Sorting Strategies in Spark and Join Strategy for Parquet Files

Apache Spark provides various strategies for sorting data internally, especially when performing transformations such as joins, aggregations, and window functions. Choosing the appropriate sorting and join strategy is critical for optimizing performance and minimizing shuffle operations.

---

1. Sort-Merge Join (SMJ)

- **Description**: Spark sorts both datasets on the join key and then merges them, similar to the merge step in merge sort.
- **Use Case**: Default for large equi-joins when neither dataset is small enough to broadcast.
- **Performance**: Can be expensive due to full data shuffling and sorting.

2. Broadcast Hash Join (BHJ)

- **Description**: The smaller dataset is broadcast to all executors and hashed in memory.
- **Use Case**: Preferred when one of the datasets is small (e.g., <10MB) and fits into memory.
- **Performance**: Fastest join for skewed or unbalanced datasets with a small side.

3. Shuffle Hash Join

- **Description**: One side is hashed after shuffle; used when BHJ is not possible and no sorting exists.
- **Use Case**: Not ideal for large datasets.
- **Performance**: Medium; involves shuffling.

4. Bucketed Sort-Merge Join

- **Description**: If both datasets are pre-bucketed and sorted on join keys, Spark avoids full shuffling.
- **Use Case**: When tables are written with bucketing and sorting by join key.

5. Skewed Join Handling (Adaptive Query Execution)

- **Description**: Spark 3.0+ can automatically detect skewed keys and split them to avoid stragglers.
- **Use Case**: Large joins with skewed distribution.

Join Strategy for Parquet Files

Assume:

- **Parquet File 1**: detectionsDf (large dataset of object detections)
- **Parquet File 2**: locationsDf (small dataset of location metadata)

Selected Strategy: Broadcast Hash Join (BHJ)

Justification:

- locationsDf is small (typically < 10MB), suitable for broadcast.
- Avoids shuffles, faster execution.
- Scales well even in presence of data skew.

If Both Datasets Are Large:

Use Sort-Merge Join with partitioning:

Conclusion

The **Broadcast Hash Join** is the optimal strategy when locationsDf is small, providing fast and shuffle-free joins. If this condition is not met, **Sort-Merge Join** with repartitioning and adaptive execution should be used to maintain performance.

This join design ensures scalability, efficiency, and minimizes Spark shuffle overheads.