# Advanced Spark Optimization Techniques

## Lecture 1

### Contents



### Performance and Query Optimization

Wen you submit a query in Spark, these are the steps before the code is ran:

- Spark creates an Unresolved Logical Plan: it performs syntax checks.
- Then the Unresolved Logical Plan is passed through the Metadata Catalog, checking the existence of the data sources and columns.
- Once the Unresolved Logical Plan is validated, it is converted into a Resolved Logical Plan or Analyzed Logical Plan.
- Then the plan is passed through the Catalyst Catalog, which performs logical optimizations and transformations.
- This results in an Optimized Logical Plan.
- From this Optimized Logical Plan, Spark creates several Physical Plans.
- The Physical Plan is then passed through the Cost Model, which selects the best plan.
- Finally, the selected Physical Plan is converted to RDDs transformations that Spark can execute.

**Catalyst Optimizer** The Catalyst Optimizer is a component of Spark that performs logical optimizations and transformations on the query plan.

It is a pluggable framework that allows you to extend the optimizer with custom rules.

**Adaptative Query Execution** **Adaptive Query Execution** (AQE) in Apache Spark is a feature introduced to optimize query execution plans dynamically at runtime. It aims to improve the performance of Spark SQL queries by adapting the execution plan based on the actual data characteristics and

runtime statistics, rather than relying solely on static optimization decisions made at compile time.

It is thought to solve data skew problems.

By adjusting the query plan based on the Type of Tuning AQE can give us:

- Shuffle Partitions: by default set to 200, AQE can reduce this number if the data is skewed.
- Optimizing Joins: converting sort merge joins to broadcast joins
- Optimizing Skew Joins: converting large partitions into smaller ones, which can help balancing the workload across executors and improve query performance.

**Memory Partitioning** Let us go into detail about the Executor Memory. It is conformed by two components:

- On-Heap Memory: `spark.executor.memory`:
  - Unified Memory: usually around 60% of the total memory. It is formed by:
    - Executor memory: performs Joins, Shuffles, Sorting, GroupBy
    - Storage Memory: stores RDDs, DataFraes, Cached objects and Broadcast Variables; usually around 50% of the total memory
  - User Memory: stores user defined variables and objects
- Off-Heap Memory: aorund 10-20% of the executor memory. Off-heap memory in Spark is used to optimize performance by reducing the burden on the JVM's garbage collector, while the JVM heap is the primary memory area managed by the JVM for Java object allocation. `spark.memory.offHeap.enabled`, `spark.memory.offHeap.size`
- Overhead Memory: 10% of the executor memory. It is used for internal Spark processes and metadata: overhead memory refers to the portion of executor memory that is reserved for internal Spark processes and metadata.. `spark.executor.memoryOverhead`

**Spark UI**

Spark UI can be used to visualize:

- File reading
- Narrow Operations: these are operations that don't involve shuffling data:
  - `filter`
  - `withColumn`

  - `select`

- Wide Operations: these are operations that involve shuffling data:
  - `groupBy`
  - `join`
  - `count`
  - `sum`

**Acknowledgements**

These notes have been taken from the lectures at DataExpert.io