databricks

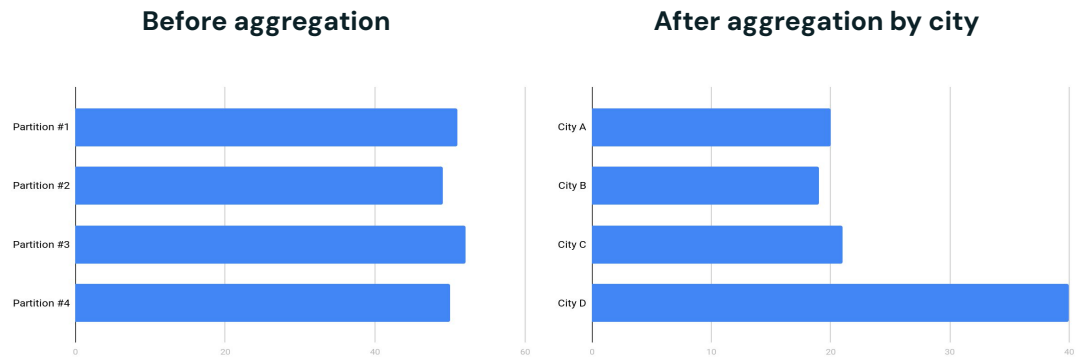
Code Optimization

LECTURE

Skew

This lecture explains what data skew is, its impact on performance, and how to handle and mitigate it effectively in Spark.

Skew – Before and After



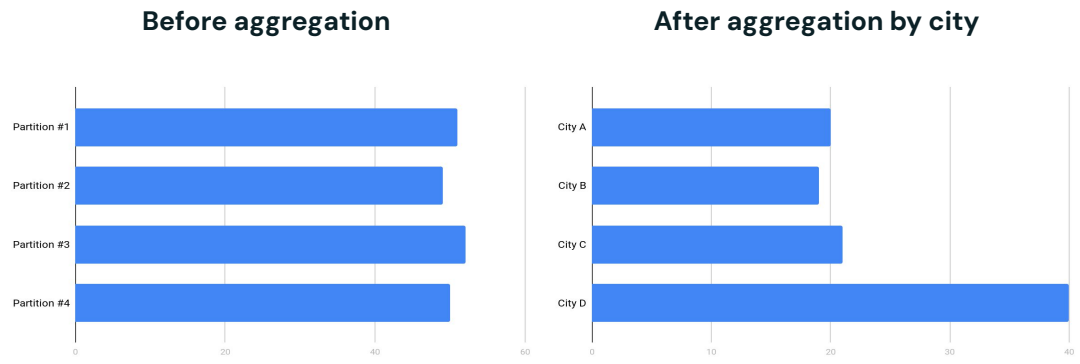
Skew

- Data is typically read in as 128 MB partitions and evenly distributed
- As the data is transformed (e.g. aggregated), it's possible to have significantly more records in one Spark-partition than another
- A small amount of skew is ignorable
- But large skews can result in spill or worse, hard to diagnose OOM Errors

Example

- In this example, we have 4 partitions. They should be evenly distributed after the initial ingest by Spark.
- City A, B, and C might be similar **IF** the data is correlated around something like population. The assumption here is that cities A, B and C are the same size.
- But city D has twice the population, thus twice as many records
- Who knows what cities E-ZZZ might look like - it doesn't matter for this illustration.

Skew – Before and After



Skew - Ramifications

If City D is 2x larger than A, B or C…

- It takes 2x as long to process
- It requires 2x as much RAM

The ramifications of that is...

- The entire stage will take as long as the longest running task
- We may not have enough RAM for these skewed partitions

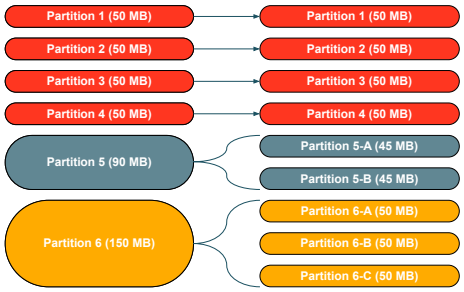
Notes:

- Once aggregated around the city, we can expect that the partition containing D will be proportionally larger than our other three cities.
- A larger partition means it has more records. More records means it will take Spark longer to process that partition.
- In our example here, it can take Spark twice as long to process City D as it would take to process A, B or C.

Handling Data Skew

Data Skew is unavoidable, Databricks handles this automatically

- In MPP systems, data skew significantly impacts performance because some workers are processing much more data.
- Most cloud DWs require a manual, offline redistribution to solve for data skew.
- With Adaptive Query Execution Spark automatically breaks down larger partitions into smaller, similar sized partitions.



So, how then do we handle data skew? What's nice about it is adaptive query execution allows for skew to be handled in a very nice way. Where a lot of cloud systems require you to manually worry about this, the AQE (adaptive query execution) implemented in Spark will make it so that some partitions that may be larger will be broken up into smaller pieces and then distributed to all of the cores to make sure the tasks are working with roughly the same amount of data

Skew – Mitigation

Three “common” solutions

1. Adaptive Query Execution (enabled by default in Spark 3.1)
2. Filter skewed values
3. Databricks’ [proprietary] Skew Hint
   - Easier to add a single hint than to salt your keys
   - Great option for version of Spark 2.x
4. Salt the join keys forcing even distribution during the shuffle
   - If none of the options are suitable, salting is the only alternative
   - It involves breaking a large skewed partition into smaller ones by adding random integers as suffixes.

Filter skewed values
If it’s possible to filter out the values around which there is a skew, then that will easily solve the issue. If you join using a column with a lot of null values, for example, you’ll have data skew. In this scenario, filtering out the null values will resolve the issue.
Skew hints
In the case where you are able to identify the table, the column, and preferably also the values that are causing data skew, then you can explicitly tell Spark about it using skew hints so that Spark can try to resolve it for you.
AQE skew optimization
Spark 3.0+’s AQE can also dynamically solve the data skew for you. It’s enabled by default but can be disabled. By default any partition that has at least 256MB of data and is at least 5 times bigger in size than the average partition size will be considered as a skewed partition by AQE. You can also change these values to fine-tune default AQE behavior. When your job has more than 2,000 shuffle partitions, Spark can no longer keep track of specific shuffle block sizes; instead, it only retains average sizes, making it impossible for AQE to detect skew. You can either reduce the number of shuffle partitions to fewer than 2,000 or change the following Spark configuration to a value greater than your shuffle partition count to resolve this problem:
Salting
If none of the above-mentioned options work for you, the only other option is to do salting. It’s a strategy for breaking a large skewed partition into smaller partitions by appending random integers as suffixes to skewed column values.

Thank you for completing this lesson and continuing your journey to develop your skills with us.