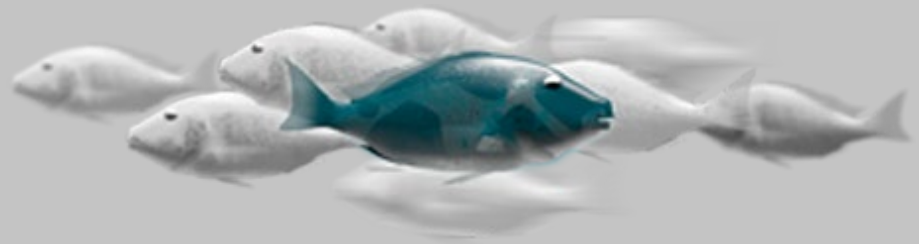
EDEM



Master Data Analytics

Apache Spark Sessions 2023-24
Optimizing Spark Applications

Pablo Pons Roger

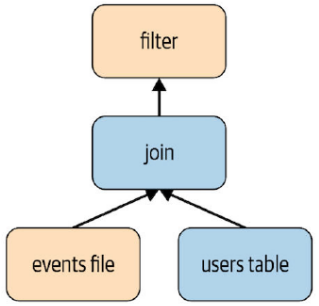
1. Catalyst & CBO

Catalyst

The Catalyst optimizer takes a computational query and converts it into an execution plan. It goes through four transformational phases:

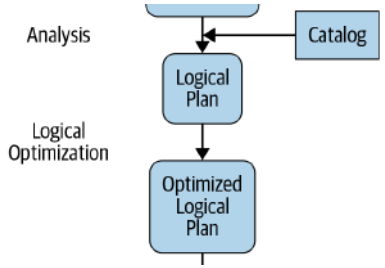
Phase 1: Analysis

The Spark SQL engine begins by generating an abstract syntax tree (AST) for the SQL or DataFrame query



Phase 2: Logical optimization

1. The Catalyst optimizer will first construct a set of multiple plans
2. Using its cost-based optimizer (CBO), assign costs to each plan



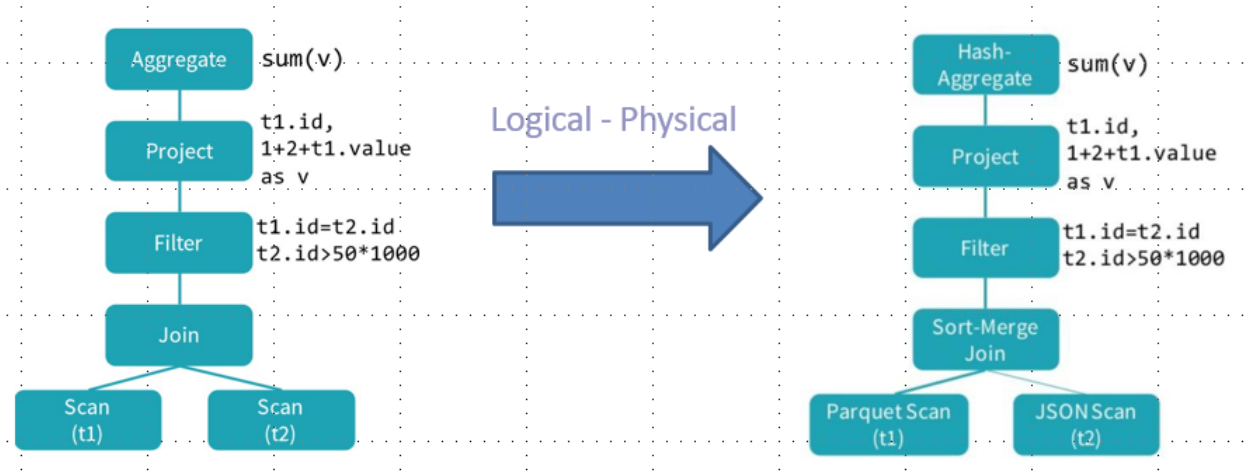
Catalyst

Phase 3: Physical planning

Spark SQL generates an optimal physical plan for the selected logical plan

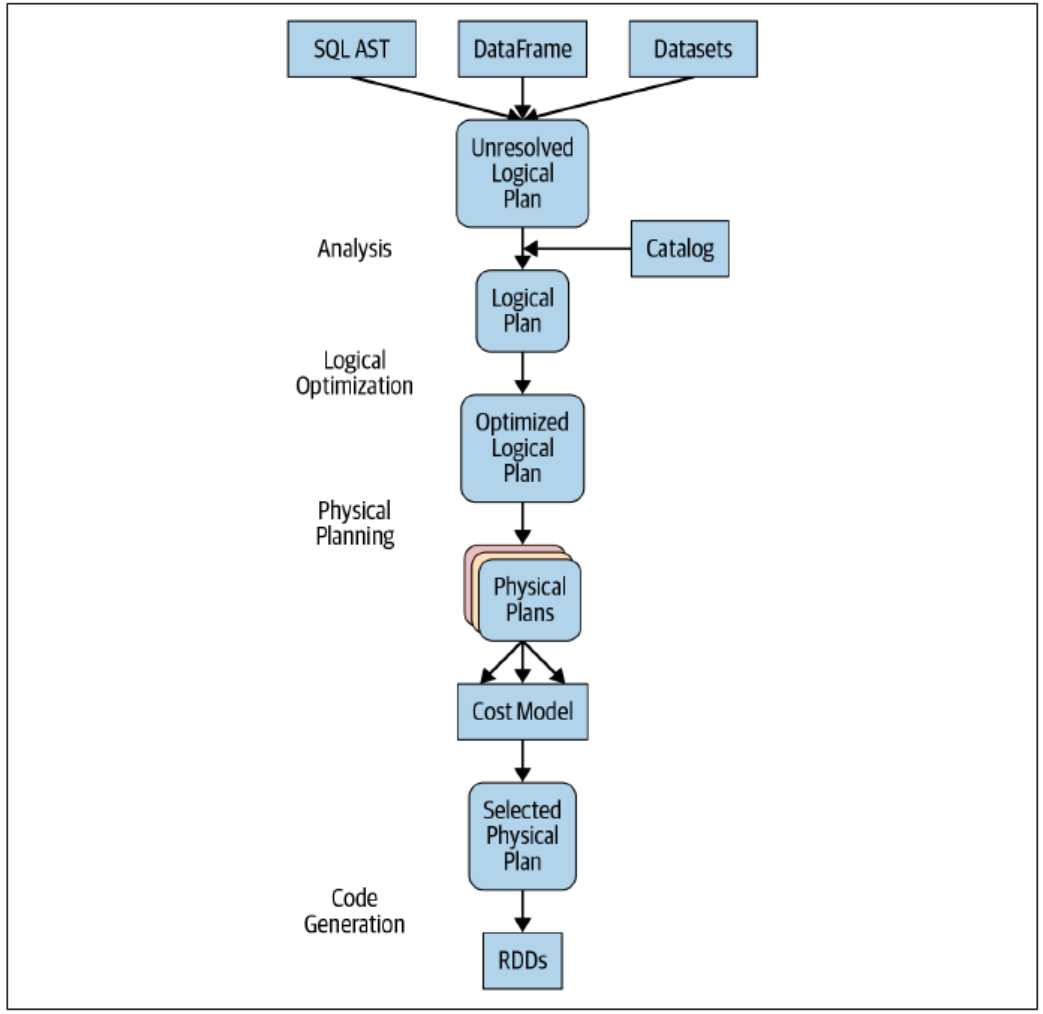
Phase 4: Code generation

The final phase of query optimization involves generating efficient Java bytecode to run on each machine



Catalyst

We can see a high-level overview f the process in the Figure on the right



2. Cache and Persist

Cache and Persist

Both contribute to better performance for frequently accessed DataFrames or tables. The latter provides more control over how and where your data is stored

cache() -> cache in-memory only

persist() -> used to store it to user-defined storage level

unpersist() -> drop Spark DataFrame from Cache

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

When to Cache and Persist

If want to access a large data set repeatedly for queries or transformations. Some examples include:

- DataFrames commonly used during iterative machine learning training
- DataFrames accessed commonly for doing frequent transformations during ETL or building data pipelines

When Not to Cache and Persist

When DataFrames that are too big to fit in memory or an inexpensive transformation on a DataFrame not requiring frequent use

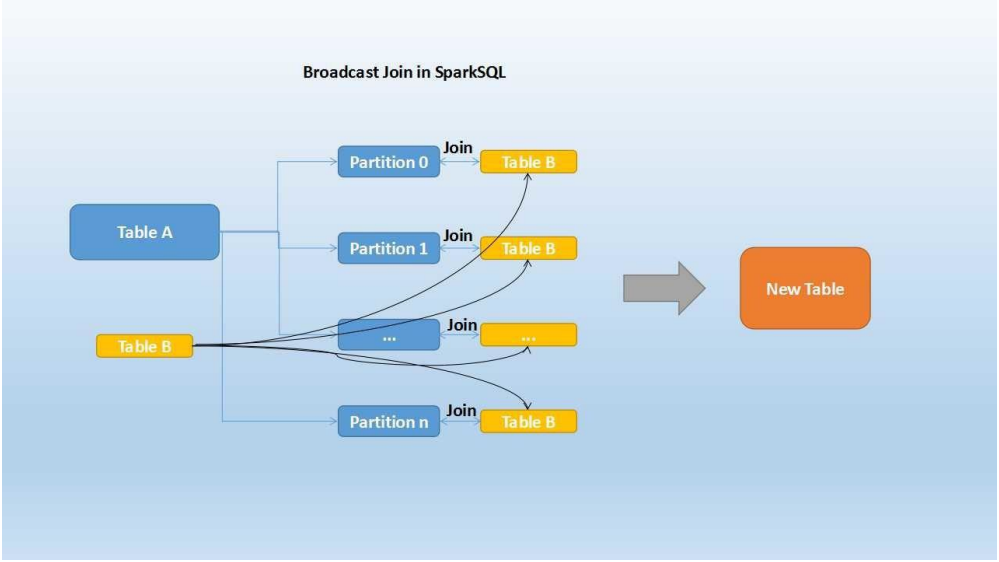
2. Different Spark Joins

Broadcast Hash Join

- Used when we are joining: **large/medium data set – small data set**
- One table is small enough to be replicated for each executor
- The main idea is to avoid the shuffle
- Max size of the small tables that can be broadcasted is defined by this property:

  spark.sql.autoBroadcastJoinThreshold

- Its default value is 10MB, you can adjust it according to the needs of your application including disabling it by giving it a value of -1



Shuffle Sort Merge Join

- Used when we are joining **two large data sets**
- This is the default join strategy in Apache Spark since Spark 2.3
- It can be disabled using spark.sql.join.preferSortMergeJoin=false
- The join keys need to be sortable