

2

T5 : PySpark

Spark me sirve para programar distinto? Claro!!

¿Qué es Spark?

Motor de procesamiento distribuido implementado en Scala.

se ejec. fundamentalmente en memoria.

se construye sobre Hadoop: (es más rápido q hadoop)

1) Orientado a **procesam.** en **HDFS** → DOP < ejecuta

2) Lazy Evaluation: op. de transformación solo se realizan cuando se realizan acciones (map, filter, ...)

3) Permite trabajar en **Datos** real y programar en Batch y forma interactiva

4) Múltiples lenguajes de programación: R, Java, Python, Scala

Principales componentes

Core → Núcleo de procesamiento de spark en el q se ejec. las siguientes librerías

SparkSQL → Permite trabajar sobre los datos con una interfaz SQL en programas basados en SPARK

Streaming → Permite procesamiento en **Datos** real de datos

Mlib → Permite Aprendizaje Automático

GraphX → Visualización & manejo de grafos en spark

Cada una de estas librerías están encapsuladas

↳ Puedo generar código de sparkmlib

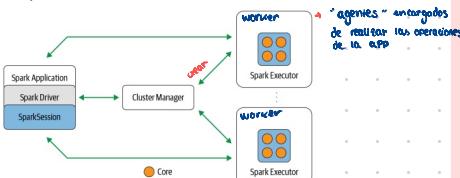
↳ seré producido

Un DAG + ejecutado + Core de spark

Modelos de Despliegue

Mode	Spark driver	Spark executor	Cluster manager
Local	Runs on a single JVM, like a laptop or single node	Runs on the same JVM as the driver	Runs on the same host
Standalone	Can run on any node in the cluster	Each node in the cluster will launch its own executor JVM	Can be allocated arbitrarily to any host in the cluster
YARN (client)	Runs on a client, not part of the cluster	YARN's Resource Manager works with YARN's Application Master to allocate the containers on NodeManagers for executors	YARN's Resource Manager works with YARN's Application Master to allocate the containers on NodeManagers for executors
YARN (cluster)	Runs with the YARN Application Master	Same as YARN client mode	Same as YARN client mode
Kubernetes	Runs in a Kubernetes pod	Each worker runs within its own pod	Kubernetes Master

Arquitectura

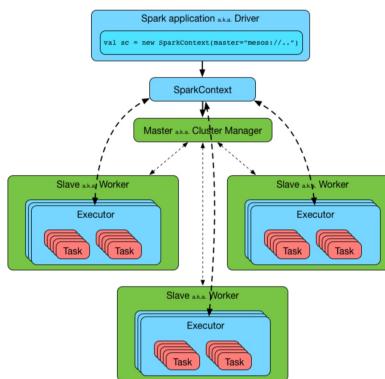


Spark Driver → Inicializar el contexto & enviarlo (spark session)

cluster 'scheduler' por ejemplo: spark → Gestión recursos
encargado de realizar el DAG + Monitorear & Gestionar las ejec. tareas

Spark Session → definir DF, DataFrames, Lectura & ESCRITURA, ...
(metida terminal)

Los nodos (Workers) → interactúan con el driver < realizar ejec. tareas



PySPark

PySpark RDD → Resilient Distributed Dataset

Abstracción fundamental de datos

• Colección < immutable de objetos q pueden ser operados en //

• se le pueden aplicar operaciones (métodos) de transf. en otro RDD
→ invocar acciones para recuperar resultados

• Particionado: se distrib. los datos por todos los nodos del cluster n° x defecto → num. cores distintos.

• Inmutables: una vez se crean no se pueden cambiar

• Resilientes: si un nodo muere, los datos se redistribuirán

• NO tiene soporte para la grec. optimizada → no optimiza la ejecución de acciones o realizar para obtener un resultado + permite hacerlo fuerte

↳ Python: débil
↳ Java: fuerte

• Procesam. ocurre de forma distribib. ? nodos → datos se guardan filas de cada nodo (spark!!)

2. Operaciones en Spark

TRANSFORMACIONES

• operaciones q se aplican a un RDD

→ para crear otro RDD

• genera un nuevo RDD q **DEPENDE** el resultado de la transformación

MAP → una fun a cada elem

FlatMap → = map pero después q el elem
por cada elem de entrada

ACCIONES

• operaciones q se aplican a un RDD

→ Recopilar datos, contar elem, ...

Group By

ACCION → Transf 1 → Transf 2 → Transf 3 → ... → genera un Grafo Acíclico Generado (DAG)

Particiones

Δ Rendimiento → los datos en una estructura de ficheros Hadoop

Reduce By Key

Permite meter los valores de un RDD en forma clave:valor.

Estructura valores q le llegan → si encuentras alguna clave:valor → tu encuentras el otro valor.

Agregaciones

groupby y agg

3 formas ↪ **map**, de ventana, → cálculos basados en un conjunto de filas
SQL

T5 : PySpark

Ejemplo Agregaciones:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import avg
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number

# Crear una SparkSession
spark = SparkSession.builder.appName("Ejemplo").getOrCreate()

# Crear un DataFrame de ejemplo
data = [("Juan", 100), ("María", 150), ("Juan", 200), ("María", 250)]
df = spark.createDataFrame(data, ["Nombre", "Monto"])

# 1. groupby() y funciones de agregación
resultado_groupby = df.groupby("Nombre").agg(avg("Monto"))

# 2. Funciones de ventana
windowSpec = Window.partitionBy("Nombre")           ----- solo se crean 2 particiones por Nombre
resultado_window = df.withColumn("fila", row_number().over(windowSpec))
----- crea una columna llamada "fila"

# 3. Funciones de agregación SQL
df.createOrReplaceTempView("datos")
resultado_sql = spark.sql("SELECT Nombre, AVG(Monto) AS Promedio FROM datos GROUP BY Nombre")

# Mostrar resultados
print("Resultado usando groupby() y funciones de agregación:")
resultado_groupby.show()

print("Resultado usando funciones de ventana:")
resultado_window.show()

print("Resultado usando funciones de agregación SQL:")
resultado_sql.show()

# Detener la SparkSession
spark.stop()
```

• ¿Porq hago esto? → leyendo de la sección

spark.read().format("BigQuery")
spark.read("BigQuery")

20

se aguanta todos los contenidos



UDF: User Defined Functions

Funciones realizar sobre DF (A) de spark



- UDF Facilitan abstracción de lógica de negocio
↳ se ejec a nivel FILA
 - se aplican a columnas pero se ejec a nivel FILA

- $\nabla \otimes \text{ejec.}$
 - $H_{\text{ef}} \neq H_{\text{pos}}$ según como estén orientados (m bocanadas)

- **UDF** → **@UDF**
- **Pandas-UDF** → **@Pandas-UDF** (return Type = " . ")
- **Grouped Map** → **@Pandas-UDF** (schema, PandasUDFType 。 GROUPED-D-MAP)
- **Grouped Agg** → **@Pandas-UDF** (schema, PandasUDFType 。 GROUPED-D-AGG)

VNF → ejec a nivel de filo

Pandas - UDF → For detrows use pandas + Arrow

distribuis dates relumine.

GROUPED - MAP

```

schema = StructType([
    StructField("Casa", StringType(), True),
    StructField("median_income", DoubleType(), True),
    StructField("porc_ventas", DoubleType(), True)])
)

pandas_udf(schema, PandasUDFType.GROUPED_MAP)

def calcular_porcentaje(ingresos):
    ingresos['porc_ventas'] = ingresos['median_income'] / sum(ingresos['median_income'])
    return ingresos[['Casa', 'median_income', 'porc_ventas']]

# Aplicar la UDF agrupando por categoria
resultado = df_spark.select("Casa", "median_income").groupby("Casa").apply(calcular_porcentaje)
resultado.show()

```

• ¿Qué nos proporciona

RENDIMIENTO

↓
como sabes q es
Group by
↓
Ej: "Apliq" correctamente

GROUP - AGG

```
@pandas_udf(FloatType(), PandasUDFType.GROUPED_AGG)
def agregar(ingreso):
    ingreso_totales = ingreso.sum()
    porcentaje = (ingreso.mean() / ingreso_totales) * 100
    return porcentaje.mean()

# Aplicar la UDF agrupando por categoria
resultado = df_pais.select(['Casa', 'median_income']).groupby('Casa').apply(agregar)
```

Lo mismo q Group by pero
devuelve valores