

COMP90024 Cluster and Cloud Computing

Assignment 1 - HPC Twitter GeoProcessing

Yixin Chen 522819

a.chen15@student.unimelb.edu.au

Introduction

In this project, the target is searching a large geocoded Twitter dataset to identify and the geographical location of the tweets. The twitter file is around ten gigabytes and in JSON format. In order to perform searching on a large file like this and make use of Spartan system, I have developed a parallelized application. To achieve parallelization, MPI is used for communication between cores when multi-tasking. I used the package mpi4py because it is easy to learn and implement, therefore the application is written in Python language.

Configuration

Project1.py is the main application for this assignment. To make it run properly, a few conditions must be met. Firstly, the main program should be uploaded on to the Spartan system within the same directory of the twitter file and the Melbourne grid file (or their shortcut path). Then a script must be uploaded before using command line arguments to submit job. Example of a job script runs on 1 node and 8 cores is shown below:

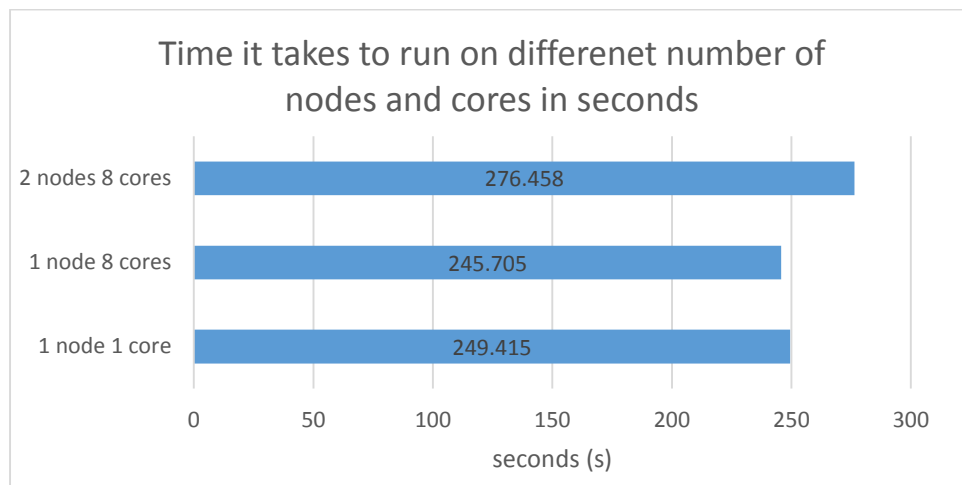
```
1 #!/bin/bash
2 #SBATCH --time=00:30:00
3 #SBATCH --nodes=1
4 #SBATCH --ntasks=8
5 #SBATCH --ntasks-per-node=8
6 module load Python/3.4.3-goolf-2015a
7 time mpirun python project1.py
```

After that, use the command “sbatch proj1_1n8c.slurm” (the name of script file is proj1_1n8c.slurm) to submit the job. The output will be generated on the same directory when the job is finished.

Approach to parallelize code

The approach to parallelize code used in this assignment is partitioning data. First of all, all of the processes (each core) will read the Melbourne grid data and store it locally for later use. One of the process (in this case, rank == 0) will read the twitter data line by line, pre-process it by only taking the coordinates and store the coordinates in a list. Then the process will partition the data by the size of number of cores and scatter those list of coordinates as tasks to all other processes including itself. Each of the processes will then compare the coordinates with Melbourne grid and classify it as a location ID. Finally, the process labelled with rank 0 gathers all location IDs and uses Counter method to produce a dictionary format result.

Performance on different number of nodes and cores



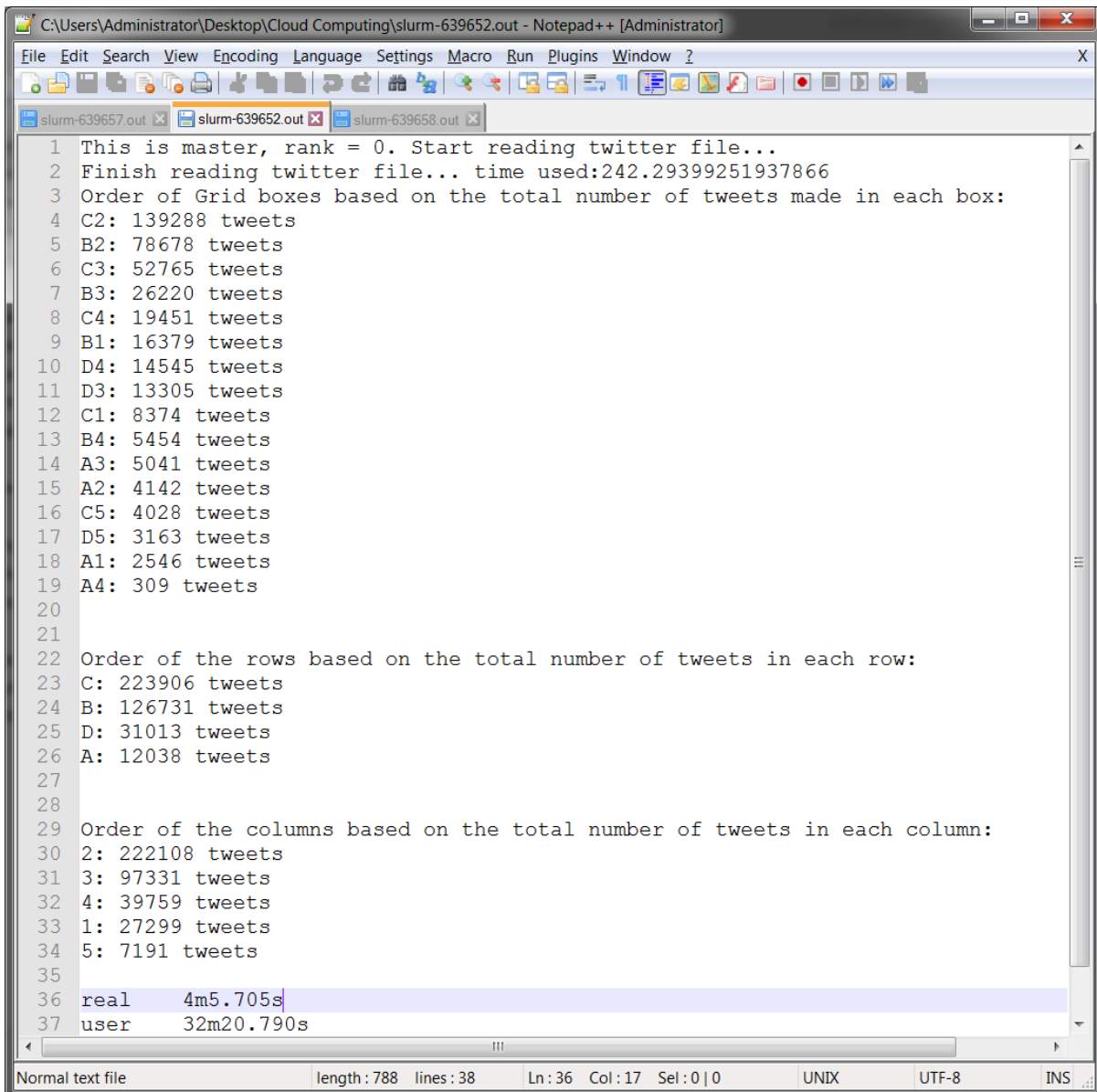
The time that the application takes to run on 1 node 1 core, 1 node 8 cores and 2 nodes 8 cores are 249.415 seconds, 245.705 seconds and 276.458 seconds respectively. The performance of the application is similar on different numbers of nodes and cores. The execution time of the application runs on 1 node 1 core is close to, or even lesser than those taking the advantage of parallelizing code (1 node 8 cores/2 nodes 8 cores). This is not a desirable outcome. By adding timer to my "readtwitter()" function, I have found that the time it takes for the rank process reading twitter file and pre-processing is huge. For example, this function spends 242.294 seconds for 1 node 8 cores configuration which is 98.6% of the total execution time. During that time, all other 7 cores are idle. The part of code that take advantage of parallelizing only executes in very small amount of time. After deducting the reading and pre-processing time from the total execution time, we now have 8.059s, 3.411s and 5.368s respectively. This is reasonable since 1 node 8 cores and 2 nodes 8 cores take the advantage of parallel computing, their execution time after deduction is less than the time for 1 node 1 core. On the other hand, 2 nodes 8 cores takes longer to run than 1 node 8 cores because there might be communication overheads between different nodes.

	Total execution time	Reading twitter file & pre-processing	Other tasks such as scatter, gather and producing results
1 node 1 core	249.415s	241.356s	8.059s
1 node 8 cores	245.705s	242.294s	3.411s
2 nodes 8 cores	276.458s	271.093s	5.368s

Conclusion

From the observation, it shows that I does not make the best use of Spartan system and parallel programing because most of the cores are idle during the execution. If time permits I will implement a master slave version to observer the differences in performance. On the other hand, if the number of lines in the twitter file is pre-known or the twitter file is chopped into smaller files equal to the number of cores, there will be an increase in performance for the partitioning data approach. All processes can be assigned with tasked before execution and start working immediately after the application runs. The idle time of cores will be greatly reduced. However the downside is that different cores or processes may have various execution time on the same amount of workload. I will do more research in the future to figure out utilising all the cores with HPC.

Example of output:



```
1 This is master, rank = 0. Start reading twitter file...
2 Finish reading twitter file... time used:242.29399251937866
3 Order of Grid boxes based on the total number of tweets made in each box:
4 C2: 139288 tweets
5 B2: 78678 tweets
6 C3: 52765 tweets
7 B3: 26220 tweets
8 C4: 19451 tweets
9 B1: 16379 tweets
10 D4: 14545 tweets
11 D3: 13305 tweets
12 C1: 8374 tweets
13 B4: 5454 tweets
14 A3: 5041 tweets
15 A2: 4142 tweets
16 C5: 4028 tweets
17 D5: 3163 tweets
18 A1: 2546 tweets
19 A4: 309 tweets
20
21
22 Order of the rows based on the total number of tweets in each row:
23 C: 223906 tweets
24 B: 126731 tweets
25 D: 31013 tweets
26 A: 12038 tweets
27
28
29 Order of the columns based on the total number of tweets in each column:
30 2: 222108 tweets
31 3: 97331 tweets
32 4: 39759 tweets
33 1: 27299 tweets
34 5: 7191 tweets
35
36 real 4m5.705s
37 user 32m20.790s
```

Normal text file length: 788 lines: 38 Ln: 36 Col: 17 Sel: 0 | 0 UNIX UTF-8 INS