Cluster and Cloud Computing Assignment 1

Yufei Li (1338325) Junkai Zhang ()

1. Introduction

Parallel computing using High-Performance Computing (HPC) can greatly facilitate the processing of large-scale data; however, the application of such does not come without limitations. Our experiment with large twitter data file showed: 1. Utilizing multi-node, multi-core processing for data yields an increase in performance more than fivefold compared to single-node, single-core processing; 2. the enhancement in processing capability brought about by increasing the number of cores within the same node setting is comparatively limited.

The intension of this report is to briefly introduce the development process, discuss the possible explanations and lesson learned.

2. Methods

2.1. Filtering Attributes with Regular Expression

The calculation sentiment score for different time frames lies on the extraction of “created\_at” and “sentiment” attributes. We applied regular expression to extract those keywords.

Compared with Python’s JSON encoder and decoder, this process provided an incomplete solution to attribute extraction. It is prone to fragility in handling complex JSON structures and lacks robustness against format inconsistencies. Moreover, the readability and maintainability of regular expressions can significantly decrease as complexity increases, presenting challenges of less tolerant of errors.

However, after examination of the small datasets, we found out that the positions of keys are usually missing or inconsistent, which can lead to loading failure. Furthermore, the calculation only needs two attributes and we are not interested in the entire object, and the structure is beyond our concern.

Thus, given the simplicity of the required data, we do not need to parse the entire data structure. Also, in this particular data environment, utilizing regular expressions is simpler, accurate, and more flexible than parsing entire JSON objects.

2.2. Data Storage with Numpy Array

In our project, we utilizde two three-dimensional Numpy arrays for data storage. The primary array is dedicated to capturing the sentiment scores for each hour, while the secondary array tallies the count of tweets within the same time frame. The employment of three-dimensional arrays is particularly advantageous due to their innate capability to facilitate direct addition across different ranks, thereby streamlining the process of data aggregation.

In contrast, while dictionaries present a more intuitive method for data representation, potentially ensuring O(1) time complexity for data retrieval, they introduce significant challenges in terms of data aggregation. Specifically, the inherent structure of dictionaries does not lend itself to straightforward summation of values. Furthermore, the storage of keys within dictionaries imposes an additional memory overhead.

2.3. Data Distribution and Amalgamation

To facilitate equitable data distribution among multiple processes, we ascertain the total data size and accordingly partition it based on the aggregate number of processes (size). Each process, informed by its respective rank and the predetermined segment size it is to read, computes the precise location within the file from where it should commence reading. In instances where the file size does not permit an exact division among the processes, the final process is tasked with reading the residual data.

Acknowledging the potential edge case wherein the final line of the dataset might be split across two distinct data segments, we implement a strategic approach: Processes with a rank other than 0 (indicating they are not the inaugural process in the sequence) initially position their reading cursor one byte prior to their calculated starting point. They then proceed to read up to the first occurrence of a newline character (\n). This procedure ensures that each process begins reading from the start of a full, unbroken line, effectively mitigating the risk of commencing mid-line due to the division of data.

3. Results & Discussion

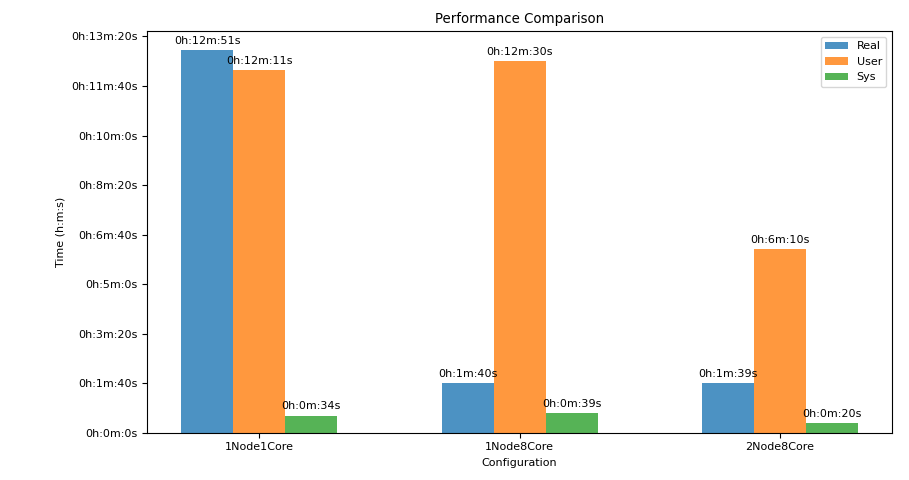


Figure 1. Time Consumptions for different Configurations

Under ideal circumstances, we would anticipate that the speedup ratio corresponds proportionally to the number of cores utilized; that is, an eight-core configuration is theoretically expected to achieve an eightfold increase in speed compared to a single-core setup. However, Figure 1 exhibits a speedup that surpasses this expectation, particularly noticeable when scaling from an eight-core single node to a sixteen-core dual-node configuration.

Such instances of superlinear speedup are atypical and contravene the fundamental tenet of Amdahl's Law, which posits that a fixed proportion of serial work sets a ceiling on the overall speedup. In our case study, as previously discussed, we capitalized on 'size' to facilitate concurrent data reading and balanced workload distribution across ranks. In the final aggregation phase, we employed collective communication (comm.reduce) for parallel summation. These factors may have contributed to the unexpected and extraordinary outcome observed.

Appendix

Scripts Used for Submitting the Job to SPARTAN

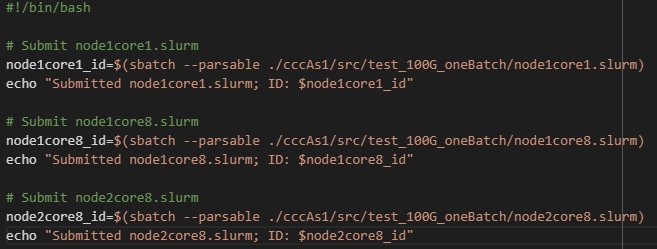


Figure 2. submit.sh

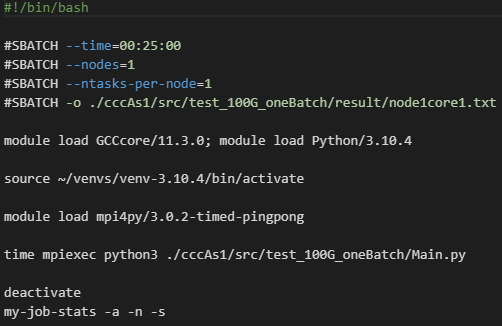


Figure 3. node1core1.slurm.

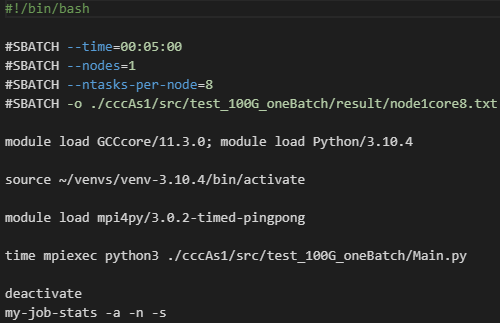


Figure 4. node1core8.slurm.

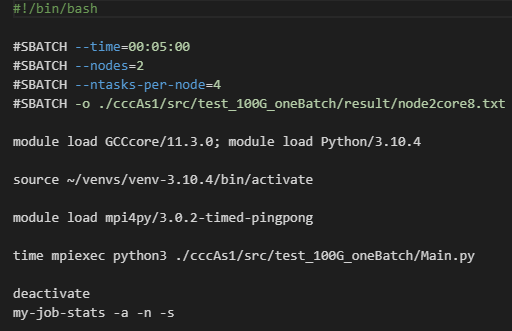


Figure 5. node2core8.slurm.