Cluster and Cloud Computing Assignment 1 Report

Junkai Zhang 1166036 Yufei Li 1338325

April 11, 2024

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 five-fold 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 intention 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 utilized 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.

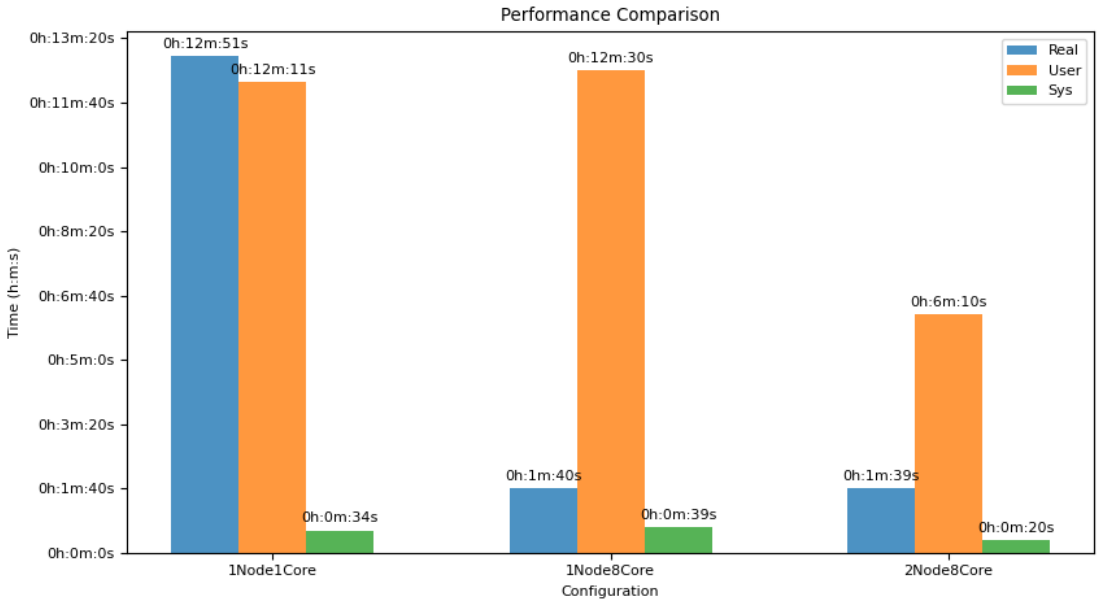

Figure 1: Time Consumption for different Configurations

2.3 Data Distribution and Amalgamation

The idea of our process paralyzing is divide the large chunk of data in to small chunks and separate them to each cores to make the task run faster and use less RAM. The multi-process part is by giving each core a small chunk of data, process the data within each core and return all results to core 0 and core 0 will output the final result.

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. 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 and Discussion

For project results, see Figure 2, 3, 4. 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 eight-fold 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 super linear 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.

4 Appendix

```
12
13    The Happiest Hour: [12, 31, 13] with sentiment of 3658.952143925703
14    The Happiest Day: [12, 31] with sentiment of 30402.262571420666
15    The Most Active Hour: [5, 21, 12] with tweets of 39061.0
16    The Most Active day: [5, 21] with tweets of 424372.0
17
18    real    12m51.778s
19    user    12m11.750s
20    sys     0m34.572s
21    Job ID           : 58229355
22    Cluster          : spartan
23    User/Project     : yufeil10/comp90024
24    Nodes            : 1
25    Wall-clock time  : 00:12:56 / 00:25:00
26
27    Displaying overall resources usage from 2024-04-11 06:55:57 to 2024-04-11 07:08:53:
28
29    NODE            CPU#        TOT%   ( USR   / SYS   / WIO   / IDLE  )
30
31    spartan-bm001 :
32                    CPU# 1    : 97.7   (  92.7 /   5.0 /   0.0 /   2.3 )
33
34
35    Allocated CPUs            : 1
36      CPUs with usage <25%    : 0
37      CPUs with usage <50%    : 0
38      CPUs with usage >50%    : 1
39
40    Memory used (RAM)         : 3.9%  [162MB of 4195MB]
41
```

Figure 2: Result of 1 Node 1 Core

```
12
13    The Happiest Hour: [12, 31, 13] with sentiment of 3658.952143925703
14    The Happiest Day: [12, 31] with sentiment of 30402.262571420666
15    The Most Active Hour: [5, 21, 12] with tweets of 39061.0
16    The Most Active day: [5, 21] with tweets of 424372.0
17
18    real    1m40.924s
19    user    12m30.557s
20    sys     0m39.248s
21    Job ID           : 58229356
22    Cluster          : spartan
23    User/Project     : yufeil10/comp90024
24    Nodes            : 1
25    Wall-clock time  : 00:01:46 / 00:05:00
26
27    Displaying overall resources usage from 2024-04-11 06:54:12 to 2024-04-11 06:55:58:
28
29    NODE            CPU#        TOT%   ( USR   / SYS   / WIO   / IDLE  )
30
31    spartan-bm024 :
32                    CPU# 1    : 74.6   (  70.9 /   3.7 /   0.0 /  25.4 )
33                    CPU# 2    : 74.7   (  71.1 /   3.6 /   0.0 /  25.3 )
34                    CPU# 3    : 74.5   (  70.9 /   3.6 /   0.0 /  25.5 )
35                    CPU# 4    : 74.8   (  71.1 /   3.7 /   0.0 /  25.2 )
36                    CPU# 5    : 74.6   (  71.0 /   3.6 /   0.0 /  25.4 )
37                    CPU# 6    : 75.4   (  71.7 /   3.7 /   0.0 /  24.6 )
38                    CPU# 7    : 74.6   (  71.1 /   3.6 /   0.0 /  25.4 )
39                    CPU# 8    : 74.6   (  71.1 /   3.5 /   0.0 /  25.4 )
40
41
42    Allocated CPUs            : 8
43      CPUs with usage <25%    : 0
44      CPUs with usage <50%    : 0
45      CPUs with usage >50%    : 8
46
47    Memory used (RAM)         : 3.8%  [1269MB of 33555MB]
48
```

Figure 3: Result of 1 Node 8 Core

```
12
13      The Happiest Hour: [12, 31, 13] with sentiment of 3658.952143925703
14      The Happiest Day: [12, 31] with sentiment of 30402.262571420666
15      The Most Active Hour: [5, 21, 12] with tweets of 39061.0
16      The Most Active day: [5, 21] with tweets of 424372.0
17
18      real    1m39.431s
19      user    6m10.259s
20      sys     0m20.013s
21      Job ID           : 58229357
22      Cluster          : spartan
23      User/Project     : yufeil10/comp90024
24      Nodes            : 2
25      Wall-clock time  : 00:01:45 / 00:05:00
26
27      Displaying overall resources usage from 2024-04-11 06:54:12 to 2024-04-11 06:55:57:
28
29      NODE            CPU#        TOT%   ( USR   / SYS   / WIO   / IDLE  )
30
31      spartan-bm024 :
32                      CPU# 1    : 74.6   (  70.8 /   3.7 /   0.0 /  25.4 )
33                      CPU# 2    : 74.5   (  70.9 /   3.6 /   0.0 /  25.5 )
34                      CPU# 3    : 75.4   (  71.6 /   3.9 /   0.0 /  24.6 )
35                      CPU# 4    : 75.1   (  71.3 /   3.8 /   0.0 /  24.9 )
36
37      spartan-bm025 :
38                      CPU# 5    : 83.1   (  78.5 /   4.6 /   0.0 /  16.9 )
39                      CPU# 6    : 83.1   (  78.5 /   4.6 /   0.0 /  16.9 )
40                      CPU# 7    : 83.0   (  78.7 /   4.3 /   0.0 /  17.0 )
41                      CPU# 8    : 83.1   (  78.6 /   4.5 /   0.0 /  16.9 )
42
43
44      Allocated CPUs           : 8
45        CPUs with usage <25%   : 0
46        CPUs with usage <50%   : 0
47        CPUs with usage >50%   : 8
48
49      Memory used (RAM)        : 1.9%  [641MB of 33555MB]
50
```

Figure 4: Result of 2 Node 8 Core

```bash
#!/bin/bash

# Submit node1core1.slurm
node1core1_id=$(sbatch --parsable ./cccAs1/src/test_100G_oneBatch/node1core1.slurm)
echo "Submitted node1core1.slurm; ID: $node1core1_id"

# Submit node1core8.slurm
node1core8_id=$(sbatch --parsable ./cccAs1/src/test_100G_oneBatch/node1core8.slurm)
echo "Submitted node1core8.slurm; ID: $node1core8_id"

# Submit node2core8.slurm
node2core8_id=$(sbatch --parsable ./cccAs1/src/test_100G_oneBatch/node2core8.slurm)
echo "Submitted node2core8.slurm; ID: $node2core8_id"
```

Figure 5: submit.sh

```
#!/bin/bash

#SBATCH --time=00:25:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH -o ./cccAs1/src/test_100G_oneBatch/result/node1core1.txt

module load GCCcore/11.3.0; module load Python/3.10.4

source ~/venvs/venv-3.10.4/bin/activate

module load mpi4py/3.0.2-timed-pingpong

time mpiexec python3 ./cccAs1/src/test_100G_oneBatch/Main.py

deactivate
my-job-stats -a -n -s
```

Figure 6: node1core1.slurm

```
#!/bin/bash

#SBATCH --time=00:05:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH -o ./cccAs1/src/test_100G_oneBatch/result/node1core8.txt

module load GCCcore/11.3.0; module load Python/3.10.4

source ~/venvs/venv-3.10.4/bin/activate

module load mpi4py/3.0.2-timed-pingpong

time mpiexec python3 ./cccAs1/src/test_100G_oneBatch/Main.py

deactivate
my-job-stats -a -n -s
```

Figure 7: node1core8.slurm

```
#!/bin/bash

#SBATCH --time=00:05:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH -o ./cccAs1/src/test_100G_oneBatch/result/node2core8.txt

module load GCCcore/11.3.0; module load Python/3.10.4

source ~/venvs/venv-3.10.4/bin/activate

module load mpi4py/3.0.2-timed-pingpong

time mpiexec python3 ./cccAs1/src/test_100G_oneBatch/Main.py

deactivate
my-job-stats -a -n -s
```

Figure 8: node2core8.slurm

Submit all 3 jobs through one .sh file

```
sbatch ./cccAs1/src/test_100G_oneBatch/submit.sh
sprio -j 58229354
my-job-stats -j 58229354
```

Check Job Status for node1core1

```
vim ./cccAs1/src/test_100G_oneBatch/node1core1.slurm
vim ./cccAs1/src/test_100G_oneBatch/result/node1core1.txt
sprio -j 58229355
my-job-stats -j 58229355
```

Check Job Status for node1core8

```
vim ./cccAs1/src/test_100G_oneBatch/node1core8.slurm
vim ./cccAs1/src/test_100G_oneBatch/result/node1core8.txt
sprio -j 58229356
my-job-stats -j 58229356
```

Check Job Status for node2core8

```
vim ./cccAs1/src/test_100G_oneBatch/node2core8.slurm
vim ./cccAs1/src/test_100G_oneBatch/result/node2core8.txt
sprio -j 58229357
my-job-stats -j 58229357
```

Figure 9: command line