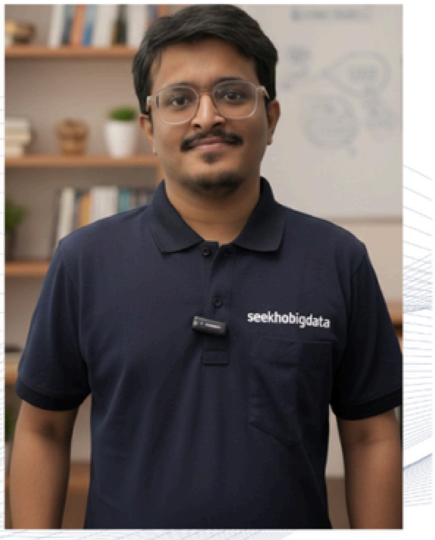
Seekho Bigdata
Data is the New Oil

Broadcast Join vs Shuffle Hash Join vs Sort-Merge Join



Karthik Kondpak
9989454737

Day 17 — Spark Optimization Topic

Broadcast Join vs Shuffle Hash Join vs Sort-Merge Join

(With physical plan and when to use which)

Spark has **3 major join strategies**.

Choosing the right one decides whether your job runs in **5 seconds or 5 minutes**.

1. Broadcast Hash Join (BHJ)

Fastest join in Spark.

✓ **When Spark Uses It**

- One side of the join is **small enough** (usually < 10 MB)
- `spark.sql.autoBroadcastJoinThreshold allows` broadcast

Default = **10 MB**

✓ What Happens Internally

- Small DataFrame → sent to **all executors**
- Big DataFrame remains distributed
- Executors keep small table in memory as a **hash map**
- Performs **hash lookup**, no shuffle

⚡ Performance

- Zero shuffle
- Super-fast
- Best for **dimension table joins**

✓Code example

```
df_big.join(broadcast(df_small), "id")
```

📜 Physical Plan

```
BroadcastHashJoin
   BuildRight
   BroadcastExchange
```

❌ When NOT to use

- Small table > threshold
- Executory memory is low
- Both tables huge

🎯 2. Shuffle Hash Join (SHJ)

Used when:

- Join keys are not sorted
- Data fits in hash table memory
- Broadcast is NOT possible

✔ What Happens Internally

- Both DataFrames are **shuffled by join key**
- Executors build **hash tables** for partitions
- Perform hash join

🔥 Faster than Sort-Merge Join if:

- Data is **small to medium**
- Join key has **good distribution**

📜 Physical Plan

```
ShuffledHashJoin
  Exchange  hashpartitioning
  Exchange hashpartitioning
```

❌ When NOT ideal

- Data is too large to fit in hash table
- Heavy skew
- Data requires sorting anyway

3. Sort-Merge Join (SMJ)

Most common join for **large-scale** workloads.

✔ When Spark Uses It

- Both sides are large
- Join key supports sorting
- Default join for many big-data operations

✔ What Happens Internally

- Both DataFrames **shuffle** by key
- Data is **sorted** within each partition
- Merge step happens like merging two sorted arrays

⚡ Good For

- Very large tables (100GB–100TB)
- Range join, time-series join
- Joins requiring ordering

📜 Physical Plan

```
SortMergeJoin
  Sort
  Exchange hashpartitioning
```

❌ When NOT ideal

- Sorting overhead is high
- Data not sortable
- Better alternatives exist (broadcast)

Side-by-Side Comparison Table

Feature	Broadcast Join	Shuffle Hash Join	Sort-Merge Join
Shuffle Sorting	No	✓ Yes	✓ Yes
required Memory usage	No	No	✓ Yes Low
Best for	High (broadcast table)	Medium	Very large tables
Fastest?	Small + large table	Medium tables	
Handles skew?	Yes	✓ Good	Slowest
	Poor	Poor	✓ Better

How to Read in Physical Plan

Use:

```
df.explain(True)
```

🔥 Broadcast Join Example

```
BroadcastHashJoin
  BuildRight
  BroadcastExchange HashedRelation...
```

🔥 Shuffle Hash Join Example

```
ShuffledHashJoin
  Exchange hashpartitioning
```

🔥 Sort-Merge Join Example

```
SortMergeJoin
  Sort
  Exchange hashpartitioning
```

Scenario (Swiggy)

Example: Joining

- `orders` → 450 GB
- `restaurants_dim` → 8 MB

Best join:

```
Broadcast Hash Join
```

Because:

- restaurants_dim < 10MB
- No shuffle
- Orders table streamed efficiently across nodes

Seekho Bigdata

Data is the New Oil

Let’s build your Data Engineering journey together!

📩 Call us directly at: 9989454737

🌐 https://seekhobigdata.com/