**Adaptive Query Execution (AQE)**

* **Explanation**: AQE dynamically optimizes query execution plans at runtime. It adjusts partitioning, joins, and data skew based on runtime statistics, improving performance.

import org.apache.spark.sql.SparkSession

val spark = SparkSession.builder()

.appName("AQEExample")

.config("spark.sql.adaptive.enabled", "true")

.getOrCreate()

val df = spark.read.option("header", "true").csv("data.csv")

df.groupBy("category").agg("value" -> "sum").show()

AQE automatically adjusts the plan, e.g., coalescing shuffle partitions or switching join strategies, without manual intervention.

**Dynamic Partition Pruning**

* **Explanation**: This feature prunes unnecessary partitions during joins, reducing data scanned and improving query performance, especially in large datasets with partitioned tables.

val sales = spark.read.parquet("sales.parquet")

val products = spark.read.parquet("products.parquet")

sales.join(products, sales("product\_id") === products("id"))

.where(products("category") === "electronics")

.select("sales.\*")

.write.partitionBy("date").save("output")

Dynamic pruning filters out irrelevant partitions of sales based on products conditions at runtime.

**Join Hints and Optimization**

* **Explanation**: Spark 3.0 introduces join hints (e.g., BROADCAST, SHUFFLE\_HASH) to guide the optimizer, allowing fine-tuned control over join strategies for better performance.

val smallDF = spark.read.json("small.json")

val largeDF = spark.read.parquet("large.parquet")

smallDF.hint("broadcast").join(largeDF, "id").show()

The broadcast hint ensures the smaller dataset is broadcasted to all nodes, optimizing the join.

**Catalyst Optimizer Improvements**

* **Explanation**: The Catalyst optimizer in Spark 3.0 has enhanced cost-based optimization and rule-based optimizations, leading to more efficient query plans.

val df = spark.sql("SELECT \* FROM sales WHERE amount > 1000")

df.explain(true) // Shows optimized plan with Catalyst enhancements

Improvements include better predicate pushdown and join reordering.

**Deep Dive into Spark**

1. **Spark Internals and Architecture**
   * **Explanation**: Spark's architecture includes the Driver, Executors, and Cluster Manager. The Driver runs the main program and DAG scheduler, while Executors handle task execution. Data is managed in RDDs and DataFrames.

val rdd = sc.textFile("input.txt")

val mappedRDD = rdd.map(line => line.length)

mappedRDD.collect()

* This shows RDD creation and transformation, reflecting Spark's internal data processing.

**Task Scheduling and Resource Management**

* **Explanation**: Spark uses a DAG scheduler to break queries into stages and a TaskScheduler to assign tasks to executors, optimizing resource use across the cluster.

spark.conf.set("spark.task.cpus", "2")

val df = spark.read.parquet("data.parquet")

df.rdd.getNumPartitions // Check partition distribution

Configuration and partition inspection help tune task scheduling.

**Memory Management**

* **Explanation**: Spark 3.0 improves memory management with the Unified Memory Management model, balancing execution and storage memory to avoid OOM errors.

spark.conf.set("spark.memory.fraction", "0.8")

val cachedDF = spark.read.parquet("data.parquet").cache()

cachedDF.count()

The cache() operation leverages optimized memory allocation.

Cluster Management

**YARN**

* **Explanation**: Yet Another Resource Negotiator (YARN) manages resources in Hadoop clusters, allocating CPU and memory to Spark applications.

val spark = SparkSession.builder()

.appName("YARNExample")

.master("yarn")

.getOrCreate()

spark.sparkContext.setLogLevel("INFO")

submit my spark job to yarn

**Kubernetes**

* **Explanation**: Spark 3.0 supports Kubernetes for containerized deployment, offering scalability and resource isolation.

val spark = SparkSession.builder()

.appName("KubernetesExample")

.master("k8s://https://<k8s-apiserver>")

.getOrCreate()

**Standalone**

* **Explanation**: Spark’s standalone mode is a simple cluster manager included with Spark, suitable for small-scale or testing environments.

val spark = SparkSession.builder()

.appName("StandaloneExample")

.master("spark://<master-host>:7077")

.getOrCreate()