Spark y Arquitecturas de Datos

Data Science (CC3084) - 2026

Spark y Arquitecturas de Datos

Semestre 02, 2026

De Dónde Venimos

La lección anterior dejó un problema abierto.

- Hadoop reparte los datos en bloques replicados sobre HDFS y los recursos del clúster mediante YARN.
- MapReduce los procesa en tres fases, mapa, mezcla y clasificación, y reducción.
- Entre cada etapa escribe el resultado intermedio a disco.

Para un proceso de un solo paso eso es aceptable. Para un algoritmo iterativo que recorre los mismos datos cincuenta veces, leer y escribir disco domina el tiempo total. Esta lección empieza justo ahí.

Apache Spark

Spark es una plataforma de procesamiento de datos a gran escala que permite trabajar con grandes volúmenes distribuidos en muchos servidores.

Por qué Spark

- Velocidad: fue construido desde su núcleo enfocado en rendimiento y puede llegar a ser cien veces más rápido que MapReduce en procesamiento a gran escala, porque mantiene los datos en memoria en lugar de escribir todo a disco.
- Facilidad de uso: ofrece interfaces con código sencillo y limpio, y más de cien operadores para transformar datos y manejar información semiestructurada.
- Motor unificado: incluye librerías de alto nivel para consultas SQL, datos en streaming, aprendizaje automático y procesamiento de grafos, combinables sin complejidad adicional.

Características

- Es más rápido que MapReduce porque usa procesamiento en memoria.
- Soporta Python, Scala, Java, SQL y R.
- Es perezoso, en el sentido de que no ejecuta nada hasta que se pide un resultado concreto.
- Funciona en una computadora personal o en clústeres de miles de nodos.
- Trabaja con múltiples sistemas de almacenamiento, como HDFS, bases de datos o la nube.
- Está diseñado para ser rápido, escalable y tolerante a fallos.

Evaluación perezosa

Spark distingue transformaciones de acciones. Una transformación describe un cambio pero no lo ejecuta; una acción pide un resultado y dispara todo el cómputo acumulado.

```
df.filter(...)      transformación   ->  no ejecuta nada
df.select(...)      transformación   ->  no ejecuta nada
df.count()          acción           ->  ahora sí ejecuta todo
```

Al esperar hasta la acción, Spark ve el plan completo, lo representa como un grafo dirigido acíclico y lo optimiza antes de mover un solo dato.

Spark y Hadoop

Spark no reemplaza a Hadoop, reemplaza a MapReduce. Es común usar HDFS como almacenamiento barato y confiable y Spark como motor de procesamiento rápido sobre esos mismos datos.

| Aspecto | MapReduce | Spark | |---|---|---| | Resultados intermedios | Disco | Memoria | | Modelo | Dos etapas fijas | Grafo dirigido acíclico | | Velocidad iterativa | Lenta | Hasta cien veces más rápida | | Streaming | No incluido | Incluido | | Lenguajes | Java principalmente | Python, Scala, Java, SQL, R |

Arquitectura de Spark

Spark usa una arquitectura maestro-esclavo. Consiste en un controlador, que funciona como nodo maestro, y varios ejecutores que corren como nodos de trabajo en el clúster. Sirve tanto para procesamiento por lotes como en tiempo real.

Componentes de la arquitectura

| Componente | Rol | |---|---| | Driver | Coordina la ejecución, corre la función principal y crea el SparkContext | | SparkContext |

Punto de entrada a Spark, representa la conexión al clúster y coordina tareas | | Cluster Manager | Asigna recursos y administra el clúster, sea YARN, Mesos o el propio de Spark | | Executor | Proceso en un nodo de trabajo que ejecuta tareas y guarda datos en caché | | Task | Unidad de trabajo más pequeña, un cálculo sobre una sola partición de datos |

Flujo de ejecución

```
Driver  ->  SparkContext  ->  Cluster Manager  ->  Executors  ->  Tasks
                                                        |
                                                resultados de vuelta al Driver
```

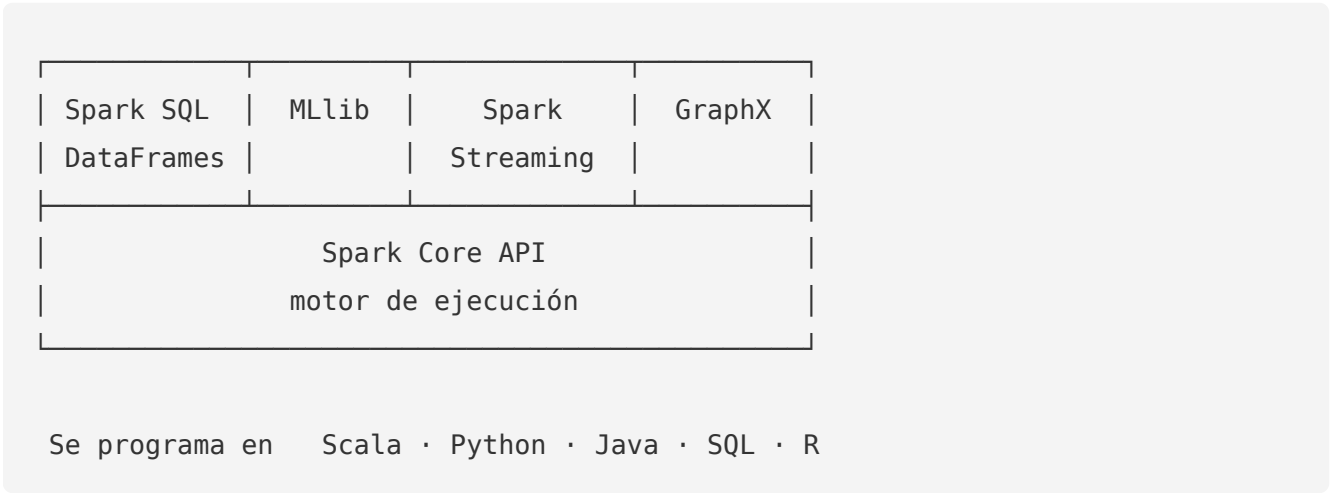
El driver parte el trabajo en tareas y las asigna a los ejecutores. Los ejecutores trabajan de forma simultánea, guardan datos en memoria o disco para reutilizarlos y se comunican con el driver y con el administrador de clústeres.

RDD y DataFrame

- RDD, o Resilient Distributed Dataset, es la abstracción original de Spark. Es una colección distribuida, inmutable y tolerante a fallos gracias a que recuerda cómo fue construida.
- DataFrame es una capa estructurada sobre esa base, con columnas y tipos, que el optimizador de consultas puede analizar y mejorar.
- En la práctica se trabaja con DataFrames, y solo se baja a RDD cuando se necesita control fino.

Componentes de Spark

Spark se organiza como un núcleo con librerías especializadas encima, todas compartiendo el mismo motor.



Spark Core

- Es el motor de ejecución general subyacente sobre el que se construye todo lo demás.
- Proporciona la capacidad de cálculo en memoria.
- Hace referencia a conjuntos de datos almacenados en sistemas externos.
- Contiene la programación, la distribución y la monitorización de trabajos, el despacho de tareas y la recuperación ante fallos.

Spark SQL y DataFrames

- Permite consultas SQL interactivas para exploración.
- Procesa datos estructurados y aplica optimizaciones adicionales sobre ellos.
- Se integra con programas Spark en Java, Scala, Python y R.
- Da acceso uniforme a fuentes como JSON, JDBC, Parquet, Hive y Avro.
- Permite conectarse mediante controladores JDBC u ODBC desde cualquier herramienta de inteligencia de negocios.
- Es compatible con HiveQL, y puede usar tablas, metaalmacenes y funciones definidas por el usuario de Hive.

MLlib

- Es la biblioteca de aprendizaje automático escalable de Spark.
- Ofrece algoritmos de alta calidad y alta velocidad que aprovechan la redundancia del clúster.
- Desde Spark 2.0 se desarrolla sobre Spark SQL, con una interfaz basada en DataFrames.
- Interopera con NumPy y con bibliotecas de R, y admite fuentes de datos de Hadoop.
- Corre de forma independiente o sobre Mesos, YARN y otros administradores, y accede a HBase, Cassandra o Hive.

Spark Streaming

- Procesa flujos de datos en vivo de forma escalable, con alto rendimiento y tolerancia a fallos.
- Se conecta a flujos de entrada y salida como Kafka, Kinesis o sockets TCP.
- Implementa microlotes, es decir, toma los datos en tiempo real de un flujo de entrada, los divide en varios lotes y entrega el resultado también en lotes.
- Se recupera de fallos de nodo sin escribir código adicional.
- Reutiliza el mismo código para unir transmisiones, procesamiento por lotes y consultas ad hoc.

GraphX

- Es el motor de ejecución paralela de grafos y de análisis de grafos de red.
- Modela un multigrafo dirigido con propiedades asociadas a cada vértice y a cada arista.
- Extiende el RDD de Spark, que es inmutable.

- Combina análisis exploratorio y cálculo iterativo de grafos en un solo sistema, viendo los mismos datos como colección o como grafo.
- Proporciona un conjunto amplio de algoritmos sobre grafos.

Almacenamiento Analítico

Hadoop y Spark resuelven cómo procesar los datos. Falta decidir dónde viven y bajo qué reglas. Tres arquitecturas dominan esa conversación.

Data Warehouse

Repositorio central de datos ya limpios, integrados y modelados, pensado para responder consultas analíticas de negocio.

- Esquema al escribir, es decir, los datos se estructuran antes de guardarse.
- Solo admite datos estructurados, en tablas con tipos definidos.
- Consultas rápidas y confiables, con transacciones y control de acceso.
- Almacenamiento costoso, porque el cómputo y el disco suelen venir acoplados.
- Cambiar el modelo es caro, y lo que no entró al esquema se pierde.

Ejemplos de plataforma son Snowflake, BigQuery, Redshift y Teradata.

Data Lake

Repositorio que guarda los datos en su formato crudo, sin imponer una estructura previa, sobre almacenamiento de objetos barato.

- Esquema al leer, es decir, la estructura se decide en el momento del análisis.
- Admite datos estructurados, semiestructurados y no estructurados por igual.
- Almacenamiento barato y escalable, típicamente HDFS, S3 o equivalentes.

- No garantiza transacciones ni consistencia entre escrituras concurrentes.

Guardar primero y decidir después es la ventaja del lake, y también su trampa.

El pantano de datos

Un lago sin catálogo, sin documentación y sin control de calidad se convierte en un depósito donde nadie encuentra nada ni confía en lo que encuentra. En la industria se le llama data swamp.

Guardar todo es barato, pero la gobernanza es el costo real. Sin metadatos ni linaje, el volumen acumulado deja de ser un activo y pasa a ser una carga.

Data Lakehouse

Arquitectura que pone las garantías de un warehouse sobre el almacenamiento barato de un lake, para no mantener dos sistemas separados con copias duplicadas de los mismos datos.

| | Warehouse | Lake | Lakehouse | |---|---|---|---| | Esquema | Al escribir | Al leer | Al leer, con validación | | Datos | Estructurados | Todos | Todos | | Costo | Alto | Bajo | Bajo | | Transacciones | Sí | No | Sí | | Casos de uso | Reportes | Exploración | Ambos |

Formatos de tabla abiertos

La pieza técnica que hace posible el lakehouse. Delta Lake, Apache Iceberg y Apache Hudi son las tres opciones principales.

- Agregan transacciones sobre archivos guardados en almacenamiento de objetos.
- Permiten consultar el estado de una tabla en un momento anterior.

- Validan el esquema al escribir, para que un cambio no rompa las consultas existentes.
- Registran metadatos y versiones junto a los datos, sin depender de un motor específico.

El término lakehouse lo acuñó Databricks en 2020. Describe un cambio real de arquitectura, pero conviene recordar que nace de un proveedor y no de un consenso neutral.

Databricks

Databricks es una plataforma de inteligencia de datos que integra en un solo lugar las herramientas del ciclo de vida del dato, con Spark como motor de fondo.

Módulos

- ETL, para extraer, transformar y cargar datos.
- Warehousing, para almacenamiento analítico consultable.
- Data Sharing, para compartir datos entre equipos y organizaciones.
- Orchestration, para programar y encadenar trabajos.
- Governance, para permisos, linaje y auditoría.
- Inteligencia artificial, para entrenar y servir modelos.

Por qué usarla en el curso

Levantar un clúster de Hadoop o Spark desde cero exige administración de sistemas que no es el objeto de esta clase. Databricks entrega el clúster ya configurado y deja el foco en el análisis.

Preparación del ambiente

1. Ingresar a https://login.databricks.com/
2. Registrarse y crear una instancia gratuita.
3. Crear un notebook y asociarlo al clúster que la plataforma provee.
4. Verificar la instalación ejecutando una consulta sencilla sobre un conjunto de datos de ejemplo.

Resumen

- Spark reemplaza a MapReduce manteniendo los datos en memoria, y llega a ser cien veces más rápido en cargas iterativas.
- Es perezoso, así que ve el plan completo antes de ejecutarlo y lo optimiza como grafo dirigido acíclico.
- Su arquitectura es maestro-esclavo, con un driver que coordina y ejecutores que trabajan en paralelo.
- Sobre Spark Core se apoyan Spark SQL, MLlib, Spark Streaming y GraphX, todos con el mismo motor.
- El warehouse exige esquema al escribir y solo acepta datos estructurados.
- El lake guarda todo crudo y barato, pero sin gobernanza se convierte en un pantano de datos.
- El lakehouse busca las garantías del warehouse sobre el costo del lake, apoyado en formatos de tabla abiertos.
- Los formatos de tabla abiertos, como Delta Lake e Iceberg, son lo que da transacciones al lakehouse.
- Databricks integra el ciclo de vida completo del dato sobre Spark, sin administrar la infraestructura.

Bibliografía

- Ecosistema de Spark. https://www.cloudduggu.com/spark/ecosystem/
- RDD Programming Guide. https://spark.apache.org/docs/latest/rdd-programming-guide.html
- Delta Lake. https://delta.io/
- Apache Iceberg. https://iceberg.apache.org/