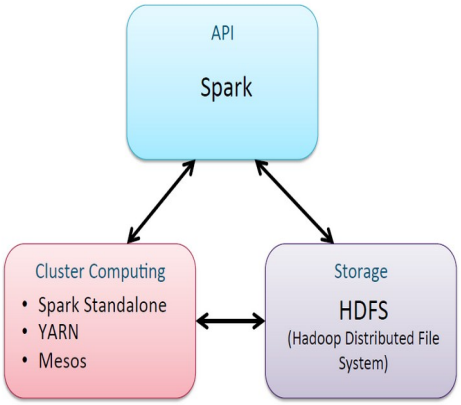
Introducción a Apache Spark

Introducción al análisis de datos con Spark

Historia de Spark



- Proyecto open source creado y mantenido por la comunidad de desarrolladores
- Fue creado en 2009 en la universidad de Berkley, como respuesta a la ineficiencia de MapReduce(MR) para procesos iterativos e interactivos
- Poco después de su creación, se publicó que Spark era entre 10 y 20 veces más rápido que MR
- Enseguida captó la atención de importantes empresas como: Databricks, Yahoo! o Intel
- En 2011 se empezaron a añadir componentes de mayor nivel como Hive sobre Spark
- En 2013 pasó a formar parte de la Apache Software Foundation, donde ahora es uno de sus proyectos top level.

Ejemplos SPARK

```scala
val lines = sc.parallelize(List("hello world", "hi"))
val words = lines.flatMap(line => line.split(" "))
words.first() // returns "hello"
```

Ejemplo Scala

```python
lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
words.first() # returns "hello"
```

Ejemplo Python

```java
JavaRDD<String> lines = sc.parallelize(Arrays.asList("hello world", "hi"));
JavaRDD<String> words = lines.flatMap(new FlatMapFunction<String, String>() {
public Iterable<String> call(String line) {
return Arrays.asList(line.split(" "));
}
});
words.first(); // returns "hello"
```

Ejemplo Java

Por qué Scala con Spark

Normalmente, incluso hoy en día, muchísima analítica se hace con R, Python, Matlab, Weka, SPSS …

En empresas muy muy grandes, se pueden generar datasets mayores que la memoria disponible en los servidores; por lo que para poder procesar tantos datos tenemos que migrar el código a otro entorno o lenguaje.

Empresas del sector de telecomunicaciones, RRSS, banca, IOT…

SPARK forma parte del ecosistema HADOOP, aunque no necesita de ninguna de sus herramientas para ejecutarse. SI tiene completa compatibilidad con dichas herramientas.

Spark, su API principal y el Spark Shell están escritos en Scala, por lo que SPARK utiliza multitud de funciones de Scala.

Utilizar Scala con Spark nos proporciona una forma de implementar paralelismo sin necesidad de utilizar las librerías existentes para ello (como puede ser en Java la librería Thread). Pues el framework SPARK distribuye automáticamente el trabajo entre los diferentes nodos.

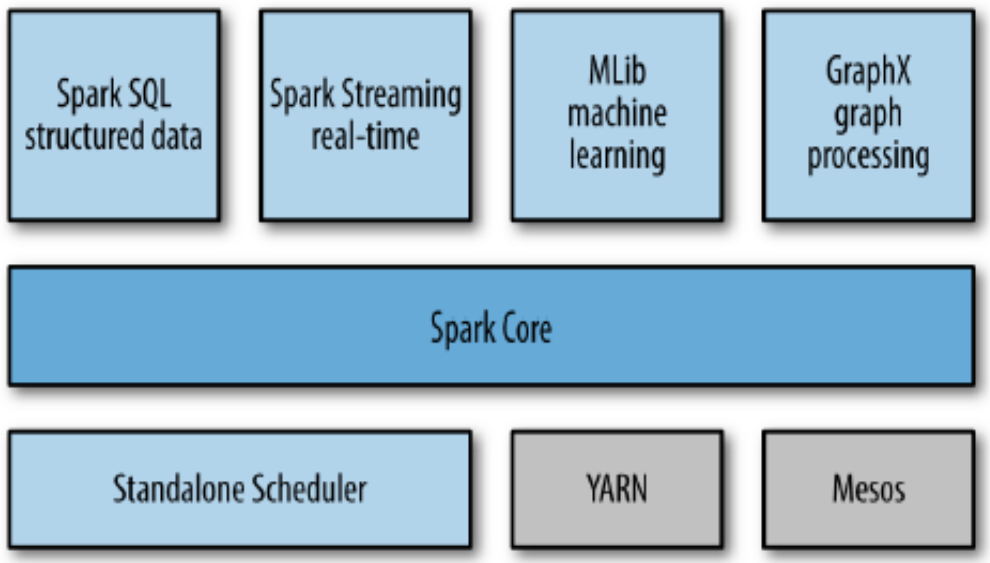
Ofrece mejor rendimiento.

Y no solo en términos de tiempo de ejecución.
Sino en términos de interactividad: se puede ejecutar una query sobre datos en streaming y generar un nuevo streaming a partir de ellos combinable con otros datos para su posterior análisis. Por ejemplo, en webanalytics, podemos ejecutar queries contra datos que se están produciendo ahora mismo.
Es una extensión del modelo MapReduce cuyo objetivo es soportar más tipos de computación, específicamente en streaming, con idea de poder explorar los datos interactivamente, en lugar de tener que esperar horas.

Spark

Para llevar a cabo toda esta computación utilizaremos Apache Spark, que no es más que un framework para programación distribuida sobre datos en paralelo.
Para ello, Spark implementa un modelo de datos paralelos distribuidos llamado RDD: Resilient Distributed Dataset.
A partir de ahora, tenemos que pensar en nuestros datos distribuidos como si fueran una única colección.
Es algo de lo que no podemos prescindir. Hemos de tenerlo en mente cuando programamos.

Arquitectura básica



Introducción al análisis de datos con Spark

Spark Core
Contiene la funcionalidad básica de *Spark*, como programación de tareas, gestión de memoria, recuperación ante fallos, interacción con sistemas de almacenamiento y otras cosas.
También es el lugar donde se aloja la API que define los RDD (*Resilient Distributed Dataset*), que son la abstracción de *Spark* más importante y que representan una colección de elementos distribuidos a través de los nodos del clúster y que pueden ser manipulados en paralelo.

Spark SQL
Antiguamente llamado *Shark.*
Es el paquete de *Spark* orientado a trabajar con datos estructurados a través de SQL o HQL.
Soporta varios tipos de datos como *Hive, Parquet o JSON.*
Permite intercalar comandos de manipulación de *RDDs*, lo que permite realizar análisis de datos muy potentes.

Spark Streaming
Es un componente de *Spark* que permite procesar *streams* de datos en tiempo real, por ejemplo *weblogs* o *serverlogs* mientras se van generando

Mllib
Es una librería que contiene funcionalidades comunes de *Machine Learning.*
Algunas de ellas son Clasificación, Regresión, *Clustering*, Filtros Colaborativos, evaluación de modelos e importación de datos.
También provee primitivas de más bajo nivel.

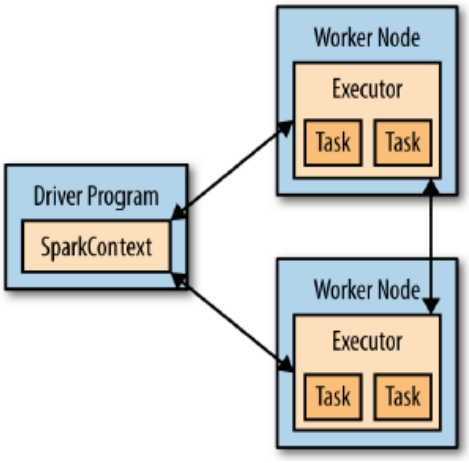
GraphX
Es una librería para manipular grafos y ejecutar computación paralela sobre ellos.

Versión de *Spark*
Actualmente la 2.2

Capa de almacenamiento aceptadas por *Spark*
HDFS
Cualquier otra soportada por *Hadoop* como *Amazon S3, Cassandra, Hive, HBase*, etc.
Spark no requiere de *Hadoop* para funcionar. Simplemente aporta un soporte para almacenamiento implementando la API de *Hadoop.*
Spark soporta *TextFiles, SequenceFiles, Avro, Parket* y cualquier otro formato de entrada soportado por *Hadoop.*

Comenzando a trabajar con Spark

Introducción a los conceptos de Spark Core



En alto nivel, las aplicaciones *Spark* tienen un *driver* que lanza varias operaciones en paralelo en un clúster.
El driver contiene la función *main* y define los *datasets* distribuidos en el clúster. Luego aplica la función sobre ellos.
Los *drivers* acceden a *Spark* a través del objeto *SparkContext (sc)*, que representa la conexión con el clúster.
Cuando trabajamos a través del *shell*, el *SparkContext* se crea automáticamente.
Para correr estas operaciones, el driver típicamente maneja un grupo de nodos del clúster llamados *Executors* (Ejecutores).