Report

Task 1: PageRank Algorithm Implementation

In Task 1, the PageRank algorithm was implemented without custom RDD partitioning, with all workers active. The algorithm completed significantly faster than converting RDD to DataFrame (DF) and saving as CSV. While the algorithm took milliseconds, converting and saving took minutes.

Task 2: Custom RDD Partitioning

In Task 2, the performance of the PageRank algorithm was evaluated using different custom RDD partitioning strategies. The task was split into three trials:

1. Partitioned by 4 with just one RDD (linesRDD).
2. Partitioned by 8 with just one RDD (linesRDD).
3. Partitioned by 4 with all RDDs (lines and ranksRDD).

Compared to Task 1, where no custom partitioning was used, all trials in Task 2 took a shorter time in both algorithm implementation and converting/saving, although the time for printing the top 50 rows was slightly more for trials 1 and 2. This suggests that adding partitions helped the workers manage the workload between the worker nodes.

Furthermore, comparing between trials in Task 2, increasing the partitions at the beginning improved time (e.g., from 8 minutes to 6 minutes). However, partitioning multiple times was even better (e.g., from 8 minutes to 4 minutes, 6 minutes to 4 minutes) even with reduced partitions.

The reason why partitioning multiple times might be better is that it allows for finer-grained partitioning, which can reduce the amount of data that needs to be shuffled between nodes. This can lead to more efficient data processing, as related data is more likely to be co-located within the same partition. Thus, the best partitioning strategy that minimizes data shuffling and maximizes parallelism seems to involve having more instances of partitions and more parts. Further testing is recommended to confirm this hypothesis.

Task 3: Worker Process Failure

Two trials were conducted in Task 3: running Task 1 and killing one worker once it reached about 60% job completion, and running the best performing Task 2 set (the multiple partition set) and killing one worker once it reached about 60% job completion. Both trials took significantly longer to complete compared to their counterparts where no Worker was killed. This difference in completion time might be due to running tasks on the killed Worker needing to be rescheduled and executed on the other Worker. That can delay the time as the Master needs to determine which tasks need to be re-executed and then assigned to be executed on the second Worker.

Or if the Worker was killed when the data needs was being shuffled, the process could've been disrupted, leading to incomplete or incorrect shuffling, requiring the shuffling process to be restarted or adjusted. In general though, having more Workers should technically be better than having less Workers if everything runs smoothly.

Final Thoughts

Custom RDD partitioning can significantly improve PySpark application performance, particularly in algorithm implementation and data conversion/saving. The best partitioning strategy appears to involve more instances of partitions and more parts, although further testing is recommended. Worker process failure can impact performance, potentially leading to temporary degradation or job failure. It is crucial to consider fault tolerance mechanisms and optimize partitioning strategies to maximize parallelism and minimize data shuffling.

Outline

For the PDF report, you should report the application completion time with respect to Task 1-3 in Part 3. Present or reason about the differences in performance or your own findings, if any. Take a look at the DAG lineage graphs of applications, Executor statistics (e.g., shuffle read, shuffle write), and the number of tasks for every execution from the Spark Jobs web UI; these information may help you better understand the performance issues.

Task 1 had no partitions and all of the workers were active.

- The implementation of PageRank Algo took a significantly shorter time than converting RDD into DF and saving as csv
    - Algo wall time took ms, converting and saving took mins
    - Sim. pattern seen in cpu times

Task 2 was split into three trials:

1. Partitioned by 4 with just one RDD (linesRDD)
2. Partitioned by 8 with just one RDD (linesRDD)
3. Partitioned by 4 with all RDDs (lines and ranksRDD)

Compared to task 1:

- All trials took a shorter time in both algo implementation and converting/saving, but not printing top 50 (time was slightly more for trial 1&2)
- I assume that adding the partitions helped the workers with managing the workload between the worker nodes

Compared between trials:

- Increasing the partitions at the beginning improved time (from 8min to 6min), however, partitioning multiple times was even better (from 8min to 4min, 6min to 4min) even with reduced parts
- partition just once = data is divided into a fixed number of partitions just once, partition is not optimal
    - it can lead to uneven data distribution or inefficient data processing
    - data that is more likely to be accessed together might not be in same location
- Partitioning many times is better maybe because:
    - partitioning according to our data
    - can potentially reduce the amount of data that needs to be shuffled between nodes (related data is more likely to be co-located within the same partition)
- The best partitioning strategy that minimizes data shuffling and maximizes parallelism looks like having more instances of partitions AND having more parts, though it would be nice to test that out to confirm

Task 3 was split into two trials:

1. Ran task 1 and killed one worker once it reached about 60% job completion
2. Ran the best performing task 2 set (the multiple partition set) and killed one worker once it reached about 60% job completion

- Both trials took a significantly longer time to complete compared to their counterparts
- A reason for this increased time could be:
    - Master needs to determine which tasks need to be redone

- If Worker killed during reshuffling ⇒ shuffling is incomplete or completely incorrectly ⇒ need to resuffle
- In general, two workers should be faster than one