COMP90024 Social Media Analytics Report

1152502 Jongho Park, 1214235 Jiahe Liu

1. Introduction

In this project, our team implemented parallel computing on social media analysis through the University of Melbourne High Performance Computing (HPC) facility SPARTAN. This report will demonstrate the invoking procedure and code parallelising approach of our application. Additionally, it will introduce the data analysis algorithm and relevant assumptions. Moreover, this report will describe the performance variations on different numbers of nodes and discuss the potential reasons and limitations.

2. Invoking Procedure

1. Login into SPARTAN with ssh (either using password or key pair)
2. To install required modules, use `sbatch installing.slurm` command in the terminal.
3. In the home directory, create a ~/`script` directory to store `MPI_Series` scripts, and a ~/`data/raw` directory to store the bigTwitter.json and sal.json dataset.
4. To test different multiprocessor, use command `sbatch 1node1core.slurm` for 1-node-1-core, `sbatch 1node8cores.slurm` for 1-node-8-core, and `sbatch 2nodes8cores.slurm` for 2-node-8-core HPC system.
5. To view the current status of the job, use command `squeue -u [your user name]` or `squeue [job ID]`
6. To view the result and execution time, use command `less slurm-[job ID].out`

3. Code Parallelising Approach

`MPI_Series_1.py` demonstrated the parallel process with data reading, splitting, and creating buckets. To store the partitioned big data in each processor, we created a number of buckets which equal the number of processors. The big twitter data will be read row-by-row using ijson by each processor at the same time for memory efficiency. After reading one row of data, 'Author ID' and 'Location' will be extracted and sent to an assigned bucket. To decide which bucket the data will be sent to, we introduced the modulus hash function:

```
Author ID % Number of Cores = Bucket ID
```

This function will assign the Bucket ID for each Author ID, while ensuring that data with the same author ID remains in the same bucket. This leverages efficiency in ranking the top 10 authors. Since the same author ID will be sent to the same bucket, we can select the top 10 authors from each bucket and compare them altogether to generate the overall top-10 authors. Although the modulus hash function may divide the big data into uneven sizes and impact performance due to idling, this function ultimately improves overall efficiency by reducing transfer time. Additionally, we converted the csv format to parquet format for efficiency.

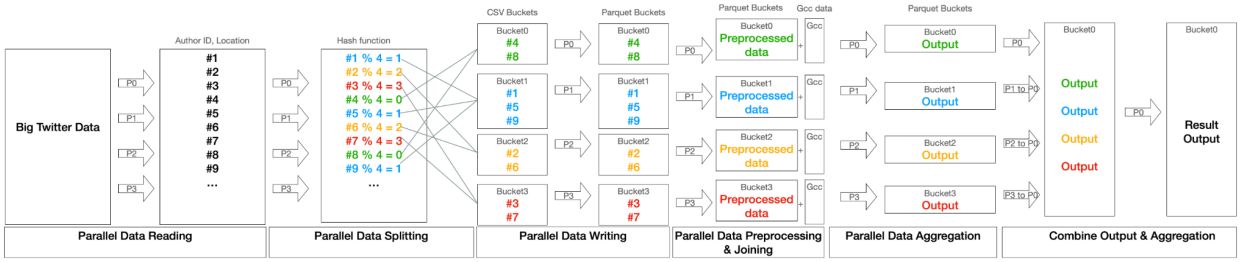

Figure 1. 4-Processor Parallel Data Reading, Splitting, Writing

3.1 Data Analysis Algorithm

`MPI_Series_2.py` demonstrates the paralleled preprocessing and merging algorithm. To reach the analysis goal, the location of each tweet will be connected to a standardised location index through joining the gcc dataset. These datasets can be merged as both of their locations can be presented with the form `"city, state"` after preprocessing.

3.1.1 Outlier Detection

We assumed that we will only be matching twitter's location in the form of *"city, state"* in Australia. Hence, we identified the four types of outliers: Non-Australian: `"Toronto, Ontario"`; State + Country: `"Victoria, Australia"`; Country only: `"Australia"`; City only: `"Barangaroo"`.

In total, 876,903 (5.76%) outliers were removed before joining. However, we discovered that many state capital cities will appear at the "state" position (e.g., "Greenwich, Sydney"). Also, other territories have a different "state" format (e.g., "Jervis Bay, Jervis Bay Territory"). We assumed that they are not outliers and applied a more flexible matching algorithm.

3.1.2 Preprocessing

In both datasets, some city names contain parentheses and hyphens which may interfere with the matching process. Hence, we performed preprocessing steps for both datasets. All strings will be transferred to lowercase and states will be mapped with its acronym.

A hyphen sign occurs to present the location in the form "smaller area - greater area, state". We kept the part after the hyphen (i.e., greater area) for both datasets to ignore the ambiguity that the same smaller area may occur in multiple greater areas. Parentheses may occur in "city" or "state" to indicate state suffixes. We will exclude the parentheses if it occurs in "city" and include the parentheses if it is in "state". Duplicated gcc data will be dropped to avoid rematch. However, some duplicates occurred due to the manipulation of hyphens and parentheses (i.e., "mount archer" and "spring creek"). Hence, we did some manual manipulation to resolve it.

3.1.3 Data Joining

To assign the gcc index to twitter data, we assume that the city and state of twitter's location has to be exactly the same as gcc's city and state. Hence, we firstly applied an inner join on two keys (city, state). We applied a more flexible joining strategy for state capital cities (Melbourne, Sydney, Brisbane etc.) and other territories with single key match, as a gcc index can be assigned when either twitter's city is matched with a gcc city or gcc state in these special cases.

For the 14,355,165 data that attended the joining process, 14,264,415 (99.37%) data were matched in total, where 14,162,748 (98.66%) data were matched by the first join and 101,667 (0.71%) data were matched by the second join.

3.2 Aggregation Process

The goal for this project is to show how many tweets are made by the author and Greater Capital City (GCC). To achieve this goal, In `MPI_Series_3.py`, we applied a divide-and-conquer strategy where each processor will aggregate their data first and send informative aggregated data back to the main processor for final aggregation.

To demonstrate how many tweets are made in each GCC, each processor will count the number of tweets in GCC and send the results to the main processor to generate the total counting results. To demonstrate the number of tweets by the top 10 authors who tweeted the most, each processor will rank and send the top 10 authors to the main processor for combined results, since the tweets with the same author ID will be assigned to the same processor through hash function. Based on the combined results, the main processor will perform another ranking to select the overall top 10 authors who tweeted the most. Similar divide-and-conquer approach was also applied to rank the author according to the number of GCC and the number of tweets they have made.

4. Results and Performance Discussion

Slurm scripts in Figure 2 generated results in Table 1 which suggests that Melbourne, greater Sydney, and greater Brisbane ranked the top 3 areas where most tweets were posted. Author 1429984556451389440 tweeted the most in number of tweets and number of unique cities.

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --time=0-12:00:00

module load python/3.7.4
module load mpi4py/3.0.2-timed-pingpong

mkdir ~/virtualenv
virtualenv ~/virtualenv/python3.7.4
source ~/virtualenv/python3.7.4/bin/activate
pip install pandas fastparquet ijson pyarrow

srun -n 3 python3 ./script/MPI_Series_1.py

##DO NOT ADD/EDIT BEYOND THIS LINE##
##Job monitor command to list the resource usage
my-job-stats -a -n -s
```

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=0-12:00:00

module load python/3.7.4
module load mpi4py/3.0.2-timed-pingpong
source ~/virtualenv/python3.7.4/bin/activate

mpiexec -n 1 python ./script/MPI_Series_1.py
mpiexec -n 1 python ./script/MPI_Series_2.py
mpiexec -n 1 python ./script/MPI_Series_3_q1.py
mpiexec -n 1 python ./script/MPI_Series_3_q2.py
mpiexec -n 1 python ./script/MPI_Series_3_q3.py

##DO NOT ADD/EDIT BEYOND THIS LINE##
##Job monitor command to list the resource usage
my-job-stats -a -n -s
```

```
#!/bin/bash
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=8
#SBATCH --time=0-12:00:00

module load python/3.7.4
module load mpi4py/3.0.2-timed-pingpong
source ~/virtualenv/python3.7.4/bin/activate

mpiexec -n 8 python ./script/MPI_Series_1.py
mpiexec -n 8 python ./script/MPI_Series_2.py
mpiexec -n 8 python ./script/MPI_Series_3_q1.py
mpiexec -n 8 python ./script/MPI_Series_3_q2.py
mpiexec -n 8 python ./script/MPI_Series_3_q3.py

##DO NOT ADD/EDIT BEYOND THIS LINE##
##Job monitor command to list the resource usage
my-job-stats -a -n -s
```

```
#!/bin/bash
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --time=0-12:00:00

module load python/3.7.4
module load mpi4py/3.0.2-timed-pingpong
source ~/virtualenv/python3.7.4/bin/activate

mpiexec -n 8 python ./script/MPI_Series_1.py
mpiexec -n 8 python ./script/MPI_Series_2.py
mpiexec -n 8 python ./script/MPI_Series_3_q1.py
mpiexec -n 8 python ./script/MPI_Series_3_q2.py
mpiexec -n 8 python ./script/MPI_Series_3_q3.py

##DO NOT ADD/EDIT BEYOND THIS LINE##
##Job monitor command to list the resource usage
my-job-stats -a -n -s
```

Figure 2 Slurm Script for Module Installation, 1-node-1 core, 1-node-8-core, and 2-node-8-core

As illustrated by Table 2, The execution time was 404, 345, and 328 seconds for 1-node-1-core, 1-node-8-core, and 2-node-8-core respectively. The 2-node-8-core HPC system delivered the highest speed of 328s which is 1.05 times faster than 1-node-8 core although they have the same number of cores. Reasons for this may be that processors in one node share limited resources including memory bandwidth, cache spaces, and I/O bandwidth. Hence more resources are allocated to each processor in 2-node-8-core, resulting in higher efficiency.

Q1: The number of tweets in the various capital cities

Greater Capital City	Number of Tweets Made
2gmel (Greater Melbourne)	2286891
1gsyd (Greater Sydney)	2218396
3gbri (Greater Brisbane)	859994
5gper (Greater Perth)	589322
4gade (Greater Adelaide)	465908
8acte (Australian Capital Territory)	202646
6ghob (Greater Hobart)	98816
7gdar (Greater Darwin)	46357
9oter (Other Territories)	182

Q2: Top 10 tweeters (in terms of the number of tweets made)

Rank	Author Id	Number of Tweets Made
#1	1498063511204761601	68477
#2	1089023364973219840	28128
#3	826332877457481728	27718
#4	1258331934242123776	25350
#5	1423662808311287813	21034
#6	1183144981252280322	20765
#7	1278672820792508417	20503
#8	820431428835885059	20063
#9	778785859030003712	19403
#10	1104295492433764353	18781

Q3: Tweeters that have tweeted in the most Greater Capital cities and the number of times they have tweeted from those locations

Rank	Author Id	Number of Unique City Locations and #Tweets
#1	1429984556451389440	8 (#1921 tweets - 1880gmel, 13acte, 11gsyd, 7gper, 6gbri, 2gade, 1gdar, 1ghob)
#2	17285408	8 (#1208 tweets - 1060gsyd, 60gmel, 40gbri, 23acte, 11ghob, 7gper, 4gdar, 3gade)
#3	702290044601692 16	8 (#1177 tweets - 327gsyd, 252gmel, 220gbri, 151gper, 119gade, 46acte, 42ghob, 20gdar)
#4	87188071	8 (#401 tweets - 116gsyd, 86gmel, 65gbri, 51gper, 34acte, 29gade, 15ghob, 5gdar)
#5	774694926135222272	8 (#272 tweets - 38gmel, 37gsyd, 37gbri, 36ghob, 34acte, 34gper, 28gdar, 28gade)
#6	1361519083	8 (#266 tweets - 193gdar, 36gmel, 18gsyd, 9gade, 6acte, 2ghob, 1gper, 1gbri)
#7	502381727	8 (#250 tweets - 214gmel, 10acte, 8ghob, 8gbri, 4gade, 3gper, 2gsyd, 1gdar)
#8	921197440885886977	8 (#208 tweets - 58gmel, 49gsyd, 37gbri, 28gper, 24gade, 7acte, 4ghob, 1gdar)
#9	601712763	8 (#146 tweets - 44gsyd, 39gmel, 19gade, 14gper, 11gbri, 10acte, 8ghob, 1gdar)
#10	2647302752	8 (#80 tweets - 32gbri, 16gmel, 13gsyd, 5ghob, 4acte, 4gper, 3gdar, 3gade)

Table 1 Analysis Results

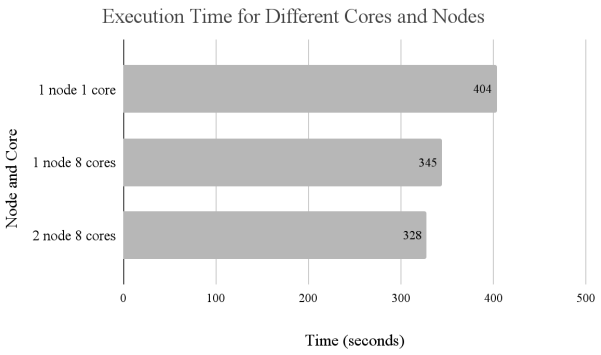

Table 2 Execution Time

Although the number of cores of 1-node-8-core is 8 times more than the number of cores of 1-node-1-core, the performance is not 8 times faster. This can be explained by Amdahl's Law (1967), as the potential speedup of a task is the serial part. Theoretically, the maximum speed up 1-node-8-core would be 5.47 times faster than 1-node-1-core where we assumed `MPI_Series_1` and `MPI_Series_2` to be parallel processes and `MPI_Series_3` to be a serial process.

$$Speedup = \frac{T(1)}{T(N)} = \frac{T_{serial(1core)} + T_{parallel(1core)}}{T_{serial(1core)} + \frac{T_{parallel(1core)}}{P}} = \frac{26.63 + 377.37}{26.63 + \frac{377.37}{8}} = 5.47$$

However, the actual speed up is only 1.17 times, which is less than the theoretical speed up. This is because Amdahl's Law assumptions may become invalid due to the lack of considerations in parallelisation overhead, memory access times, communication between processors, which can lead to non-linear relationships between the number of processors and performance.

Parallelisation overhead can arise primarily from communication and idling. For communication, each processor works on a partition of the task and has to communicate its results to the master processor. As the number of processors increases, it becomes more likely that the size of the partitioned data can fit into the cache memory of each processor, leading to shorter memory access times and communication time. Idling may occur due to workload imbalances, where some processors must wait for others that are dealing with longer tasks through synchronisation. Moreover, due to heterogeneity, each processor may exhibit different levels of efficiency, resulting in varying execution times.