Módulo Profesional: Big Data Aplicado

SPARK

¿QUÉ ES APACHE SPARK?

Es un motor de procesamiento de datos en gran escala, diseñado para realizar operaciones distribuidas sobre grandes volúmenes de datos.

Necesita trabajar en un sistema donde pueda acceder a los datos de forma concurrente y distribuida, lo que significa que los datos deben estar distribuidos a través de múltiples nodos o máquinas dentro de un clúster.

¿QUÉ ES APACHE SPARK?

- **Apache Spark** es open-source y es un framework de computación. Requiere:
  - **Cluster manager**
    - Standalone – un gestor de clúster sencillo incluido en Spark
    - Apache Mesos – un gestor de clúster general
    - Hadoop YARN – gestor de recursos incluido Hadoop 2.
  - **Sistema de archivos distribuido**
    - Hadoop Distributed File System (HDFS) – normalmente
    - Cassandra
    - Amazon S3, Google GCS etc…

Soporta un modo pseudo-distribuido para desarrollo y test.
Sistema local de ficheros y un worker por cada CPU core.

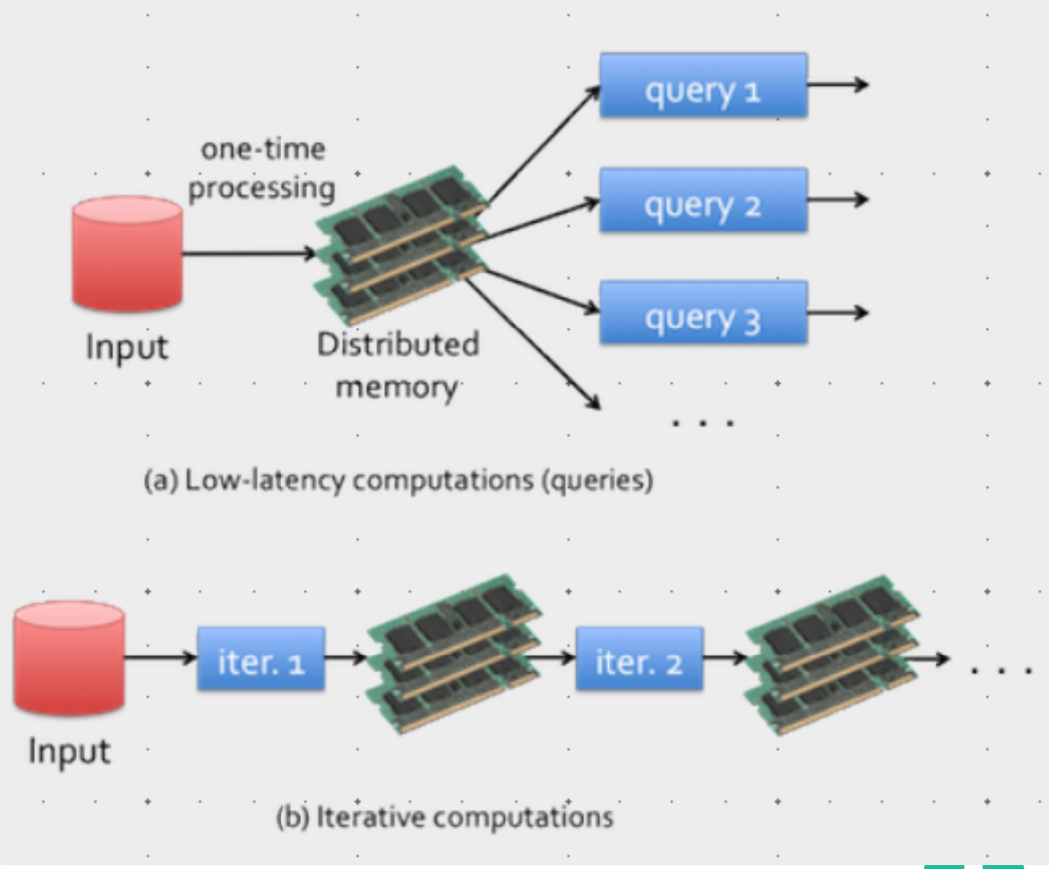
¿QUÉ ES APACHE SPARK?

- Spark proporciona los siguientes beneficios principales:
  - **Computacion muy rápida**
    - Los datos se cargan en memoria distribuida (RAM) sobre servidores en clúster.
    - No necesita persistir pasos intermedios a disco.
  - **Muy accesible**
    - APIs de ~~Java~~, **Scala, Python,** R o SQL
  - **Compatibilidad**
    - Con todo los sistemas de Hadoop existentes
  - **Práctico**
    - Shells Interactivas en Scala y Python (REPL)
  - **Mayor productividad**
    - Debido a estructuras de alto nivel que facilitan centrarse en los cálculos.

¿QUÉ ES APACHE SPARK?

- Apache Spark es una plataforma de computación cluster diseñada para **ser rápida, altamente accesible y de uso general.**

- **Muy rápido**



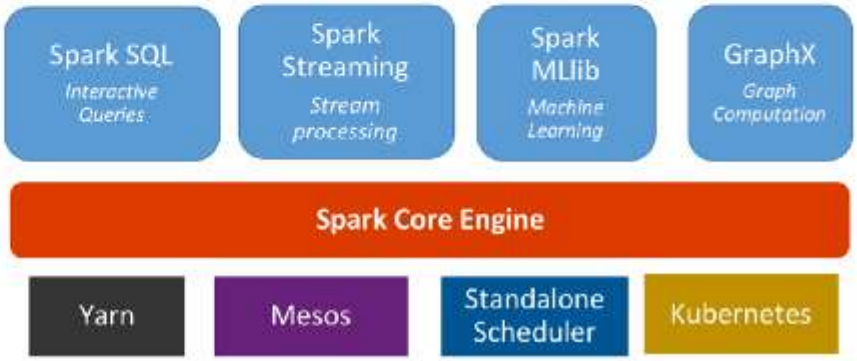
ARQUITECTURA

Spark

Unified, open source, parallel, data processing framework for Big Data Analytics

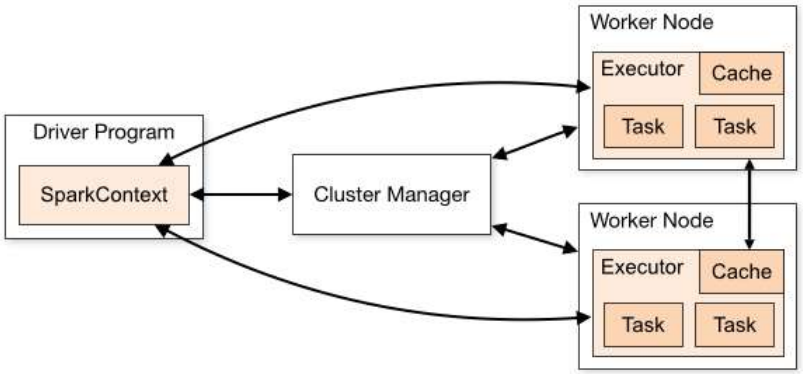
Spark Unifies:

- Batch Processing
- Real-time processing
- Stream Analytics
- Machine Learning
- Interactive SQL

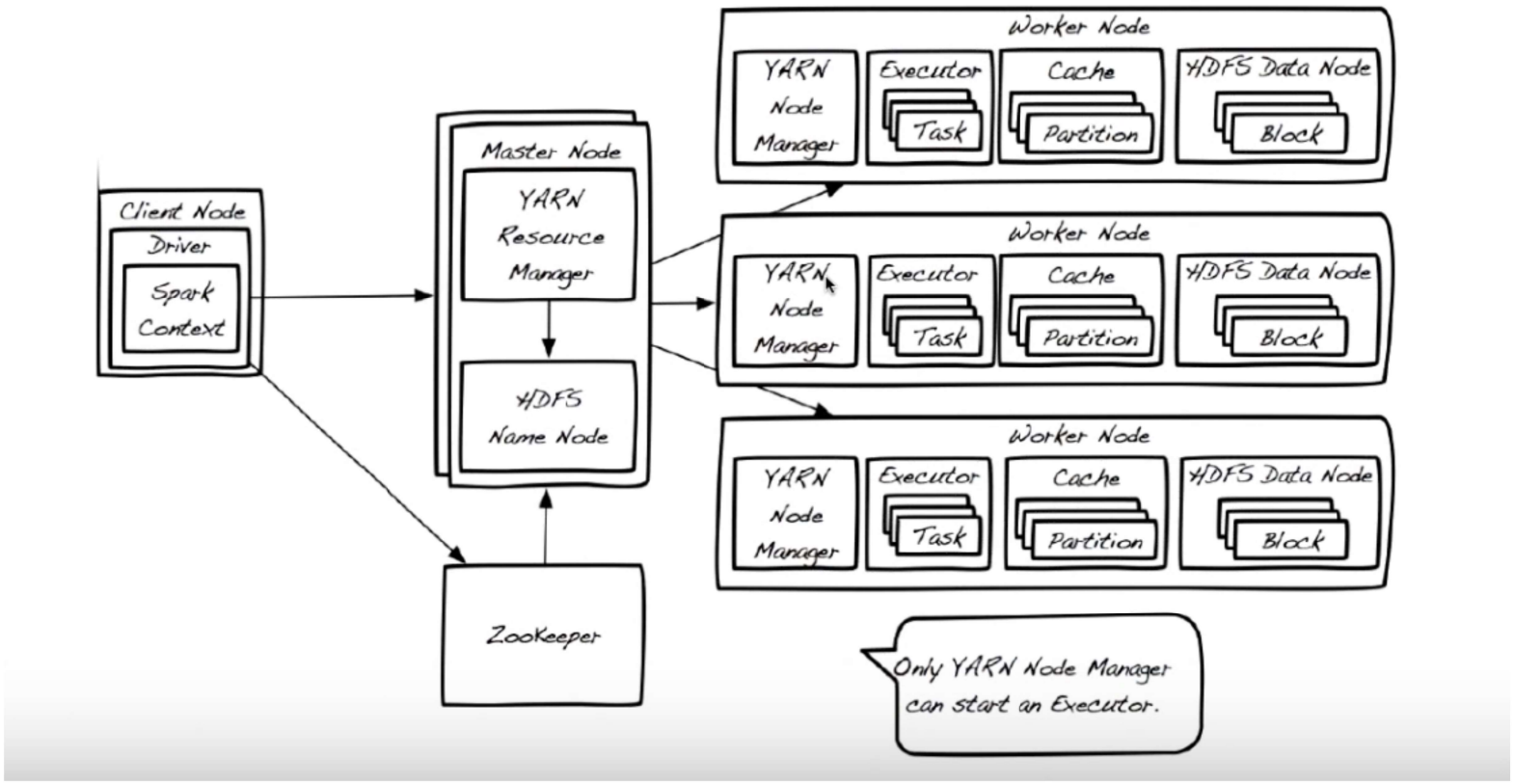


ARQUITECTURA: SPARK RUNTIME

- Arquitectura Maestro/Esclavo
- Coordinador central **driver** (own java process)
- Demonios en workers called **executor** (own java process)
- **Driver + executors = Spark application**
- La aplicación se lanza utilizando un gestor de cluster



Spark YARN-based architecture



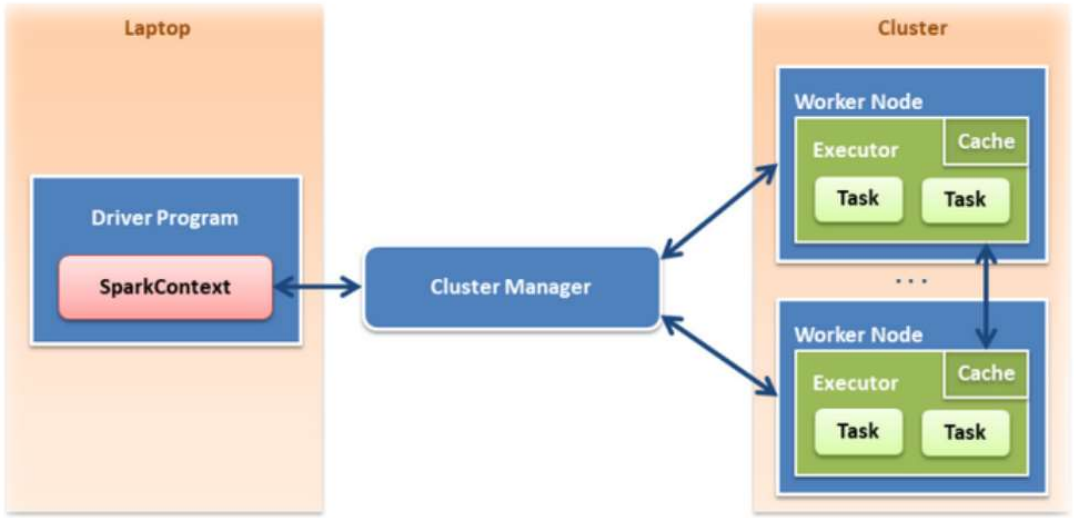
Conceptos Core Spark

- Toda aplicación Spark application debe tener un ***driver program*** que define los datos distribuidos datasets en un clúster y luego lanza las operaciones en paralelo.

- Driver puede ser tu propio programa o **Spark shell** con el tipo de operaciones que tú quieres ejecutar.

- Driver accede a Spark a través de **`SparkSession` object** que representa la conexión al clúster.

Apache Spark - DataFlow in client mode



Apache Spark - DataFlow

- El código de Sparks se ejecuta en el Driver program, conectándose a través de *SparkSession*

- Nosotros definiremos todas fuentes y configuración con SparkSession, donde RDDs y otras estructuras que se crearán también.

```
import findspark
findspark.init('/opt/mapr/spark/spark-2.0.1/')
import pyspark
from pyspark.sql import SparkSession

spark=SparkSession.builder.appName("variable_selection")\
                          .config("spark.master","yarn")\
                          .config("spark.eventLog.enabled","true")\
                          .config("spark.executor.instances","5")\
                          .config("spark.executor.cores","3")\
                          .config("spark.executor.memory","5g")\
                          .getOrCreate()


df=spark.read.csv('hdfs:/data/databases/cars.csv')

df.show()
```

RDD – Resilient Distributed Dataset

- Es una **colección de datos inmutable y distribuida,** el cual está **particionado** en cluster.
- Facilita dos tipos de operaciones:
  - **Transformación**
    - Operaciones como filter(), map(), o union() en RDD que dan como resultado otro RDD.
    - **Evaluación "Lazy". Quiere decir que la evaluación no se ejecutan hasta que se realizan las acciones.**
  - **Acciones**
    - Una acción es una operación como count(), first(), take(n), o collect() que lanzan los **cálculos**, devuelve un **valor** al Master o **escribe** en sistema de almacenamiento.

- Driver recuerda las transformaciones aplicadas al RDD (porque has construido un plan para ello),además si una partición se ha perdido, se puede volver a reconstruir en otra máquina del clúster otra vez.
  - Esto por lo que se llama "**Resilient**"

RDD

- RDDs se pueden crear de tres formas:

1. Cargando de un dataset externo en HDFS, S3, GCS, fichero local, etc…

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

2. **Paralelizar** una colección de objetos existente en memoria del driver/workes

```
scala> val input = sc.parallelize(List(1, 2, 3, 4))
```

3. Como resultado de una tranformación a otro RDD

```
scala> val rdd2 = rdd1.filter(line => line.contains("error"))
```

Databricks

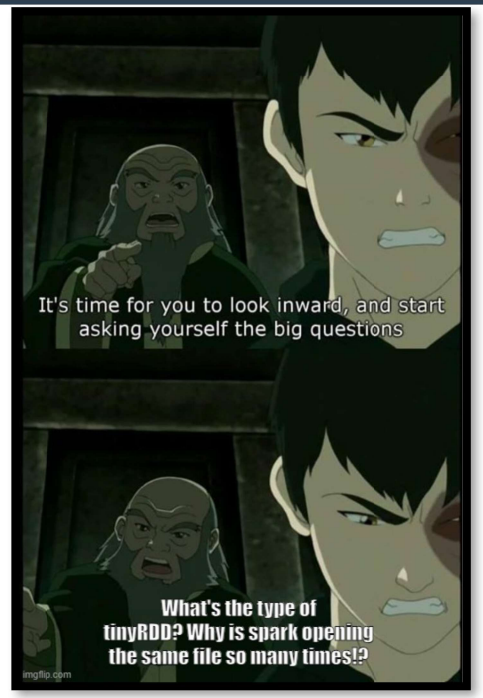
Se basa en apache Spark que nos permite realizar un procesamiento distribuido de datos, siendo una soluciones que permiten trabajar buscando la optimización del procesamiento y análisis de manera escalable y eficiente.

RDD – Introducción Apache Spark

- Carga el notebook "**0 – Introduction Apache Spark**" en Databricks
  - *Ejecuta y analiza las siguientes secciones*
    - **1 - Basic RDD reads**
    - **2 - Parallelize RDDs & basic actions**

    - **Abrir https://www.databricks.com/try-databricks#account**
      - **Cuenta de correo**

        https://www.databricks.com/



https://login.databricks.com/?dbx_source=www&itm=main-cta-login&tuuid=6aa42787-859a-4b6d-ba23-f62309435aa3