xData Technical Test for Nigel Mun

Distributed Computing Tasks II

1. Suppose there is data skew in one of the geographical locations in Dataset A. Please provide another code snippet on how you will be re-implement part of the program to speed up the computations (10 marks)

<code is shown in the TopItemsProcessorSkewed object>

I have leveraged the Adaptive Query Execution (AQE) combined with the salting technique to enhance computational efficiency in Spark applications.

To enable AQE, I used *spark.conf.set("spark.sql.adaptive.enabled", "true").* AQE optimizes query execution dynamically based on runtime data statistics, enabling the engine to consider and adapt different join strategies, and address join skew by redistributing skewed data across partitions more evenly.

I have also manually implemented a salting technique—where a random value is appended to the skewed join key, thereby increasing its cardinality— to further disperse the data across more partitions to combat the skew. This process involves adding a salt column (in *saltedDetectionsDF*, there are 2 additional columns *"salt"* and *"salted_oid"*) to the skewed dataset before the join and replicating the corresponding dataset across the range of salt values used, ensuring that the join operation distributes the workload more evenly across the cluster.

By combining AQE's automatic optimizations salting, we not only solve the bottlenecks caused by data skew but also enhance the overall performance by leveraging Spark's execution capabilities, resulting in a more efficient use of computational resources and significantly faster processing times for skewed datasets.

2. Explain the different sorting strategies in Spark and which strategy you will be adopting when joining Parquet File 1 and 2 if you are implementing the code in Spark Dataframe.

Spark implements several sorting strategies to manage how data is organized and joined across its distributed architecture, such as including Sort Merge Join, Broadcast Hash Join, and Shuffle Hash Join.

Sort Merge Join is Spark's default join strategy for large datasets that cannot fit into memory. It sorts both datasets by the join keys and then merges them with minimal shuffling. This strategy is particularly effective for evenly distributed datasets but can become less efficient if there's significant data skew.

Broadcast Hash Join involves broadcasting the smaller of the two datasets to all nodes in the cluster, where it's joined in memory with the partitioned data from the larger dataset. This strategy is efficient for joins where one dataset is significantly smaller than the other, reducing the need for data shuffling.

Shuffle Hash Join is used when both datasets are too large to be broadcasted but small enough to fit in memory after being partitioned. This strategy hashes the join keys and shuffles the data to ensure that matching keys are co-located, facilitating the join operation.

In my code, I have implemented a basic .join() function with no specification on the method, meaning that it will default to Sort Merge Join. For the dataset I created, I used the random function for the creation of data, which assumes that dataset is generally evenly distributed, hence Sort Merge join would seem flexible and optimized enough. However, when it comes to uneven datasets, using AQE allows Spark can adaptively choose the most efficient join strategy during runtime based on the actual data. This means it can automatically switch to a different join such as broadcast join if one of the DataFrames is below the broadcast threshold or implement optimizations to handle skewed data.