Optimizing Apache Spark

The Five Most Common Performance Problems

Spill

The 5 Most Common Performance Problems (The 5 Ss)

Spill

- Spill is the term used to refer to the act of moving an RDD from RAM to disk, and later back into RAM again
- This occurs when a given partition is simply too large to fit into RAM
- In this case, Spark is forced into [potentially] expensive disk reads and writes to free up local RAM
- All of this just to avoid the dreaded OOM Error

The 5 Most Common Performance Problems (The 5 Ss)
Spill - Examples

There are a number of ways to induce this problem:

- Set **spark.sql.files.maxPartitionBytes** to high (default is 128 MB)
- The **explode()** of even a small array
- The **join()** or **crossJoin()** of two tables
- Aggregating results by a skewed feature

The 5 Most Common Performance Problems (The 5 Ss)

Spill - Memory & Disk

In the Spark UI, spill is represented by two values:

- **Spill (Memory):** For the partition that was spilled, this is the size of that data as it existed in memory
- **Spill (Disk):** Likewise, for the partition that was spilled, this is the size of the data as it existed on disk

The two values are always presented together

The size on disk will always be smaller due to the natural compression gained in the act of serializing that data before writing it to disk

The 5 Most Common Performance Problems (The 5 Ss)
Spill - In the Spark UI

A couple of notes:

- Spill is only represented in the details page for a single stage...
  - **Summary Metrics**
  - **Aggregated Metrics by Executor**
  - The **Tasks** table
- Or in the corresponding query details
- This makes it hard to recognize because one has to hunt for it
- When no spill is present, the corresponding columns don't even appear in the Spark UI - that means if the column is there, there is spill somewhere

The 5 Most Common Performance Problems (The 5 Ss)

Spill - Spill Listener

See Experiment #6518, **Step A-2**

- The **SpillListener** is taken from Apache Spark's test framework
- The **SpillListener** is a type of **SparkListener** and tracks when a stage spills
- Useful to identify spill in a job when you are not looking for it
- We can see example usage in **Step B** through **Step E**

The 5 Most Common Performance Problems (The 5 Ss)

Spill - Examples

See Experiment #6518

- Note the four examples:
  - Step B: Spill induced by ingesting large partitions
  - Step C: Spill induced by unioning tables
  - Step D: Spill induced with explode operations
  - Step E: Spill induced by a skewed join
- For each example...
  - Find and note the total **Spill (Memory)** and **Spill (Disk)**
  - Find and note the min, median and max **Spill (Memory)** and **Spill (Disk)**
- Which of the four examples is uniquely different in how it manifests spill?

The 5 Most Common Performance Problems (The 5 Ss)
Spill - Examples, Review

Step	Min	25th	Median	75th	Max	Total
B - shuffle	~2 GB / ~550 MB	~2 GB / ~560 MB	~2 GB / ~565 MB	~2 GB / ~570 MB	~2 GB / ~580 MB	~33 GB
C - union	~2 GB / ~110 MB	~2 GB / ~120 MB	~2 GB / ~125 MB	~2 GB / ~130 MB	~2 GB / ~150 MB	~60 GB
D - explode	0 / ~1.5 GB	0 / ~1.5 GB	0 / ~1.5 GB	0 / ~1.5 GB	0 / ~1.5 GB	~750 GB
E - join*	0 / 0	0 / 0	0 / 0	0 / 0	6 GB / 3 GB	~50 GB

- In **Step B**, the config value **spark.sql.shuffle.partitions** is not managed
- **Steps C & D** simply grow too large as a result of their transformations
- In **Step E** the spill is a manifestation of the underlying skew

What can we do to mitigate spill?

The 5 Most Common Performance Problems (The 5 Ss)

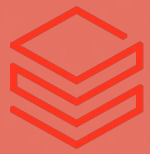
Spill - Mitigation

- The quick answer: allocate a cluster with more memory per worker
- In the case of skew, address that root cause first
- Decrease the size of each partition by increasing the number of partitions
  - By managing **spark.sql.shuffle.partitions**
  - By explicitly **repartitioning**
  - By managing **spark.sql.files.maxPartitionBytes**
  - Not an effective strategy against skew

The 5 Most Common Performance Problems (The 5 Ss)
Spill - Mitigation

- Ignore it - consider the example in **Step E**.
  - Out of ~800 tasks only ~50 tasks spilled
  - Is that 6% worth your time?

- However, it takes only one long task to delay an entire stage



databricks