🧠 Adaptive Query Execution (AQE)

Definition:

Adaptive Query Execution (AQE) is a feature in Apache Spark that dynamically optimizes query plans at runtime based on the actual data statistics observed during execution. This helps improve performance, especially for complex or skewed data.

Key Features of AQE:

---

1. Dynamically Coalesce Partitions

What it means:

- After a shuffle stage (e.g., after a join or aggregation), Spark may generate a large number of small partitions.
- Instead of processing each tiny partition (which causes task overhead), AQE can combine multiple small partitions into fewer, larger ones at runtime.

Why it helps:

- Reduces task scheduling overhead.
- Improves parallelism and resource utilization.
- Leads to faster processing of stages with many small shuffle files.

How it works:

- Spark collects statistics about partition sizes after a shuffle stage.
- If many small partitions are detected, Spark will merge them on the fly before launching the next stage.

---

2. Optimizing Join Strategy During Runtime

What it means:

- Spark usually decides the join strategy (e.g., sort-merge join, broadcast join) at query planning time.
- With AQE enabled, Spark can change the join strategy at runtime depending on actual data sizes.

Example:

- Initially, Spark may plan for a Sort-Merge Join.
- But if one of the tables turns out to be small enough during execution, Spark switches to a Broadcast Join for better performance.

Why it helps:

- Reduces shuffle.
- Broadcast Join avoids sorting and partitioning the large table.
- More efficient use of memory and CPU.

---

3. Optimizing Skewed Joins

What it means:

- In some datasets, certain keys are highly skewed (i.e., occur much more frequently than others).
- When Spark joins on such keys, a few tasks get large amounts of data while others get very little — leading to task skew.

What AQE does:

- Detects skewed partitions dynamically.
- Splits the skewed keys into smaller sub-tasks to balance the load across executors.

- Non-skewed data continues to be processed normally.

Why it helps:

- Prevents "long tail" of slow tasks.
- Leads to better parallelism and reduced overall execution time.

Summary of Benefits:

- AQE brings dynamic optimizations that were not possible at compile time.
- Reduces overhead by avoiding static assumptions.
- Enables smarter, runtime-based decisions to improve performance and resource efficiency.

Configuration in Spark:

To enable AQE in Spark, set the following in spark.conf:

```
spark.conf.set("spark.sql.adaptive.enabled", "true")
```

Other relevant parameters:

```
spark.conf.set("spark.sql.adaptive.coalescePartitions.enabled", "true")
spark.conf.set("spark.sql.adaptive.skewJoin.enabled", "true")
spark.conf.set("spark.sql.adaptive.localShuffleReader.enabled", "true")
```

🔍 Detailed Explanation: With vs Without Adaptive Query Execution (AQE) in Spark

When you run a Spark job that involves grouping and aggregation (like groupBy().count()), Spark builds a physical plan to execute the logic. This plan can be **static (without AQE)** or **adaptive (with AQE)** depending on your configuration.

1. Presence of AdaptiveSparkPlan

In both cases, the final physical plan is wrapped in an AdaptiveSparkPlan node. This doesn't necessarily mean AQE is active. It simply means Spark **prepared the plan for possible adaptation**. If AQE is disabled or if the optimizer finds no need to adapt, the plan remains unchanged — it still shows as an AdaptiveSparkPlan, but with isFinalPlan=false.

When AQE is active, it dynamically adjusts the plan based on **runtime statistics**. This can lead to major improvements in performance — particularly when data is skewed, large, or unevenly distributed.

---

2. Shuffle and Exchange Behavior

Without AQE, Spark determines the number of shuffle partitions at the **planning stage**. By default, this is 200 partitions. This value remains fixed, even if your data is too small to benefit from such granularity. This leads to **unnecessary shuffling and resource usage**.

With AQE, Spark initially plans using the default 200 partitions. However, after the first stage of the query runs and collects stats, AQE may **dynamically coalesce** these into fewer shuffle partitions. This reduces overhead and speeds up downstream processing. You can see this optimization reflected as an AQEShuffledRead(n) node in the plan, where n is the optimized number of partitions.

---

3. Coalescing Shuffle Partitions

In a non-AQE setup, coalescing does not happen. Even if only a small amount of data is present, Spark still performs shuffle across 200 partitions, which is inefficient.

When AQE is enabled and coalescing is configured (e.g., via spark.sql.adaptive.coalescePartitions.enabled = true), Spark **monitors the size of shuffle files** and merges partitions to a more optimal number at runtime. This is a key performance benefit of AQE.

---

4. Static vs Dynamic Plan Generation

Without AQE, Spark generates a static execution plan during the compile phase, based on **estimates**, not real data. This can lead to suboptimal join strategies or inefficient resource usage.

With AQE, the plan is **revised during execution**, using **actual data metrics** (like row counts and size of shuffled data). As a result, Spark can pick better join strategies, eliminate unnecessary shuffles, and dynamically prune partitions.

---

5. Execution Strategy and Monitoring

In a static plan, once Spark starts executing, it follows the fixed route regardless of inefficiencies. There's no mechanism to learn from what's happening during execution.

AQE introduces intelligence into the execution phase. Spark tracks runtime behavior and dynamically adjusts execution steps. For instance, it may switch from a sort-merge join to a broadcast join mid-execution if it detects that one side is small enough.

---

6. Optimization Features

Without AQE, Spark does not apply dynamic optimizations — it uses only the Catalyst and physical optimizers during planning.

With AQE enabled, Spark performs:

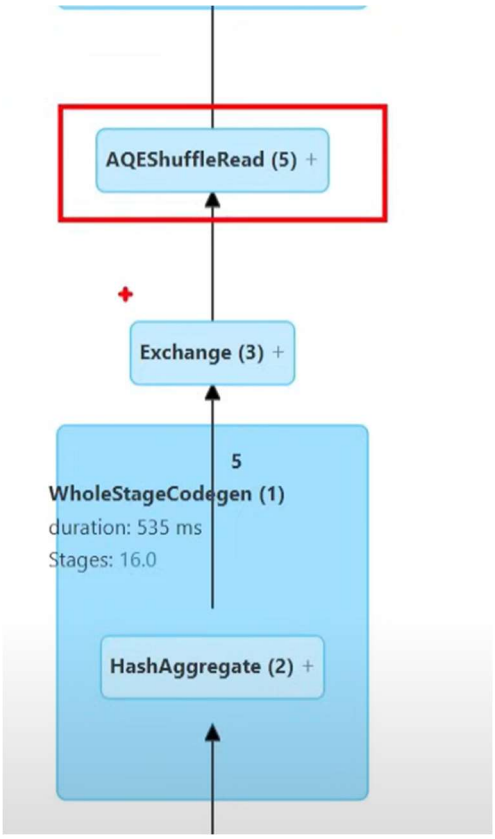
- **Coalescing shuffle partitions** to reduce the number of tasks.
- **Dynamic join strategy selection** (e.g., switching to broadcast).
- **Skewed join handling**, where one partition is disproportionately large.
- **Dynamic partition pruning** (especially helpful with filters on large tables).

Aspect	Without AQE	With AQE (Adaptive Query Execution)
AdaptiveSparkPlan Presence	Even if AQE is disabled, Spark 3.x still wraps the physical plan inside an AdaptiveSparkPlan object. However, this wrapper does not actually trigger any adaptive behavior like plan changes or optimization. It's just a container.	Spark wraps the plan with AdaptiveSparkPlan and actively monitors the execution. Based on runtime statistics (data size, partition skew, etc.), it can change the query plan dynamically to make it more efficient.
Exchange Operator	Spark uses a static exchange/shuffle strategy. It predefines 200 shuffle partitions by default (hashpartitioning(Item_Fat_Content, 200)), regardless of the actual size or skew of the data.	AQE initially plans for 200 partitions too, but after the shuffle stage completes, it inspects actual shuffle sizes and may reduce the number of partitions dynamically,

		eliminating small files and improving performance.
Coalesced Shuffle Partitions	Not applied. The shuffle partitions remain fixed at the plan stage. If most of them are empty or small, Spark still processes them, resulting in inefficient execution.	If AQE determines that many shuffle partitions are small, it dynamically coalesces them into fewer partitions (e.g., 5). This reduces scheduling overhead and improves performance. You'll see AQEShuffledRead(n) in the plan.
Final Plan Generation	The plan is static and determined before execution begins. Spark won't make any changes based on what happens during execution.	The final plan is generated at runtime, meaning Spark can make smarter decisions (like switching join types, reducing partitions, or re-optimizing broadcast behavior) after it sees actual data characteristics.
Shuffle Partitioning	The shuffle is hard-coded to use 200 partitions regardless of actual data size. This can cause over-parallelization and unnecessary overhead.	AQE analyzes the shuffle output sizes. If it finds that many partitions are too small, it can repartition on the fly using fewer, more optimal partitions (e.g., reducing from 200 to 5).
Execution Mode	Spark follows the physical plan as is. There is no runtime feedback loop to modify or adjust execution behavior.	Execution is dynamic and adaptive. Spark makes runtime decisions based on metrics collected during shuffle or stage boundaries to optimize execution on the fly.
Optimization Features	None of the AQE optimizations are active. Spark relies only on static cost-based or rule-based optimization during query planning.	AQE enables several smart optimizations during execution: Coalescing shuffle partitionsDynamically switching join strategies (e.g., sort-merge → broadcast)Handling skew joins by splitting skewed partitions

Summary:

- Without AQE: Spark uses a **static execution plan**, which may be inefficient for unpredictable data patterns, especially in shuffles or joins.
- With AQE: Spark is **smarter and more flexible**, adjusting plans based on **real data sizes and distribution** during execution, improving performance automatically without code changes.


AQEShuffleRead (5) +
Exchange (3) +
5
WholeStageCodegen (1)
duration: 535 ms
Stages: 16.0
HashAggregate (2) +