Cloud Computing Assignment 1

Andrew Stringer and Zexi Liu

April 2021

1 Code Overview

1.1 Approach

This project is divided into two main tasks - read and calculation. Firstly, the program read JSON files in gigabytes. The shortcoming of json.load() method for process big-size data, is solved by utilizing ijson pacakge.

After JSON file is parsed into smaller objects, we perform the calculation task. The Calculation takes to search and aggregation. That is, Search through a tweet for exactly matched words in affin.txt, where each matched word assigns to a sentimental score. In this first substring search, a Dictionary structure, (key: matched word, value: sentimental score) comes naturally to reduce loop-up time. After aggregating total scores for a tweet, another search for coordinate grid takes place. Because the geometry of a grid is a rectangle, two binary searches on longitudes and latitudes uniquely find the grid where a tweet belongs to. At last, further aggregation on scores is calculated at the grid level.

Parallelization is done at the calculation task. Divide and conquer techniques can be applied to the calculation task. The main calculation task is easy to divide - split the input data into partitions. And the result of each subtask can be aggregated to the final result. Therefore, to realized this technique, the main calculation task was divided into smaller calculation subtask, then "parallelized" to different cores, completing their calculation subtask at the same time.

Master-worker hybrid design pattern was deployed. As shown in Figure3, in our code, 1 node is set to be a master core. Its job is to read, and divided what's read, then scattered the partitions into the rest worker core, hence delegated sub-work to worker cores.

Noted, in Figure3, the master core also took sub-work. That is for two reasons. One is for the robustness of our program. Because there will be the case of one core, and no worker node to delegate work to. Secondly, while the master waits for the output of its worker, the master core is left unused. To further optimized the computation, it was decided to delegate some small portion of sub-work to the master core while it's waiting. At last, the master gathered all the results, aggregated and print out the output.

Assumptions on calculating score:

- special punctuation in different language encodings, for example, period in Chinese, or in Thai will not be stripped at the end of word.
- the beginning of the matched word is assumed to be a white space or punctuation. For example, 'hello! !good morning'. 'good' here would be a match.

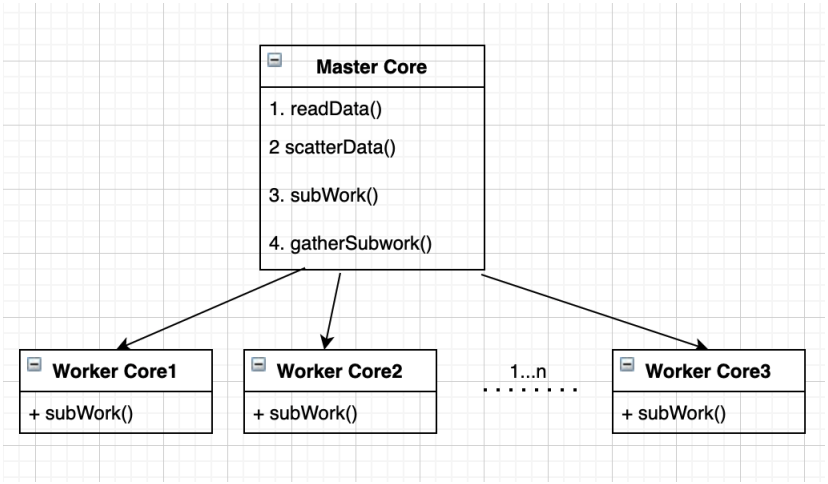

Figure3: schematics for parallelization

1.2 Executing code — SLURM scripts

The three SLURM scripts used in submitting these jobs to the MRC SPARTAN facility are those named TwitterSentiment_onenode_onecore.slurm, TwitterSentiment_onenode_eightcore.slurm, and TwitterSentiment_twonodes_eightcore.slurm.

The only noticeable differences in submission scripts are the resource requests and time request. The resource requests detail the number of nodes and cores in each submission (1 node 1 core, 1 node 8 cores, and 2 nodes 8 cores respectively). These requests are made in the lines beginning with #SBATCH and the line with srun. After seeing that the one node one core submission had a run time of under 6 minutes, the time request in the eight core submissions was updated to be 0-30 minutes.

Finally, the lines beginning with "module load" instruct SPARTAN to load Python 3 (version 3.7.4) for us to run the Python program described above.

2 Results

The results of this cloud computing project were intriguing. Not only did we discover we are living in the least happy area of Melbourne (B2), more importantly

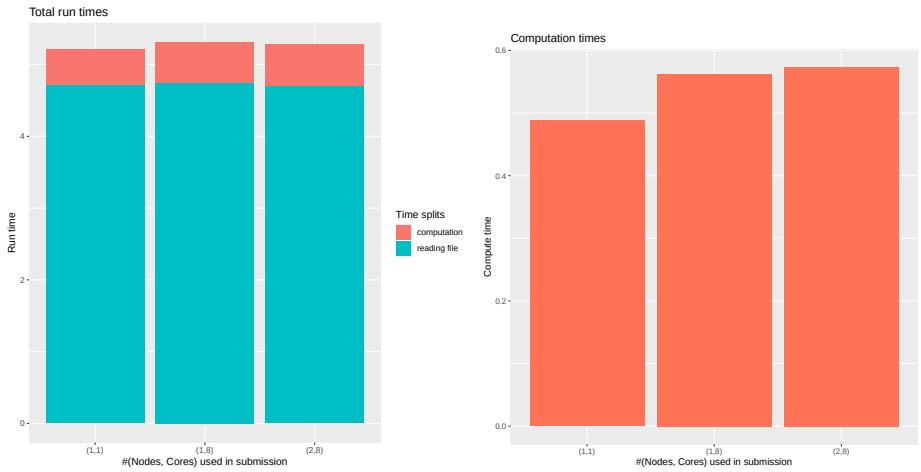

Figure 1: On left: total code running time for 1 node 1 core, 1 node 8 cores, and 2 nodes 8 cores segmented by time spent reading file vs doing computations. On right: time spent doing computations for each of the three node-core combinations.

it was revealed the fastest run time was produced by the one-node-one-core submission.

The total running time was comparable in each of the three node-core combinations taking 5.21 mins (1 node 1 core), 5.31 mins (1 node 8 cores), and 5.28 mins (2 nodes 8 cores). Moreover, the time spent reading the bigTwitter.json file was comparable in each submission taking 4.72 mins, 4.75 mins, and 4.71 mins in the respective configurations. However, the most pronounced time difference was for the computation time of the one-core vs eight-core configurations with the compute times being 0.49 mins, 0.56 mins, and 0.57 mins. Additionally, as the computation times are an order of magnitude smaller than the file reading times, the impact of this difference on the total run time is not noteworthy. Figure 1 highlights these results.

3 Interpretation

These results are interesting as they run counter to our expectations entering this computational experiment. It was anticipated that the two configurations with 8 cores would have a shorter run time than the one-node-one-core configuration. Moreover under Amdahl's analysis, the non-parallel part is the read time, $\sigma = 4.72$ mins, and the parallelisable part is the compute time, $\pi = 0.49$ mins — using 8 cores, the total run time is projected to drop from $\sigma + \pi = 5.21$ mins to $\sigma + \pi/8 = 4.78$ mins. Understanding that this reflects an optimistic best-case outcome, this projection was used as a first level of analysis and a benchmark to compare our results against.

What features may have led to this unexpected result? One possibility is that the non-zero latency of message passing took longer than the time for 7 computations by the single core. This interpretation would reliably explain the difference between using 1 core vs 8 cores while also explaining the slightly longer compute time for 2 nodes vs 1 node when using 8 cores. We cannot firmly make this conclusion without further analysis, though it appears the most appropriate explanation for these results.

If this is the case, then it is evidently a major limitation of both Amdahl's and Gustafson-Barsis' laws as neither account for this integral feature of distributed computing. In doing so Amdahl's law misled us into believing that using 8 cores would speed up our program when it slowed our program down (albeit a very minor slow-down).

On the other hand, Figure 1 suggested the nature of the task is I/O bounded instead of CPU bounded. That is, the read/write task takes the majority of the time than the calculation task. In the other words, the workload on CPUs is not as heavy in the first place for both one core and eight cores. Therefore, it would not be too surprising that the latency time is larger than the CPU calculation time.

Future work building off this computational study may include: testing if this hypothesis is really behind these results, and extending Amdahl's model of distributed computing to account for non-zero latency (I'm an applied mathematician).