Spark Join Strategies

Spark offers several join strategies, each with distinct characteristics and optimal use cases:

1. Broadcast Hash Join

How it works:

- The smaller dataset is broadcast (copied) to all worker nodes
- A hash table is built from the smaller dataset in memory
- The larger dataset is processed by each executor, with lookups performed against the local hash table

Best suited for:

- Joining a large dataset with a small dataset (when one dataset can fit in memory)
- When the smaller dataset size is less than `spark.sql.autoBroadcastJoinThreshold` (default: 10MB)

Advantages:

- Eliminates the need for shuffling the larger dataset
- Significantly reduces network traffic
- Generally the fastest join strategy when applicable

Disadvantages:

- Memory intensive for the smaller dataset
- Not suitable when both datasets are large

2. Shuffle Hash Join

How it works:

- Both datasets are partitioned using the join key

- Corresponding partitions are sent to the same executor
- Hash tables are built for one dataset's partitions
- The other dataset's partitions probe the hash tables to find matches

Best suited for:

- When neither dataset fits in memory
- When join key cardinality is high and distribution is relatively even

Advantages:

- More memory efficient than broadcast joins
- Performs well when hash tables fit in memory

Disadvantages:

- Requires shuffling both datasets (network intensive)
- Performance degrades with skewed data
- In most modern Spark systems, Sort Merge Join is often preferred over Shuffle Hash Join for its more consistent performance

3. Sort Merge Join

How it works:

- Both datasets are partitioned by the join key
- Each partition is sorted by the join key
- Matching records are found by traversing the sorted datasets simultaneously

Best suited for:

- Large datasets that cannot fit in memory
- Default strategy for Spark SQL when broadcast threshold is exceeded

Advantages:

- Handles larger datasets than hash-based joins
- Performs well with high cardinality keys
- Resilient to data skew (better than hash joins)

Disadvantages:

- Slower than hash-based joins due to sorting overhead
- Requires both shuffling and sorting

4. Cartesian Join

How it works:

- Each record from one table is joined with every row in the other table
- Results in a cross product of both datasets

Best suited for:

- Non-equi joins where specific comparison operators are used (like <, >, <=, >=)
- Very specific use cases where a cross product is actually needed

Advantages:

- Supports complex join conditions that other join types cannot handle

Disadvantages:

- Extremely costly in terms of computation and memory
- Can lead to out-of-memory issues with large datasets
- Produces result sets that grow exponentially ($n \times m$ rows)

5. Broadcast Nested Loop Join

How it works:

- One table (typically smaller) is broadcast to all executors
- For each record in the broadcasted table, the join condition is evaluated against all records in the other table
- Often used with non-equi joins when one table is small enough to broadcast

Best suited for:

- Non-equi join conditions
- When one dataset is small enough to broadcast
- Complex join conditions that cannot be expressed as equality

Advantages:

- Supports complex join predicates
- More efficient than regular Cartesian joins when one table is small

Disadvantages:

- Still computationally expensive compared to hash or merge joins
- Requires the smaller table to fit in memory on each executor

Join Strategy Selection for Our Use Case

Analysis of our Datasets

1. **Dataset A (Detection Events)**:
    - Size: ~1,000,000 rows
    - Join key: geographical_location_oid
2. **Dataset B (Geographical Locations)**:
    - Size: 10,000 rows (with only 50 locations actually used)
    - Join key: geographical_location_oid

Recommended Strategy: Broadcast Hash Join

For joining our detection events (Dataset A) with geographical locations (Dataset B), a **Broadcast Hash Join** is the optimal strategy for the following reasons:

1. **Size Asymmetry**: Dataset B is significantly smaller than Dataset A (10,000 rows vs. 1,000,000 rows)
2. **Memory Efficiency**: Dataset B is small enough to be easily broadcast to all worker nodes without memory concerns
3. **Performance**: Broadcasting eliminates the need to shuffle the larger Dataset A, significantly reducing network traffic

References

- Spark Join Strategies: Mastering Joins in Apache Spark by Venkatesh Nandikolla