

Optimizing PySpark Joins: How I Reduced Execution Time from 11s to 1.9s Using AQE & Broadcast Join!

📊 Step 1: Understanding the Dataset & Skewness

🔗 Dataset Overview

I worked with two datasets:

1 Orders Dataset (~1M+ rows)

- Contains **highly skewed** customer_id values (some IDs have **500K+ records**).
- Columns: customer_id, order_id, amount.

2 Customers Dataset (~10K rows)

- Smaller in size, **ideal for broadcasting**.
- Columns: customer_id, customer_name.

🔗 Identifying Data Skew

To check for skewness, I ran the following query:



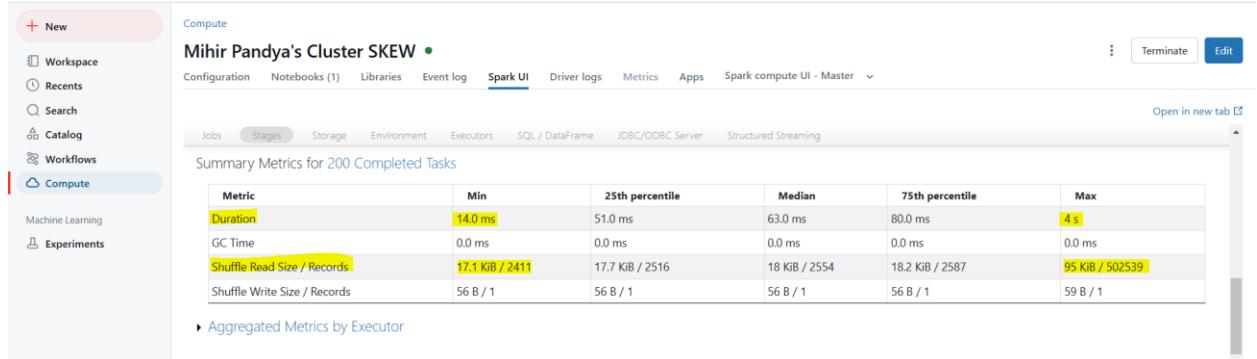
```
skewness Python ▾
File Edit View Run Help Last edit was 6 hours ago
Run all Unknown Share Publish
# Check distribution of orders
df_orders.groupby("customer_id").count().orderBy(col("count").desc()).show(5)

▶ (1) Spark Jobs
▶ df_customers: pyspark.sql.dataframe.DataFrame = [customer_id: string, customer_name: string]
▶ df_orders: pyspark.sql.dataframe.DataFrame = [customer_id: string, order_id: long ... 1 more field]

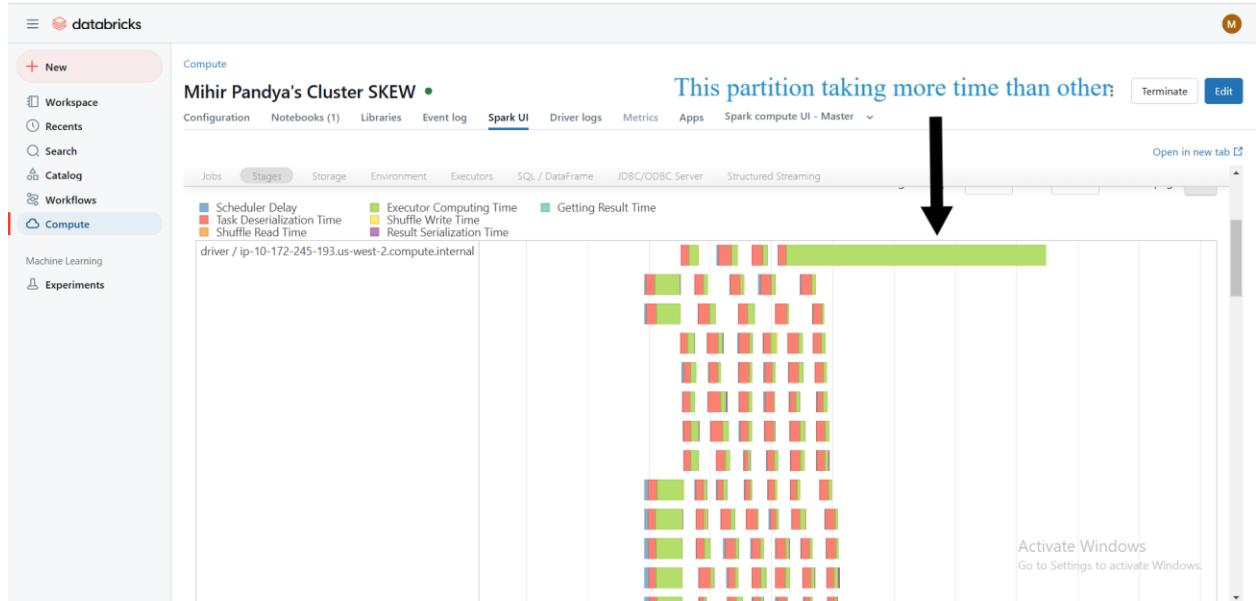
+-----+-----+
|customer_id| count |
+-----+-----+
| CUST_1|499999|
|CUST_500023| 1|
|CUST_500365| 1|
|CUST_500823| 1|
|CUST_500070| 1|
+-----+-----+
only showing top 5 rows
```

⌚ Observations from Spark UI

- Duration of some task was 14ms and while for some was 4s. This shows the skewness is present.
- Most of the task had shuffle read size/Records - 17.1Kib/2411 while a few were having 95KiB/502539. This also shows skewness.



- **Event Timeline Analysis:** The highlighted partition is taking more time than others



🔗 Step 2: Deep Dive into Optimizations

1 Without AQE & No Optimizations (11.26s) – Slowest Approach

- **Join Type:** Sort-Merge Join
- **Issues Identified:**
 - Expensive **shuffle operations** due to hash partitioning.

- Uneven task distribution due to **skewed customer_id**.
- **Some tasks took 4s**, while others took **0.14s**

💡 Logical & Physical Plan:

```

skewness Python
File Edit View Run Help Last edit was 2 minutes ago
Run all Share Publish ▾
New Workspace Recents Search Catalog Workflows Compute Machine Learning Experiments
3
df_result.explain(True)

== Optimized Logical Plan ==
Project [customer_id#2, order_id#3L, amount#4L, customer_name#9]
+- Join Inner, (customer_id#2 = customer_id#8)
  :- Filter isNotNull(customer_id#8)
    +- LogicalRDD [customer_id#2, order_id#3L, amount#4L], false
    +- Filter isNotNull(customer_id#8)
      +- LogicalRDD [customer_id#8, customer_name#9], false

== Physical Plan ==
*(3) Project [customer_id#2, order_id#3L, amount#4L, customer_name#9]
+- *(2) SortMergeJoin [customer_id#2], [customer_id#8], Inner
  :- Sort [customer_id#2 AS NULLS FIRST], false, 0
    +- Exchange hashpartitioning(customer_id#2, 200), ENSURE_REQUIREMENTS, [plan_id=273]
      :  +- *(1) Scan ExistingRDD[customer_id#2,order_id#3L,amount#4L]
  +- Sort [customer_id#8 AS NULLS FIRST], false, 0
    +- Exchange hashpartitioning(customer_id#8, 200), ENSURE_REQUIREMENTS, [plan_id=277]
      :  +- *(2) Filter isNotNull(customer_id#8)
        +- *(2) Scan ExistingRDD[customer_id#8,customer_name#9]

```

2 Without AQE + Broadcast Join (1.94s) – Best Performance

- **Join Type: Broadcast Hash Join**
- **Why It's Faster:**
 - **Avoids shuffle** by broadcasting `df_customers` to all nodes.
 - **Reduces data movement**, improving execution time by **5x**.
 - Spark automatically chooses **Broadcast Hash Join** when explicitly applied.

💡 Code Implementation:

```

Compute Machine Learning Experiments
Just now (2s)
from pyspark.sql.functions import broadcast
start_time = time.time()

df_result_opt = df_orders.join(broadcast(df_customers), "customer_id", "inner")
df_result_opt.count() # Trigger action

end_time = time.time()
print(f"Join Execution Time with Broadcast: {end_time - start_time:.2f} seconds")

(1) Spark Jobs
  Job 25 View (Stages: 2/2)
  df_result_opt: pyspark.sql.dataframe.DataFrame = [customer_id: string, order_id: long ... 2 more fields]
  Join Execution Time with Broadcast: 1.94 seconds
Activate Windows

```

💡 Logical & Physical Plan:

```

skweness Python
File Edit View Run Help Last edit was now
Run all Mihir Pandya's Cluster ...
Share Publish
New Workspace Recents Search Catalog Workflows Compute Machine Learning Experiments
skweness Python
Join Execution Time with Broadcast: 3.17 seconds
df_result_opt.explain(True)
+ ResolvedHint (strategy=broadcast)
+ LogicalRDD [customer_id#8, customer_name#9], false
== Optimized Logical Plan ==
Project [customer_id#2, order_id#3L, amount#4L, customer_name#9]
+ Join Inner, (customer_id#2 = customer_id#8), rightHint(strategy=broadcast)
:- Filter isNotNull(customer_id#2)
: + LogicalRDD [customer_id#2, order_id#3L, amount#4L], false
+ Filter isNotNull(customer_id#8)
+ LogicalRDD [customer_id#8, customer_name#9], false
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+ Project [customer_id#2, order_id#3L, amount#4L, customer_name#9]
+ BroadcastHashJoin [customer_id#2], [customer_id#8], Inner, BuildRight, false, true
:- Filter isNotNull(customer_id#2)
: + Scan ExistingRDD[customer_id#2,order_id#3L,amount#4L]
+ Exchange SinglePartition, EXECUTOR_BROADCAST, [plan_id=738]
+ Filter isNotNull(customer_id#8)
+ Scan ExistingRDD[customer_id#8,customer_name#9]

```

3 With AQE Enabled (5.28s) – Improved, but Not the Best

- **Join Type:** Sort-Merge Join (default)
- **AQE dynamically optimized shuffle partitions, but did not choose Broadcast Join automatically.**
- Execution time **improved from 11.26s → 5.28s**, but still **slower than explicit Broadcast Join**.

📌 Code Implementation:

```

spark.conf.set("spark.sql.adaptive.enabled", "true")
spark.conf.set("spark.sql.adaptive.skewJoin.enabled", "true") # Enables AQE skew join handling
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", "10MB") # enable broadcast joins

import time
start_time = time.time()

df_result = df_orders.join(df_customers, "customer_id", "inner")
df_result.count() # Trigger action

end_time = time.time()
print(f"Join Execution Time Without Optimization: {end_time - start_time:.2f} seconds")

```

📌 Logical & Physical Plan:

- AQE dynamically adjusts shuffle partitions.

- Still uses Sort-Merge Join (not the most optimal).

```

Just now (1s)
df_result.explain(True)

== Optimized Logical Plan ==
Project [customer_id#2, order_id#3L, amount#4L, customer_name#9]
+- Join Inner, (customer_id#2 = customer_id#8)
   :- Filter isNotNull(customer_id#2)
   :+ LogicalRDD [customer_id#2, order_id#3L, amount#4L], false
   +- Filter isNotNull(customer_id#8)
      +- LogicalRDD [customer_id#8, customer_name#9], false

== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- Project [customer_id#2, order_id#3L, amount#4L, customer_name#9]
   +- SortMergeJoin [customer_id#2], [customer_id#8], Inner
      :- Sort [customer_id#2 ASC NULLS FIRST], false, 0
      :+ Exchange hashpartitioning(customer_id#2, 200), ENSURE_REQUIREMENTS, [plan_id=1091]
      :  +- Filter isNotNull(customer_id#2)
      :    +- Scan ExistingRDD[customer_id#2,order_id#3L,amount#4L]
      +- Sort [customer_id#8 ASC NULLS FIRST], false, 0
         +- Exchange hashpartitioning(customer_id#8, 200), ENSURE_REQUIREMENTS, [plan_id=1092]
            +- Filter isNotNull(customer_id#8)
               +- Scan ExistingRDD[customer_id#8,customer_name#9]

```

4 With AQE + Manual Broadcast Join (2.34s) – Balanced Approach

- Join Type: Broadcast Hash Join
- Combines the benefits of AQE + Broadcast Join:
 - ✓ AQE optimizes shuffle partitions.
 - ✓ Manual Broadcast Join ensures efficient lookup.
- Execution time: Improved to 2.34s (near the best case).

📌 Code Implementation:

```

from pyspark.sql.functions import broadcast

start_time = time.time()

df_result_opt = df_orders.join(broadcast(df_customers), "customer_id", "inner")
df_result_opt.count() # Trigger action

end_time = time.time()
print("✓ Join Execution Time with Broadcast: {:.2f} seconds".format(end_time - start_time))

(3) Spark jobs
df_result_opt: pyspark.sql.dataFrame = [customer_id:string, order_id:long ... 2 more fields]

✓ Join Execution Time with Broadcast: 2.34 seconds

```

[Shift+Enter] to run and move to next cell
[Ctrl+Shift+F12] to open the command palette
[Esc H] to see all keyboard shortcuts

Key Takeaways & Recommendations

- Broadcast Join is a game-changer** when joining a **small** table (~10K rows) with a **large** one (~1M+ rows).
- AQE helps optimize shuffle partitions**, but doesn't always **automatically pick the best join strategy**.
- Skewed joins can create performance bottlenecks.** Identifying skewness using **groupBy analysis & Spark UI event timeline** is crucial.
- For best performance:**
 - If a **small dataset is involved**, manually apply Broadcast Join.
 - For **highly skewed datasets**, AQE + Broadcast Join gives the best balance.