), (u'environment', 1), (u'to', 17), (u'only', 1), (u'module', 1), (u'given.', 1), (u'rich', 1), (u'directory', 1), (u'Apache', 1), (u'Interactive', 2), (u'sc.parallelize(range(1000)).count()', 1), (u'Building', 1), (u'do', 2), (u'guide', 1), (u'return', 2), (u'which', 2), (u'than', 1), (u'Programs', 1), (u'Many', 1), (u'Try', 1), (u'built', 1), (u'YARN', 1), (u'R', 1), (u'using', 5), (u'Example', 1), (u'scala>', 1), (u'Once', 1), (u'DskipTests', 1), (u'Spark'][http://spark.apache.org/docs/latest/building-spark.html].', 1), (u'and', 9), (u'Because', 1), (u'cluster', 1), (u'name', 1), (u'-T', 1), (u'Testing', 1), (u'optimized', 1), (u'streaming', 1), (u'bin/pyspark', 1), (u'SQL', 2), (u'through', 1), (u'GraphX', 1), (u'them', 1), (u'guide][http://spark.apache.org/contributing.html]', 1), (u'[run', 1), (u'analysis', 1), (u'development', 1), (u'abbreviated', 1), (u'set', 2), (u'For', 3), (u'Scala', 2), (u'##', 9), (u'the', 24), (u'thread', 1), (u'library', 1), (u'see', 3), (u'individual', 1), (u'examples', 2), (u'MASTER', 1), (u'runs.', 1), (u'[Apache', 1), (u'Pi', 1), (u'instructions.', 1), (u'More', 1), (u'Python', 2), (u'#', 1), (u'processing', 1), (u'for', 12), (u'veral', 1), (u'review', 1), (u'its', 1), (u'contributing', 1), (u'This', 2), (u'Developer', 1), (u'version', 1), (u'provides', 1), (u'print', 1), (u'get', 1), (u'Configuration', 1), (u'supports', 2), (u'command', 2), (u'[params]', 1), (u'refer', 2), (u'available', 1), (u'be', 2), (u'Guide][http://spark.apache.org/docs/latest/configuration.html]', 1), (u'run', 7), (u'bin/run-example', 2), (u'Versions', 1), (u'Parallel', 1), (u'Hadoop', 2), (u'Documentation', 1), (u'use', 3), (u'downloaded', 1), (u'distributions.', 1), (u'Spark.', 1), (u'example', 1), (u'by', 1), (u'package', 1), (u'Maven'][http://maven.apache.org/].', 1), (u'Building', 1), (u'thread', 1), (u'package', 1), (u'of', 5), (u'changed', 1), (u'programming', 1), (u'Spark', 16), (u'against', 1), (u'site', 1), (u'Maven', 1), (u'3'][https://cwiki.apache.org/confluence/display/MAVEN/Parallel-builds-in-Maven+3].', 1), (u'or', 3), (u'comes', 1), (u'first', 1), (u'info', 1), (u'contains', 1), (u'can', 7), (u'overview', 1), (u'package.', 1), (u'Please', 4), (u'one', 3), (u'Contributing', 1), (u'You', 1), (u'Online', 1), (u'tools', 1), (u'your', 1), (u'page][http://spark.apache.org/documentation.html].', 1), (u'threads.', 1), (u'Tests', 1), (u'fast', 1), (u'from', 1), (u'[project', 1), (u'APIs', 1), (u'>', 1), (u'SparkPi', 2), (u'locally', 2), (u'system', 1), (u'submit', 1), (u'examples', 2), (u'systems.', 1), (u'start', 1), (u'IDE', 1), (u'params', 1), (u'build/mvn', 1), (u'way', 1), (u'basic', 1), (u'README', 1), (u'<http://spark.apache.org/>', 1), (u'It', 2), (u'graphs', 1), (u'more', 1), (u'engine', 1), (u'project', 1), (u'option', 1), (u'on', 7), (u'started', 1), (u'Note', 1), (u'N', 1), (u'usage', 1), (u'versions', 1), (u'DataFrames', 1), (u'particular', 2), (u'instance', 1), (u'bin/spark-shell', 1), (u'general', 3), (u'with', 4), (u'easiest', 1), (u'protocols', 1), (u'must', 1), (u'And', 1), (u'builds', 1), (u'developing', 1), (u'this', 1), (u'setup', 1), (u'shell!', 2), (u'will', 1), (u'./bin/run-example', 1), (u'following', 2), (u'Hadoop-supported', 1), (u'distribution', 1), (u'Maven', 1), (u'example', 3), (u're', 1), (u'detailed', 2), (u>Data', 1), (u'mesos://', 1), (u'stream', 1), (u'computing', 1), (u'URL', 1), (u'is', 6), (u'in', 6), (u'higher-level', 1), (u'tests', 2), (u'1000:', 2), (u'an', 4), (u'sample', 1), (u'To', 2), (u'tests][http://spark.apache.org/developer-tools.html#Individual-tests].', 1), (u'tips', 1), (u'at', 2), (u'have', 1), (u'1000.count()', 1), (u'[Specifying', 1), (u'[building', 1), (u'You', 4), (u'configure', 1), (u'information', 1), (u'different', 1), (u'Tools'][http://spark.apache.org/developer-tools.html].', 1), (u'no', 1), (u'not', 1), (u'Java', 1), (u'that', 2), (u'storage', 1), (u'documentation', 1), (u'same', 1), (u'machine', 1), (u'how', 3), (u'need', 1), (u'other', 1), (u'build', 4), (u'prefer', 1), (u'online', 1), (u'you', 4), (u'if', 4), (u'[Contribution', 1), (u'A', 1), (u'About', 1), (u'HDFS', 1), (u'[configuration', 1), (u'sc.parallelize(1', 1), (u'locally', 1), (u'Hive', 2), (u'Useful', 1), (u'running', 1), (u'uses', 1), (u'a', 8), (u'version][http://spark.apache.org/docs/latest/building-spark.html#specifying-the-hadoop-version]', 1), (u'variable', 1), (u'The', 1), (u'data', 1), (u'class', 2), (u'built', 1), (u'building', 2), (u'yarn', 1), (u'Python', 2), (u'Thriftserver', 1), (u'processing', 1), (u'programs', 2), (u'requires', 1), (u'documentation', 3), (u'pre-built', 1), (u'Alternatively', 1), (u'programs', 1), (u'local[N]', 1), (u'Spark][#building-spark].', 1), (u'clean', 1), (u'<class>', 1), (u'spark://', 1), (u'learning', 1), (u'core', 1), (u'talk', 1), (u'latest', 1)]
```

9

```
>>> exit()
```

```
>>> exit()
bigdata@bigdata:~/spark$
```

3.9.5. Ejemplo de Spark SQL

En este ejemplo, cargaremos un archivo JSON en Spark y lo consultaremos a través de consultas SQL:

1

```
mkdir /home/bigdata/ejemplosSpark
```

```
sudo nano /home/bigdata/ejemplosSpark/genios.json
```

```
bigdata@bigdata:~$ mkdir /home/bigdata/ejemplosSpark
bigdata@bigdata:~$ sudo nano /home/bigdata/ejemplosSpark/genios.json
```

2

```
{ "nombre" : "Leonhard Paul" , "apellido" : "Euler" , "campos" : [ { "campo" : "física" } , { "campo" : "matemáticas" } ] , "nacimiento" : { "fecha" : "1973-09-18" , "ciudad" : "Basilea" , "pais" : "Suiza" } }
```

```
{ "nombre" : "Johann Karl Friedrich" , "apellido" : "Gauss" , "campos" : [ { "campo" : "matemáticas" } , { "campo" : "astronomía" } , { "campo" : "geodesia" } , { "campo" : "física" } ] , "nacimiento" : { "fecha" : "1777-04-30" , "ciudad" : "Brunswick" , "pais" : "Alemania" } }
```

```
{ "nombre" : "Giuseppe Lodovico" , "apellido" : "Lagrange" , "campos" : [ { "campo" : "física" } , { "campo" : "matemáticas" } , { "campo" : "astronomía" } ] , "nacimiento" : { "fecha" : "1736-01-25" , "ciudad" : "Turín" , "pais" : "Italia" } }
```

```
{ "nombre" : "Werner" , "apellido" : "Heisenberg" , "campos" : [ { "campo" : "física" } ] , "nacimiento" : { "fecha" : "1901-12-05" , "ciudad" : "Wurzburgo" , "pais" : "Alemania" } }
```

```
{ "nombre" : "Roger" , "apellido" : "Penrose" , "campos" : [ { "campo" : "física" } , { "campo" : "matemáticas" } ] , "nacimiento" : { "fecha" : "1931-08-08" , "ciudad" : "Colchester" , "pais" : "Inglaterra" } }
```

```
[{"nombre" : "Leonhard Paul" , "apellido" : "Euler" , "campos" : [ { "campo" : "física" } , { "campo" : "matemáticas" } ] , "nacimiento" : { "fecha" : "1973-09-18" , "ciudad" : "Basilea" , "pais" : "Suiza" } , "nacimiento" : { "fecha" : "1777-04-30" , "ciudad" : "Brunswick" , "pais" : "Alemania" } , "nacimiento" : { "fecha" : "1736-01-25" , "ciudad" : "Turín" , "pais" : "Italia" } , "nacimiento" : { "fecha" : "1901-12-05" , "ciudad" : "Wurzburgo" , "pais" : "Alemania" } , "nacimiento" : { "fecha" : "1931-08-08" , "ciudad" : "Colchester" , "pais" : "Inglaterra" } ]
```

3

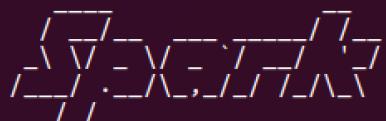
```
hdfs dfs -mkdir /ejemplosSpark
hdfs dfs -put /home/bigdata/ejemplosSpark/genios.json /ejemplosSpark
```

```
bigdata@bigdata:~$ hdfs dfs -mkdir /ejemplosSpark
bigdata@bigdata:~$ hdfs dfs -put /home/bigdata/ejemplosSpark/genios.json /ejemplosSpark
```

spark-shell

```
bigdata@bigdata:~$ spark-shell
```

Welcome to


version 2.2.1

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_151)
Type in expressions to have them evaluated.
Type :help for more information.

scala>

4

```
scala> val df = spark.read.json ("/ejemplosSpark/genios.json")
```

```
scala> val df = spark.read.json ("/ejemplosSpark/genios.json")
17/12/08 21:15:12 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
df: org.apache.spark.sql.DataFrame = [apellido: string, campos: array<struct<campo:string>> ... 2 more fields]
```

scala> df.show()

```
scala> df.show()
+-----+-----+-----+-----+
| apellido|      campos|    nacimiento|      nombre|
+-----+-----+-----+-----+
|   Euler|[{"fisica": [{"matem": [{"Basilea": "1973-09-12"}]}], "matem": [{"Brunswick": "1777-08-24"}]}]| 1736-01-25...| Leonhard Paul|
|   Gauss|[{"matematicas": [{"Brunswick": "1777-08-24"}]}]| 1777-08-24...| Johann Karl Fried...
| Lagrange|[{"fisica": [{"matem": [{"Turin": "1736-01-25"}]}]}]| 1736-01-25...| Giuseppe Lodovico|
| Heisenberg|[{"fisica": [{"Colchester": "1901-10-08"}]}]| 1901-10-08...| Werner|
| Penrose|[{"fisica": [{"Colchester": "1931-07-08"}]}]| 1931-07-08...| Roger|
+-----+-----+-----+-----+
```

5

```
scala> df.printSchema()

scala> df.printSchema()
root
|-- apellido: string (nullable = true)
|-- campos: array (nullable = true)
|   |-- element: struct (containsNull = true)
|   |   |-- campo: string (nullable = true)
|-- nacimiento: struct (nullable = true)
|   |-- ciudad: string (nullable = true)
|   |-- fecha: string (nullable = true)
|   |-- pais: string (nullable = true)
|-- nombre: string (nullable = true)
```

6

```
scala> df.select("nombre").show()
```

nombre
Leonhard Paul
Johann Karl Fried...
Giuseppe Lodovico
Werner
Roger

7

```
scala> df.select(df("nombre"),upper(df("apellido"))).show()
```

nombre	upper(apellido)
Leonhard Paul	EULER
Johann Karl Fried...	GAUSS
Giuseppe Lodovico	LAGRANGE
Werner	HEISENBERG
Roger	PENROSE

```
scala> df.filter(df("apellido") > "J").show()
```

apellido	campos	nacimiento	nombre
Lagrange	[física], [matem...]	[Turín, 1736-01-25]	Giuseppe Lodovico
Penrose	[física], [matem...]	[Colchester, 1931-....]	Roger

8

```
scala> df.filter( df("apellido") startsWith("P")).show()
```

apellido	campos	nacimiento	nombre
Penrose	[física], [matem...	[Colchester, 1931-...	Roger

9

```
df.createOrReplaceTempView("genios")
```

```
val res = spark.sql("SELECT * FROM genios WHERE apellido = 'Euler'")
```

```
res.show()
```

```
scala> df.createOrReplaceTempView("genios")
```

```
scala> val res = spark.sql("SELECT * FROM genios WHERE apellido = 'Euler'")  
res: org.apache.spark.sql.DataFrame = [apellido: string, campos: array<struct<campo:string>>]
```

```
2 more fields
```

```
scala> res.show()
```

apellido	campos	nacimiento	nombre
Euler	[física], [matem...	[Basilea, 1973-09-...	Leonhard Paul

3.9.6. Ejemplo de Spark Streaming

En este apartado, usaremos uno de los ejemplos que la instalación de Spark trae por defecto. En concreto, usaremos NetworkWordCount. Este programa es un chat apoyado en Spark Streaming que contará las palabras enviadas a través del terminal.



Figura 31. Fuentes de datos. Fuente: Fundación Apache.

```
bigdata@bigdata:~/spark/examples/src/main/scala/org/apache/spark/examples/streaming$ ls -lrt
total 60
-rw-r--r-- 1 bigdata bigdata 1578 nov 25 00:31 StreamingExamples.scala
-rw-r--r-- 1 bigdata bigdata 2968 nov 25 00:31 StatefulNetworkWordCount.scala
-rw-r--r-- 1 bigdata bigdata 3728 nov 25 00:31 SqlNetworkWordCount.scala
-rw-r--r-- 1 bigdata bigdata 6352 nov 25 00:31 RecoverableNetworkWordCount.scala
-rw-r--r-- 1 bigdata bigdata 2661 nov 25 00:31 RawNetworkGrep.scala
-rw-r--r-- 1 bigdata bigdata 1886 nov 25 00:31 QueueStream.scala
-rw-r--r-- 1 bigdata bigdata 2483 nov 25 00:31 NetworkWordCount.scala
-rw-r--r-- 1 bigdata bigdata 3767 nov 25 00:31 KafkaWordCount.scala
-rw-r--r-- 1 bigdata bigdata 2170 nov 25 00:31 HdfsWordCount.scala
-rw-r--r-- 1 bigdata bigdata 2435 nov 25 00:31 FlumePollingEventCount.scala
-rw-r--r-- 1 bigdata bigdata 2561 nov 25 00:31 FlumeEventCount.scala
-rw-r--r-- 1 bigdata bigdata 2700 nov 25 00:31 DirectKafkaWordCount.scala
-rw-r--r-- 1 bigdata bigdata 3817 nov 25 00:31 CustomReceiver.scala
drwxr-xr-x 2 bigdata bigdata 4096 nov 25 00:31 clickstream
```

1

Ejecutando el siguiente comando podremos ver el código del ejemplo:

```
$ cat
```

```
$SPARK_HOME/examples/src/main/scala/org/apache/spark/examples/streaming/NetworkWordC
```

```
package org.apache.spark.examples.streaming
```

```
import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.storage.StorageLevel
```

```
object NetworkWordCount {
  def main(args: Array[String]) {
    if (args.length < 2) {
      System.err.println("Usage: NetworkWordCount <hostname> <port>")
      System.exit(1)
    }
  }
}
```

```
StreamingExamples.setStreamingLogLevels()
```

```
// Create the context with a 1 second batch size
val sparkConf = new SparkConf().setAppName("NetworkWordCount")
val ssc = new StreamingContext(sparkConf, Seconds(1))
```

```
val lines = ssc.socketTextStream(args(0), args(1).toInt, StorageLevel.MEMORY_AND_DISK_SER)
val words = lines.flatMap(_.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
wordCounts.print()
ssc.start()
ssc.awaitTermination()
```

2

jps

```
bigdata@bigdata:~$ jps
32578 NameNode
3363 Jps
3000 ResourceManager
3147 NodeManager
507 SecondaryNameNode
32734 DataNode
```

3

cd \$SPARK_HOME

```
bigdata@bigdata:~$ cd $SPARK_HOME
```

4

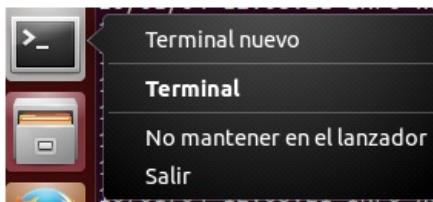
```
./bin/spark-submit \
--class org.apache.spark.examples.streaming.NetworkWordCount \
--master local[2] \
/home/bigdata/spark/examples/jars/spark-examples_2.11-2.2.1.jar \
localhost 9999
```

```
bigdata@bigdata:~/spark$ ./bin/spark-submit \
> --class org.apache.spark.examples.streaming.NetworkWordCount \
> --master local[2] \
> /home/bigdata/spark/examples/jars/spark-examples_2.11-2.2.1.jar \
> localhost 9999
SLF4J: Class path contains multiple SLF4J bindings.
```

```
17/12/08 21:24:46 ERROR ReceiverTracker: Deregistered receiver for stream 0: Restarting receiver with delay 2000ms: Error connecting to localhost:9999 - java.net.ConnectException: Conexión rehusada (Connection refused)
    at java.net.PlainSocketImpl.socketConnect(Native Method)
    at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
    at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
    at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
    at java.net.Socket.connect(Socket.java:589)
    at java.net.Socket.connect(Socket.java:538)
    at java.net.Socket.<init>(Socket.java:434)
```

5

Una vez llegados a este punto, tendremos que abrir un nuevo terminal.



```
$ nc -lk 9999
```

```
bigdata@bigdata:~$ nc -lk 9999
Hola mundo
```

En la terminal anterior ejecutamos el siguiente comando:

```
./bin/run-example streaming.NetworkWordCount localhost 9999
```

De esta forma, todo lo que escribamos en la nueva terminal, vía Streaming será procesado en la primera terminal mostrándonos el WorCount en tiempo real.

```
Time: 1512764869000 ms
(Hola,1)
(mundo,1)
```

IV. Lectura recomendada



A continuación, se recomiendan las siguientes lecturas para profundizar en algunos conceptos sobre Spark una vez finalizado el programa.

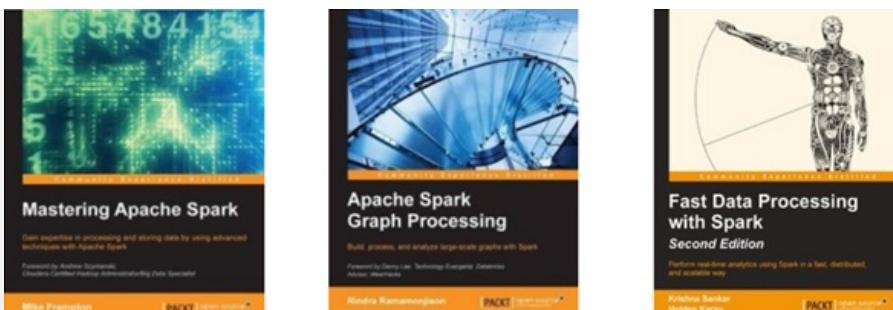


Figura 32. Portadas de las lecturas recomendadas. Fuente: Amazon.es

- *Mastering Apache Spark*, de Mike Frampton (2015).
- *Apache Spark Graph Processing*, de Rindra Ramamondjison (2015).
- *Fast Data Processing with Spark*, de Krishna Sankar y Holden Karau (2015).

V. Resumen



En esta unidad, hemos visto cómo en la lucha por obtener un mayor rendimiento y una mayor facilidad de uso de cara a los desarrolladores han aparecido soluciones tan interesantes como Spark. Spark elimina muchas de las limitaciones que nos aportaba Hadoop y nos ofrece componentes para cubrir necesidades que antes no eran triviales. Spark Streaming o MLlib son buenas pruebas de ello. Hemos repasado sus componentes principales, practicado ejercicios simples en algunos de ellos y hemos visto las diferencias principales si lo comparamos con su antecesor.



Análisis de un dataframe a través de Spark SQL



Pincha [aquí](#) para descargar los archivos necesarios para seguir el tutorial.

Ejercicios

Ejercicio



Objetivo: poner en práctica los conocimientos adquiridos sobre la herramienta para resolver el ejercicio.

1

Crea un archivo con los JSON siguientes:

```
{"year":1967,"Title":"The Piper at the Gates of Dawn","USA_Ranking":131,"Uk_Ranking":6}
```

```
{"year":1968,"Title":"A Saucerful of Secrets","USA_Ranking":999,"Uk_Ranking":9}
```

```
{"year":1969,"Title":"Music from the Film More","USA_Ranking":153,"Uk_Ranking":9},  
{"year":1969,"Title":"Ummagumma","USA_Ranking":74,"Uk_Ranking":5}
```

```
{"year":1970,"Title":"Atom Heart Mother","USA_Ranking":55,"Uk_Ranking":1}
```

```
{"year":1972,"Title":"Obscured by Clouds","USA_Ranking":46,"Uk_Ranking":6}
```

```
{"year":1973,"Title":"The Dark Side of the Moon","USA_Ranking":1,"Uk_Ranking":1}
```

```
{"year":1975,"Title":"Wish you Were Here","USA_Ranking":1,"Uk_Ranking":1}
```

```
{"year":1977,"Title":"Animals","USA_Ranking":3,"Uk_Ranking":2}
```

```
{"year":1979,"Title":"The Wall","USA_Ranking":1,"Uk_Ranking":3}
```

```
{"year":1983,"Title":"The Final Cut","USA_Ranking":6,"Uk_Ranking":1}
```

```
{"year":1987,"Title":"A Momentary Lapse of Reason","USA_Ranking":3,"Uk_Ranking":3}
```

```
{"year":1994,"Title":"The Division Bell","USA_Ranking":1,"Uk_Ranking":1}
```

```
{"year":2014,"Title":"The Endless River","USA_Ranking":3,"Uk_Ranking":1}
```

2

Carga este archivo en HDFS.

3

Genera un dataframe desde el archivo que acabas de subir a HDFS.

4

Recupera el esquema del dataframe.

5

Crea una vista temporal con el comando `createOrReplaceTempView`.

6

Calcula los discos que ocuparon a la vez posiciones entre las cinco primeras en ambos rankings.

7

Obtén la máxima y mínima posición que ocuparon los discos de Pink Floyd en EE. UU. y en Reino Unido.

8

Obtén los títulos de los discos en mayúsculas.

Solución

1

```
sudo mkdir /home/bigdata/ejercicio
```

```
sudo nano /home/bigdata/ejercicio/PinkFloyd.json
```

2

Carga este archivo en HDFS:

```
hdfs dfs -mkdir /ejercicio
```

```
hdfs dfs -put /home/bigdata/ejercicio/PinkFloyd.json /ejercicio
```

3

Genera un dataframe desde el archivo que acabas de subir a HDFS:

```
scala> val df = spark.read.json ("/ejercicio/PinkFloyd.json")
scala> df.show()
```

```
scala> val df = spark.read.json ("/ejercicio/PinkFloyd.json")
18/01/28 17:16:22 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
df: org.apache.spark.sql.DataFrame = [Title: string, USA_Ranking: bigint ... 2 more fields]
scala> df.show()
+-----+-----+-----+
| Title|USA_Ranking|Uk_Ranking|year|
+-----+-----+-----+
| The Piper at the ...| 131|       6|1967|
| A Saucerful of Se...| 999|       9|1968|
| Music from the Fi...| 153|       9|1969|
| Atom Heart Mother| 55|       1|1970|
| Obscured by Clouds| 46|       6|1972|
| The Dark Side of ...| 1|       1|1973|
| Wish you Were Here| 1|       1|1975|
|           Animals| 3|       2|1977|
|           The Wall| 1|       3|1979|
| The Final Cut| 6|       1|1983|
| A Momentary Lapse...| 3|       3|1987|
| The Division Bell| 1|       1|1994|
| The Endless River| 3|       1|2014|
+-----+-----+-----+
```

4

Recupera el esquema del dataframe:

```
scala> df.printSchema()
```

```
scala>
scala> df.printSchema()
root
|- Title: string (nullable = true)
|- USA_Ranking: long (nullable = true)
|- Uk_Ranking: long (nullable = true)
|- year: long (nullable = true)
```

5

Crea una vista temporal con el comando createOrReplaceTempView:

```
df.createOrReplaceTempView("PinkFloyd")
```

6

Calcula los discos que ocuparon a la vez posiciones entre las cinco primeras en ambos rankings:

```
val res = spark.sql("SELECT * FROM PinkFloyd WHERE USA_Ranking < 5 AND Uk_Ranking < 5")  
res.show()  
  
scala> df.createOrReplaceTempView("PinkFloyd")  
scala> val res = spark.sql("SELECT * FROM PinkFloyd WHERE USA_Ranking < 5 AND Uk_Ranking < 5")  
res: org.apache.spark.sql.DataFrame = [Title: string, USA_Ranking: bigint ... 2 more fields]  
scala> res.show()  
+-----+-----+-----+---+  
|       Title|USA_Ranking|Uk_Ranking|year|  
+-----+-----+-----+---+  
|The Dark Side of ...|      1|        1|1973|  
| Wish you Were Here|      1|        1|1975|  
|       Animals|      3|        2|1977|  
|       The Wall|      1|        3|1979|  
|A Momentary Lapse...|      3|        3|1987|  
|   The Division Bell|      1|        1|1994|  
|  The Endless River|      3|        1|2014|  
+-----+-----+-----+---+
```

Obtén la máxima y mínima posición que ocuparon los discos de Pink Floyd en EE. UU. y en Reino Unido:

```
val max_USA = spark.sql("SELECT Max(USA_Ranking) FROM PinkFloyd ")
```

```
max_USA.show()
```

```
scala> val max_USA = spark.sql("SELECT Max(USA_Ranking) FROM PinkFloyd ")
max_USA: org.apache.spark.sql.DataFrame = [max(USA_Ranking): bigint]

scala> max_USA.show()
+-----+
|max(USA_Ranking)|
+-----+
|          999|
+-----+
```

```
val max_UK = spark.sql("SELECT Max(Uk_Ranking) FROM PinkFloyd ")
```

```
max_UK.show()
```

```
val min_USA = spark.sql("SELECT Min(Uk_Ranking) FROM PinkFloyd ")
```

```
min_USA.show()
```

```
val min_UK = spark.sql("SELECT Min(Uk_Ranking) FROM PinkFloyd ")
```

```
min_UK.show()
```

Obtén los títulos de los discos en mayúsculas:

```
df.select(upper(df("Title"))).show()
```

```
scala> df.select(upper(df("Title"))).show()
+-----+
|      upper>Title)|
+-----+
| THE PIPER AT THE ... |
| A SAUCERFUL OF SE... |
| MUSIC FROM THE FI... |
|     ATOM HEART MOTHER |
|    OBSCURED BY CLOUDS |
| THE DARK SIDE OF ... |
|     WISH YOU WERE HERE |
|           ANIMALS |
|           THE WALL |
|        THE FINAL CUT |
| A MOMENTARY LAPSE... |
|     THE DIVISION BELL |
|     THE ENDLESS RIVER |
+-----+
```

Recursos

Enlaces de Interés



Web de Apache

<https://spark.apache.org/>

Bibliografía

- **Learning Spark: Lightning-Fast Big Data Analysis.** : Karau, Holden y Konwinsky, Andy. Sebastopol, CA: O'Reilly Media; 2015.
- **Mastering Apache Spark.** : Frampton, Mike. Birmingham: Packt Publishing; 2015.

Glosario.

- **Procesamiento en streaming:** Este tipo de técnicas de procesamiento y análisis de datos se basan en la implementación de un modelo de flujo de datos en el que los datos fluyen continuamente a través de una red de entidades de transformación que componen el sistema.
- **Teoría de grafos:** La teoría de grafos es una rama de las matemáticas y las ciencias de la computación que estudia las propiedades de los grafos. La teoría de grafos tiene sus fundamentos en las matemáticas discretas y las matemáticas aplicadas. Esta teoría requiere diferentes conceptos de diversas áreas como combinatoria, álgebra, probabilidad, geometría de polígonos, aritmética y topología.