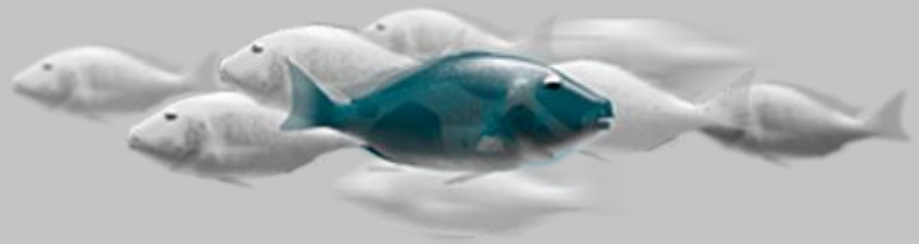
EDEM



Máster en Big Data & Cloud | VII Edición

Pyspark

Ignacio Reyes Vázquez

1. Catalyst & CBO

Catalyst

El optimizador Catalyst (cerebro de spark) toma una consulta y decide la mejor manera de ejecutarla. Para ello, pasa por cuatro fases:

- Fase 1: Análisis

  Spark SQL interpreta la consulta y construye un árbol que representa su estructura. En esta etapa comprueba que las tablas, columnas y tipos de datos sean correctos antes de continuar.

- Fase 2: Optimización lógica

  Catalyst genera diferentes opciones de cómo podría resolverse la consulta. Luego, usando su optimizador basado en costes (CBO), calcula el coste de cada opción y selecciona el plan lógico más eficiente.

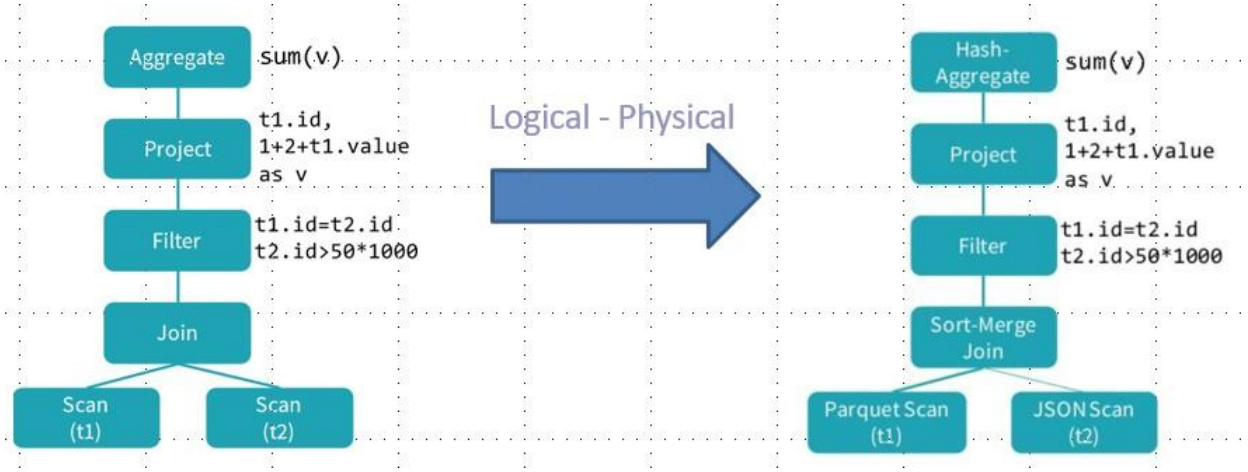
Catalyst

- Fase 3: Planificación física

  Spark SQL toma el plan lógico elegido y lo transforma en un plan físico. En esta etapa determina la mejor estrategia de ejecución posible, seleccionando operadores físicos y métodos de distribución de datos (como shuffle, broadcast o particionamiento) para obtener el máximo rendimiento.

- Fase 4: Generación de código

  Finalmente, Spark SQL convierte el plan físico en código Java bytecode altamente optimizado.



2. Cache and Persist

Cache and Persist

Ambos métodos ayudan a mejorar el rendimiento de DataFrames o tablas usados con frecuencia. La diferencia está en el control sobre el almacenamiento:

.cache() → guarda los datos sólo en memoria.

.persist() → permite definir el nivel de almacenamiento (memoria, disco, etc.).

.unpersist() → elimina el DataFrame del caché.

StorageLevel	Description
MEMORY_ONLY	Data is stored directly as objects and stored only in memory.
MEMORY_ONLY_SER	Data is serialized as compact byte array representation and stored only in memory. To use it, it has to be deserialized at a cost.
MEMORY_AND_DISK	Data is stored directly as objects in memory, but if there's insufficient memory the rest is serialized and stored on disk.
DISK_ONLY	Data is serialized and stored on disk.

Cache and Persist

Cuándo almacenar en caché y conservar

- Si desea acceder repetidamente a un conjunto de datos grande para realizar consultas o transformaciones. Algunos ejemplos son:
    1. DataFrames que se utilizan habitualmente durante el entrenamiento iterativo de aprendizaje automático.
    2. DataFrames a los que se accede habitualmente para realizar transformaciones frecuentes durante ETL o la creación de canalizaciones de datos.

Cuándo no almacenar en caché y conservar

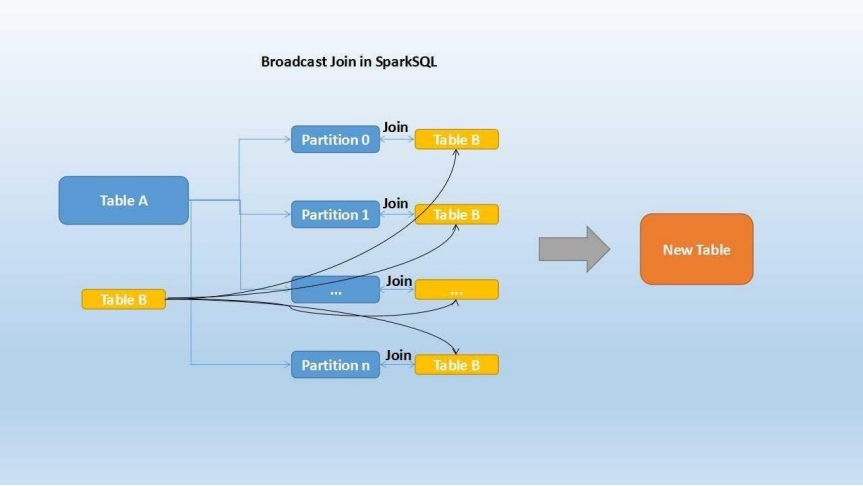
- Cuando los DataFrames son demasiado grandes para caber en la memoria o cuando se trata de una transformación económica en un DataFrame que no requiere un uso frecuente.

2. Different Spark Joins

Broadcast Hash Join

- Se usa cuando se hace un join entre un conjunto de datos grande o mediano y uno pequeño. La tabla pequeña es lo suficientemente ligera para copiarse en todos los ejecutores, evitando costosas reorganizaciones de datos (shuffle).

- El tamaño máximo de la tabla pequeña se controla con spark.sql.autoBroadcastJoinThreshold, que por defecto es 10 MB. Se puede ajustar según la aplicación o desactivar poniéndolo en -1.



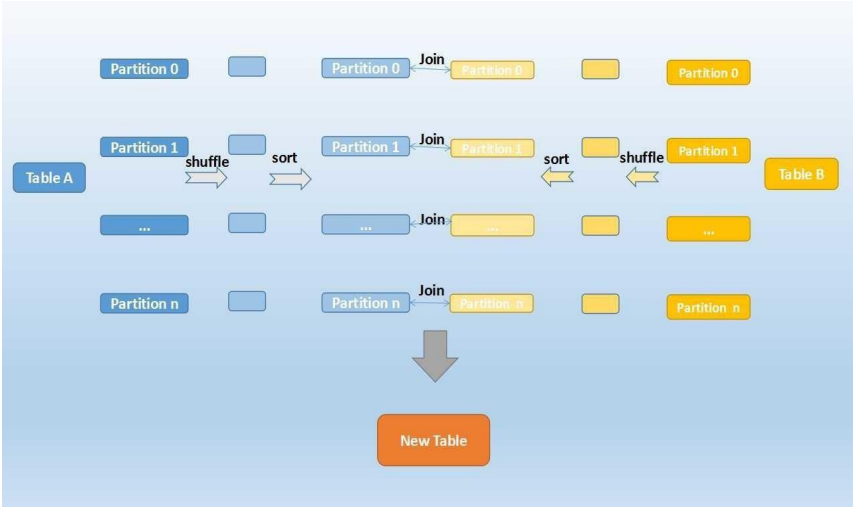

```
df_grande = spark.read.parquet("grande.parquet")
df_pequeno = spark.read.parquet("pequeno.parquet")

# Join normal (puede generar shuffle)
df_join1 = df_grande.join(df_pequeno, "id")

# Broadcast join (mejora rendimiento si df_pequeno es pequeño)
df_join2 = df_grande.join(broadcast(df_pequeno), "id")
```

Shuffle Sort Merge Join

- El Shuffle Sort Merge Join se usa cuando se unen dos conjuntos de datos grandes. Es la estrategia de join por defecto en Spark desde la versión 2.3.
- Funciona así: primero reorganiza (shuffle) los datos por las claves de unión, luego los ordena y combina. Para usarlo, las claves de unión deben ser ordenables.
- Se puede desactivar con: spark.sql.join.preferSortMergeJoin = false



PySpark - Ignacio Reyes Vázquez