Cluster and Cloud Computing Assignment 1 Report

COMP90024

Haonan Zhong *(867492)* Zhi Hern Tom *(1068268)*

Abstract

Parallel computing refers to the process of decomposing a computationally expensive problem into smaller and simpler subproblems that can be solved concurrently by multiple processors for faster processing. Message Passing Interface (MPI) is a programming interface for parallel computing on distributed memory systems. It enables multiple processes to communicate with each other, exchange data and synchronize their execution in parallel.

Keywords— Message Passing Interface, Twitter Analytic, mpi4py, HPC, Slurm

1 Introduction

This project focuses on implementing a parallelized message-passing application that utilizes the High-Performance Computing (HPC) facility SPARTAN available at the university. The application aims to process a large Australia-wide Twitter dataset to gain geospatial insights of Twitter users in different states, namely, identify the top ten tweeters of Australia, count the total number of tweets made in various Capital Cities, and finally identify the top ten Twitter users who have tweeted in the most Greater Capital cities and counting the frequency they have tweeted from those locations. This report outlines our approach to designing the application's system architecture, as well as discuss the performance benchmark of using different configurations of nodes and cores.

1.1 Twitter Dataset

Three JSON format twitter datasets are provided, where they have the same general structure but vary in size. The final analysis and benchmark comparison will primarily focus on *bigTwitter.json*, which has an approximate size of 18.74 GB. In contrast, the other two smaller datasets are solely used for implementation and testing. Each JSON object of the dataset encapsulates core attributes that describes a given tweet. However, due to the scope of the project, we will only focus on the following two attributes, the *Author ID* and *tweet sent location*.

1.2 Geospatial Dataset

The *sal.json* dataset provides some brief spatial information on towns/suburbs in Australia, such as *the State and Territory* (STE) code, *the Greater Capital City Statistical Areas* (GCC) code, and *the Suburbs and Localities* (SAL) code. However, due to the scope of the project, we will only focus on the GCC code, as it helps us to identify whether a suburb/town locates within a Greater Capital City. One of the challenges we faced with this dataset is that some of the suburbs share the same name but are located in different states. For example, *Box Hill* appears in both Victoria and New South Wales. Another issue with the dataset is that suburbs like *Central Coast* was further divided into smaller sub-regions, for example, `"alison (central coast - nsw)"` and `"green point (central coast - nsw)"`. However, the twitter dataset encodes all these locations as *Central Coast*.

2 Parallel Computing

The application was developed using the Python programming language and utilises the MPI for Python (`mpi4py`) package, enabling the application to run in parallel on multiple processors or nodes. Furthermore, we make use of the Slurm job script, which allows us to specify the job requirements, such as the number of cores, amount of memory, and application expected run time when submitting the job to the Spartan HPC system.

2.1 Divide and Conquer Paradigm

Our implemented algorithm heavily relies on the divide and conquer paradigm, which utilises the `Scatter` and `Gather` function provided in the `mpi4py` package. The idea is to distribute the data/workload from the root process to all other worker processes within the communicator. Once all three tasks are completed by each processes, the root process then collects the computed result from all worker processes using the `Gather` function, and aggregate the final result for display. Each process stores a `twitterData` object, which consists of three counters for handling result for the three tasks. In addition, data are read line by line to avoid memory overflow.

2.2 Decomposing the Data

The dataset is split into `n` approximately equal-sized blocks using the function `create_block`, where `n` is the number of processes specified by the Slurm script. The root process then scatters the parameters `block_start` and `block_end` to each worker process, which is the byte offsets unique for each process and denotes the section of data they are responsible for processing.

2.3 Task 1 - Top Ten Tweeters

Task one involves counting and identifying Australia's top ten tweeters in terms of the number of tweets made, irrespective of where they tweeted. The process is fairly simple, identify the author of each tweet and record the number of occurrence. Our approach to solve the task is to perform pattern match when reading the dataset line by line, check whether the line contains the substring `'author_id:'`, and use regex to extract the ID number if the substring exist, and the counter variable `user_counter` will then keep track of the number of tweets that all users made.

2.4 Task 2 - Number of Tweets in Greater Capital Cities

Task two involves counting the number of tweets made in each Australian Greater Capital Cities. It is important to state that tweets made by users in rural locations or tweets that contains approximate locations will be ignored in this task.

The overall procedure are similar to the one of task 1, where we use pattern matching when reading line by line, and extract the location name using regex when the substring `'full_name'` is present. To identify whether the location lies within a greater capital city, we have preprocess *sal.json* into a look up dictionary, where keys are the GCC code and the corresponding value are a list of suburbs/towns within the greater capital city. The counter variable `location_counter` will then keep track of the number of tweets made in each greater capital cities.

Furthermore, we will discuss our approach to tackle the ambiguities where tweets cannot be resolved to a single unique location. For suburbs with the same name but located in different states, we discovered that their name are attached with state abbreviation to make them distinguishable in *sal.json*, for example, `box hill (vic.)` and `box hill (nsw)`. Therefore, during the preprocessing stage, suburb name with state abbreviation attached will be reformatted to, e.g. `box hill, victoria` and `box hill, new south wales`, before being added to the look up dictionary. As for the second issue mentioned in section 1.2,

only the location name between `(` and `' - '` will be added to the look up dictionary. These two resolution should allow the algorithm to resolve tweets to a single location.

2.5 Task 3 - Tweeters Who Tweeted in the Most GCCs

Task three involves identifying the top ten tweeters who tweeted in the most greater capital cities. In comparison to Task two, Task three was relatively less complex, as we had previously developed a look up dictionary that stored tweet IDs and corresponding location-based counts. Consequently, a straightforward summation was employed to obtain the desired results.

2.6 Slurm Scripts and Submission on SPARTAN

To execute the code on the SPARTAN cluster, it is essential to create and utilize Slurm scripts for proper configuration and management of settings on the platform. The template provided below can be employed as a foundation for crafting a professional Slurm script that meets the specific requirements of this project:

```
#!/bin/bash
#SBATCH --partition=physical
#SBATCH --nodes=[xxx]
#SBATCH --ntasks=[xxx]
#SBATCH --cpus-per-task=[xxx]
#SBATCH --time=0-1:00:00
module load foss/2019b
module load python/3.7.4
module load mpi4py/3.0.2-timed-pingpong
srun -n [xxx] python -m mpi4py main.py -location [xxx] -dataset [xxx]
my-job-stats -a -n -s
```

where the text within the brackets `[xxx]` for each `#SBATCH` directive represents the specific parameter values for the corresponding configuration options. The `-nodes` option denotes the total number of nodes, while the `-ntasks` option specifies the aggregate number of cores. The `-cpus-per-task` option indicates the number of cores allocated per node.

Additionally, the `[xxx]` following the -n flag signifies the designated number of cores for the task in question. The `[xxx]` placeholders succeeding the `-location` and `-dataset` flags represent the file paths to the *sal.json* configuration file and the dataset, respectively.

To execute the Slurm script in a professional manner, please submit it via the terminal using the command `sbatch <script_name>.slurm`. For example, to submit the 2n8c script, enter `sbatch 2n8c.slurm`. This will initiate the Slurm job scheduler to manage the allocated resources and run the specified tasks in the script.

3 Result and Benchmark Comparison

The Twitter dataset was processed using three different sets of configuration, namely, one node - one core, one node - eight cores, and two nodes - eight cores (with four cores per node). The results were recorded and displayed as followed:

Top 10 Tweeters

Rank	Author ID	Number of Tweets Made
#1	1498063511204761601	68477
#2	1089023364973219840	28128
#3	826332877457481728	27718
#4	1250331934242123776	25350
#5	1423662808311287813	21034
#6	1183144981252280322	20765
#7	1270672820792508417	20503
#8	820431428835885059	20063
#9	778785859030003712	19403
#10	1104295492433764353	18781

Total Number of Tweets in Various Capital Cities

Greater Capital City	Number of Tweets Made
1gmel (Greater Melbourne)	2279689
2gsyd (Greater Sydney)	2216307
3gbri (Greater Brisbane)	853687
4gper (Greater Perth)	589188
5gade (Greater Adelaide)	452469
6ghob (Greater Hobart)	89970
7gdar (Greater Darwin)	46364

```
Top 10 Number of Unique City Locations and #Tweets
Rank    Author ID                   Number of Unique City Locations and #Tweets
#1      1429984556451389440         7(#1896 tweets - 1868gmel,2gade,7gper,11gsyd,6gbri,1ghob,1gdar)
#2      17285408                    7(#1166 tweets - 1041gsyd,40gbri,60gmel,11ghob,3gade,4gdar,7gper)
#3      702290904460169216          7(#1129 tweets - 247gmel,120gade,222gbri,19gdar,330gsyd,152gper,39ghob)
#4      87188071                    7(#363 tweets - 28gade,64gbri,52gper,114gsyd,15ghob,85gmel,5gdar)
#5      1361519083                  7(#260 tweets - 36gmel,193gdar,18gsyd,9gade,1gper,2ghob,1gbri)
#6      502381727                   7(#240 tweets - 214gmel,8gbri,8ghob,3gper,4gade,2gsyd,1gdar)
#7      921197448885886977          7(#195 tweets - 37gbri,20gade,28gper,49gsyd,56gmel,4ghob,1gdar)
#8      1382217768649465858         7(#189 tweets - 50gmel,13gade,50ghob,14gbri,8gdar,43gper,11gsyd)
#9      601712763                   7(#136 tweets - 44gsyd,39gmel,1gdar,14gper,8ghob,19gade,11gbri)
#10     2647302752                  7(#76 tweets - 32gbri,3gdar,3gade,4gper,13gsyd,16gmel,5ghob)
```

Furthermore, the running time of each configuration are shown in figure 1, the application run time of one node - eight cores and two nodes - eight cores are roughly the same, with two nodes eight cores taken an additional three seconds, which is likely due to the communication overhead involved in disseminating instructions to another node via the network and subsequently collecting results from that node. The one node one core configuration performed the poorest, requiring around 708 seconds to finish. These benchmark outcomes were expected, highlighting the significant advantage of parallelization in the context of big data processing. Overall, the other two configurations have both achieved 680% speedup compared to one node - one core.

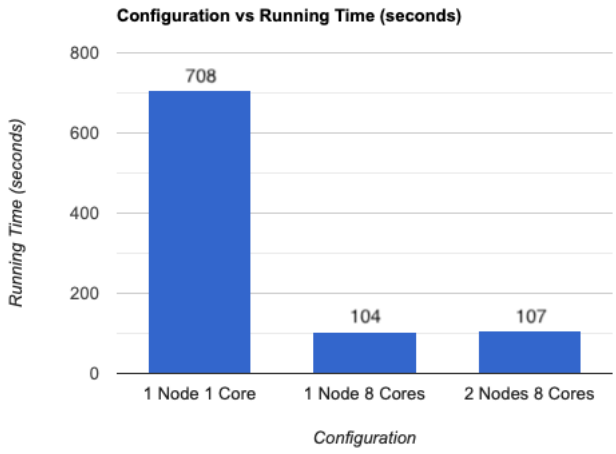

Figure 1: Configuration vs Running Time, in seconds

4 Conclusion

The application developed for this project utilises the divide and conquer paradigm by breaking a complex problem into several non-overlapping sub-problems, in which successfully process a large Twitter dataset using the HPC available at the University of Melbourne. In the process of completing this project, we gained valuable experience on using the `mpi4py` interface and the Slurm job queuing system.

References

[1] Spartan Documentation. (n.d.).
https://dashboard.hpc.unimelb.edu.au/

[2] Slurm Workload Manager - Documentation. (n.d.).
https://slurm.schedmd.com/

[3] MPI for Python — MPI for Python 3.1.4 documentation. (n.d.).
https://mpi4py.readthedocs.io/en/stable/