PySpark – Adaptive Query Execution (AQE)

Introduction to Adaptive Query Execution (AQE)

Adaptive Query Execution (AQE) is an optimization feature introduced in Spark 3.0 to enhance the performance of query execution dynamically. Unlike traditional query execution, where the execution plan is determined at the start, AQE adjusts the execution plan based on runtime statistics and feedback, optimizing it on the fly.

Key Features of AQE:

1. **Dynamic Partition Coalescing**: Reduces the number of shuffle partitions based on actual data size, improving performance.
2. **Dynamic Join Selection**: Changes the join strategy (e.g., switching from sort-merge join to broadcast join) based on the size of the join input.
3. **Dynamic Skew Join Optimization**: Handles data skew by splitting the skewed partitions into smaller ones, improving performance.

How Adaptive Query Execution Works:

When Spark executes a query, AQE collects statistics at runtime, such as the size of shuffle partitions and join inputs. These statistics are used to refine the query plan and adjust strategies, making the execution more efficient.

Enabling AQE in PySpark

AQE is disabled by default but can be enabled by setting the following Spark configurations:

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

Additional AQE configurations:

- **`spark.sql.adaptive.coalescePartitions.enabled`:** Enables dynamic coalescing of shuffle partitions.
- **`spark.sql.adaptive.skewJoin.enabled`:** Enables dynamic skew join handling.
- **`spark.sql.adaptive.join.enabled`:** Enables dynamic join strategy switching.

Example of AQE in Action

1. Dynamic Partition Coalescing

By default, Spark may over-provision shuffle partitions, leading to unnecessary overhead. With AQE, Spark can dynamically adjust the number of partitions based on the size of the data being shuffled.

Scenario: Reducing the number of partitions for a small dataset during a shuffle.

```
from pyspark.sql import SparkSession

# Initialize a Spark session with AQE enabled
spark = SparkSession.builder.appName("AQE Example").getOrCreate()
spark.conf.set("spark.sql.adaptive.enabled", "true")
spark.conf.set("spark.sql.adaptive.coalescePartitions.enabled", "true")

# Sample data
data = [(i, i % 5) for i in range(100)]
df = spark.createDataFrame(data, ["value", "group"])

# Shuffle data by group
shuffled_df = df.groupBy("group").count()
```

```
# Trigger execution and show the result
shuffled_df.show()

# Check the execution plan with AQE optimizations
shuffled_df.explain(True)
```

In this example:

- AQE dynamically adjusts the number of shuffle partitions based on the actual size of each partition. If the data is small, Spark will coalesce the partitions to reduce overhead.

2. Dynamic Join Selection

AQE can switch join strategies during runtime. For example, if Spark detects that one of the tables involved in a join is small enough, it will switch from a sort-merge join to a broadcast join, which is more efficient for smaller tables.

Scenario: AQE switching join strategy from sort-merge join to broadcast join based on data size.

```
# Sample data for two tables
data1 = [(i, "Category" + str(i)) for i in range(10)]
data2 = [(i, "Product" + str(i)) for i in range(3)]  # Small table

df1 = spark.createDataFrame(data1, ["id", "category"])
df2 = spark.createDataFrame(data2, ["id", "product"])

# Join the tables
joined_df = df1.join(df2, "id")

# Show the result of the join
joined_df.show()

# Check the query execution plan with AQE
joined_df.explain(True)
```

In this example:

- AQE dynamically switches to a **broadcast join** instead of using the default **sort-merge join** based on the size of `df2` (small table).
- The query plan will show the broadcast join strategy instead of the sort-merge join.

3. Skew Join Optimization

Data skew can cause some partitions to be disproportionately large, slowing down queries. AQE detects skewed partitions and breaks them down into smaller, more manageable pieces to balance the load.

Scenario: Handling skewed data in a join operation.

```
# Skewed data
data1 = [(1, "A"), (1, "B"), (1, "C"), (2, "D"), (3, "E")]
data2 = [(1, "X"), (2, "Y"), (3, "Z")]

df1 = spark.createDataFrame(data1, ["id", "value"])
df2 = spark.createDataFrame(data2, ["id", "value"])

# Enable skew join handling
spark.conf.set("spark.sql.adaptive.skewJoin.enabled", "true")

# Perform a join, AQE will handle the skew
skewed_join_df = df1.join(df2, "id")

# Show the result
skewed_join_df.show()
```

```
# Check the query execution plan with AQE's skew join optimization
skewed_join_df.explain(True)
```

In this example:

- AQE detects that there are skewed partitions in the `df1` DataFrame (multiple entries with `id = 1`) and automatically adjusts by splitting the skewed partitions into smaller tasks for more balanced processing.

AQE Query Plan Analysis #

When AQE is enabled, you can inspect the execution plan to see how it optimizes the query. For example:

- **DynamicPartitionCoalescing**: Coalesces shuffle partitions based on data size.
- **BroadcastHashJoin**: Switches to a broadcast join if a table is small enough.
- **AdaptiveSparkPlan**: Represents the entire query plan, with optimizations applied dynamically.

```
joined_df.explain(True)
```

Tuning AQE for Performance #

1. **Enable AQE**: Ensure AQE is enabled in your Spark session.
    - `spark.conf.set("spark.sql.adaptive.enabled", "true")`
2. **Tune Partition Coalescing**: Adjust the minimum number of shuffle partitions after coalescing.
    - `spark.conf.set("spark.sql.adaptive.coalescePartitions.minPartitionNum", "2")`
3. **Tune Join Strategy**: Set a threshold for switching to broadcast joins.
    - `spark.conf.set("spark.sql.autoBroadcastJoinThreshold", 10 * 1024 * 1024) # 10 MB`
4. **Enable Skew Join Handling**: Split skewed partitions to prevent performance degradation.
    - `spark.conf.set("spark.sql.adaptive.skewJoin.enabled", "true")`

Performance Benefits of AQE #

- **Reduced Shuffling Overhead**: Dynamic coalescing of shuffle partitions minimizes the overhead of small tasks.
- **Optimized Join Strategies**: Dynamically switching join strategies can drastically reduce query execution time, especially for joins with smaller tables.
- **Handling Data Skew**: AQE automatically adjusts for skewed data, balancing workloads across partitions and improving performance for skewed joins.

Conclusion #

Adaptive Query Execution is a game-changer for query optimization in PySpark. By dynamically adjusting the execution plan based on runtime statistics, AQE significantly improves performance, especially for complex queries involving joins and large shuffles.

Key Takeaways: #

- **Dynamic Partition Coalescing**: Adjusts the number of partitions dynamically during shuffles, reducing overhead.
- **Dynamic Join Strategy**: AQE switches between join strategies (e.g., sort-merge join to broadcast join) based on data size.
- **Skew Join Optimization**: Handles data skew by splitting skewed partitions into smaller ones.
- **Configuration**: AQE is highly configurable and can be tuned for specific workloads, ensuring optimal performance for your queries.