Introducción a Apache Spark

Plataforma de computación cluster rápida

- Extiende modelo MapReduce soportando de manera eficiente otros tipos de computación:
  - queries interactivas
  - procesado *streaming*
- Soporta computaciones en memoria
- Mejora a MapReduce para aplicaciones complejas (10-20x más rápido)

Propósito general

- Modos de funcionamiento batch, interactivo o streaming
- Reduce el número de herramientas a emplear y mantener

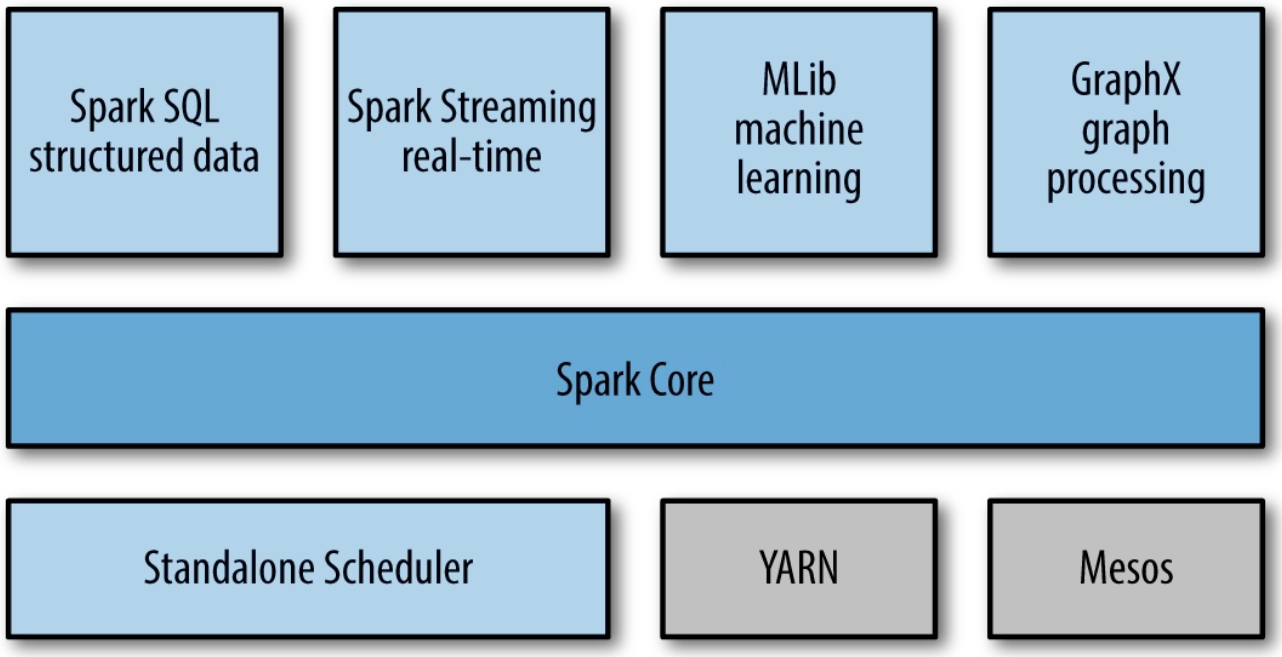
Historia

- Iniciado en el 2009 en el UC Berkeley RAD Lab (AMPLab)
  - Motivado por la ineficiencia de MapReduce para trabajos iterativos e interactivos
- Mayores contribuidores: Databricks, Yahoo! e Intel
- Declarado open source en marzo del 2010
- Transferido a la Apache Software Foundation en junio de 2013, TLP en febrero de 2014
- Uno de los proyectos Big Data más activos
- Versión 1.0 lanzada en mayo de 2014

Características de Spark

- Soporta gran variedad de workloads: batch, queries interactivas, streaming, machine learning, procesado de grafos
- APIs en Scala, Java, Python, SQL y R
- Shells interactivos en Scala, Python, SQL y R
- Se integra con otras soluciones BigData: HDFS, Cassandra, etc.

La pila Spark



(Fuente: H. Karau, A. Konwinski, P. Wendell, M. Zaharia, "Learning Spark", O'Reilly, 2015)

APIs del Spark Core

Spark ofrece dos APIs:

- API estructurada o de alto nivel
- API de bajo nivel

Cada API ofrece diferentes tipos de datos:

- Se recomienda usar la API estructurada por su mayor rendimiento
- La API de bajo nivel permite un mayor control sobre la distribución de los datos
- La API de alto nivel utiliza las primitivas de bajo nivel

Tipos de datos en la API estructurada

Datasets

Colección distribuida de objetos del mismo tipo

- Introducida en Spark > 1.6
- El API para Datasets sólo está disponible en Scala y Java
- No está disponible en Python ni R debido al tipado dinámico de estos lenguages

DataFrames

Un DataFrame es un DataSet organizado en columnas con nombre

- Conceptualmente equivalente a una tabla en una base de datos relacional o un dataframe en Python Pandas o R
- El API para DataFrames está disponible en Scala, Java, Python y R
- En Java y Scala, un DataFrame es un DataSet de objetos de tipo *Row*

Tipos de datos en la API de bajo nivel

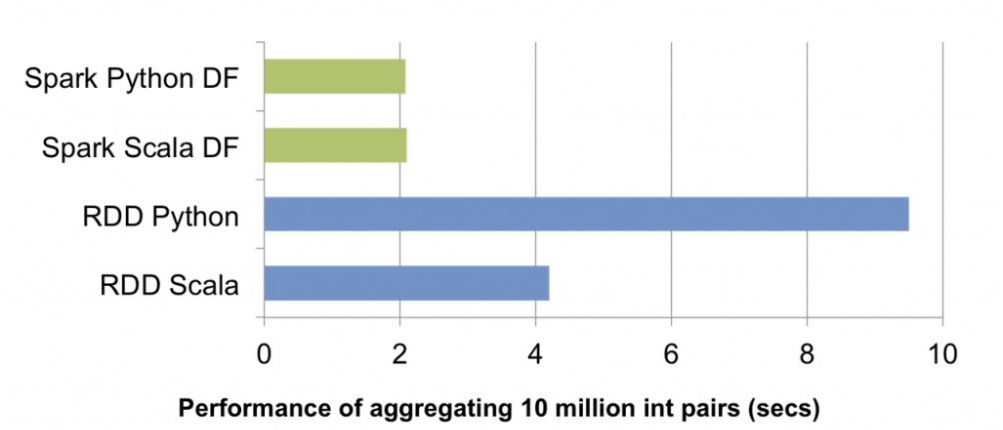
RDDs (Resilient Distributed Datasets)

Lista distribuida de objetos

- Tipos de datos básico de Spark v1.X

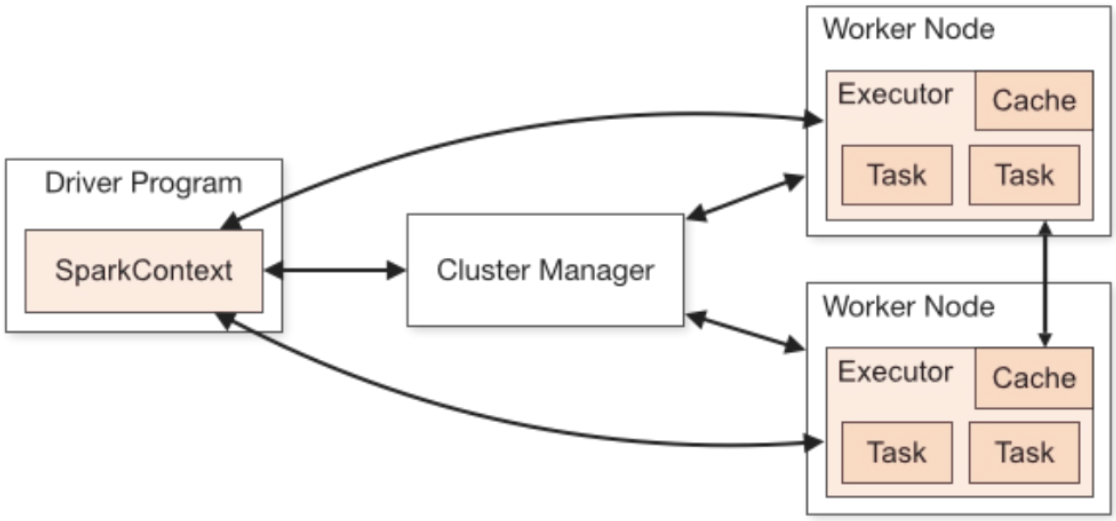
Mejor rendimiento de la API estructurada

- Spark con DataFrames y DataSets se aprovecha del uso de datos con estructura para optimizar el rendimiento utilizando el optimizador de consultas Catalyst y el motor de ejecución Tungsten.



Fuente: Recent performance improvements in Apache Spark: SQL, Python, DataFrames, and More

Conceptos clave



(Fuente: H. Karau, A. Konwinski, P. Wendell, M. Zaharia, "Learning Spark", O'Reilly, 2015)

Driver

- Crea un `SparkContext`
- Convierte el programa de usuario en tareas:
  - `DAG` de operaciones lógico -> plan de ejecución físico
- Planifica las tareas en los ejecutores

SparkSession y SparkContext

- `SparkSession`: punto de entrada de todas las funcionalidades de Spark
  - Permite especificar la configuración de la aplicación Spark
  - En el shell de Spark se crea automáticamente, y en el notebook se puede crear automáticamente, aunque aquí lo creamos a mano (variable `spark`)
- `SparkContext`: realiza la conexión con el cluster `
  - Se crea a partir del `SparkSession`
  - Punto de entrada para la API de bajo nivel
  - En el notebook (o el shell de Spark), se define automáticamente (variable `sc`)
- Creación en un script Python

In [ ]:
```
%pip install pyspark
```

In [ ]:
```
from pyspark.sql import SparkSession
from pyspark import SparkContext
# Creamos un objeto SparkSession (o lo obtenemos si ya está creado)
```

```
spark: SparkSession = SparkSession \
  .builder \
  .appName("Mi aplicacion") \
  .config("spark.alguna.opcion.de.configuracion", "algun-valor") \
  .master("local[*]") \
  .getOrCreate()

sc: SparkContext = spark.sparkContext
```

Executors

- Ejecutan las tareas individuales y devuelven los resultados al Driver
- Proporcionan almacenamiento en memoria para los datos de las tareas

Cluster Manager

- Componente *enchufable* en Spark
- YARN, Mesos o Spark Standalone

Instalación de Spark

- Para **Scala**:
  1. Descargar Apache Spark de http://spark.apache.org/downloads.html
     - La versión "Pre-built for Hadoop 3.x and later" incorpora Hadoop
     - También se puede descargar un versión sin Hadoop para usar una instalación de Hadoop ya disponible
     - Alternativamente, es posible construir Spark desde el código fuente
  2. Extraer el fichero descargado
- Para **Python**, se puede instalar con `pip`:
  1. Instalar los paquetes necesarios (instalad `jupyter` si planeáis utilizarlo):
     - `pip install pyspark jupyter ipython`
  2. Añadir en el `PATH` las carpetas de las instalaciones con `pip`:
     - `export PATH=$PATH:~/.local/bin`
  3. Si se quiere ejecutar Spark en el clúster hay que hacer que pueda encontrar la instalación de Hadoop:
     - `export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop`

Ejecución de Spark

1. Usando consolas interactivas
   - Scala: `spark-shell`
   - Python: `pyspark`
     - Python con IPython: `PYSPARK_DRIVER_PYTHON=ipython pyspark`
     - Python con Jupyter: `PYSPARK_DRIVER_PYTHON=jupyter PYSPARK_DRIVER_PYTHON_OPTS="notebook" pyspark`
   - R: `sparkR`
   - SQL: `spark-sql`
   - Usando Apache Zeppelin
2. Lanzando un script con `spark-submit`

In [ ]:
```
# Ejemplo: muestra la versión de PySpark
print("Versión de PySpark {0}.".format(spark.version))
```

Documentación

La documentación oficial sobre Apache Spark esté en https://spark.apache.org/docs/latest/

La documentación de las APIS para los distintos lenguajes está en:

- Python: https://spark.apache.org/docs/latest/api/python/
- Scala: https://spark.apache.org/docs/latest/api/scala/
- Java: https://spark.apache.org/docs/latest/api/java/