COMP90024 Cluster Cloud Computing Assignment 1 Report

Junjie Xia
Student ID: 1045673
University of Melbourne
juxia@student.unimelb.edu.au

Tianqi Yu
Student ID:1221167
University of Melbourne
tyyu2@student.unimelb.edu.au

April 2, 2022

Abstract

This assignment is mainly focusing on using parallel computing to analyse the top 10 language in the twitter dataset, and compare the performance of different situations on spartan HPC, running with Slurm Scripts. Then, evaluating the performance of the cluster with different numbers of nodes and cores.

1 Introduction

Twitter is the most popular social media application around the world, with nearly billions of active users per month. Such massive data volume provides a great opportunity for people to excavate various research topics. For instance, different attitudes of the general public towards a sensitive event, predicting users' geographic location through the content of tweets. In this application, the main objective is to calculate the number of different languages used in 16 grid cells, then count the number of tweets in various languages, to demonstrate the multicultural atmosphere in Sydney. By comparing the time and resource consuming, revealing the enchantment of parallel computing.

Parallel computing is used in laboratory and industry because of the difficulty of the task and computational heavily[GLS14]. Therefore, in this report we are going to introduce the model that we implemented and analyse the performance of the model. Moreover, some improvement strategies that we are planning to apply in future.

2 Design Section

To accomplish these goals, HPC platform and parallel computing will be used due to the massive volume in three different circumstances, one node one core, one node eight cores, two nodes eight cores. Hence, the communication between multiple cores and nodes, and data processing will be at the first priority. In this application, Python is used to implement and run by Slurm management scripts, using *mpi4py* library to deal with parallel computing problem.

2.1 Data Analysis

Given the number of different languages, the top 10 most commonly used tweets should be counted. Therefore, a rational data analysis process should be established. First, the position of each grid and coordinate should be well considered. To verify whether a tweet is in a certain position or not. The coordinate of it, and each specific range of grid cell should be calculated very carefully.

This picture simply illustrates boundary conditions and standard to verify tweets' location. Basically, if one tweet is located on the top and right sides(green sides), then it belongs to this cell, else it belongs to the cell which shares this side. There is also some exception happened. If a tweet is precisely located on the low left corner of the cell, then let it count to the cell which is located next to the low left. The boundary condition is the blue point that is pointed in the picture. If tweets are beyond this point or others like this. Then just ignore this tweet and identify it as an illegal one.

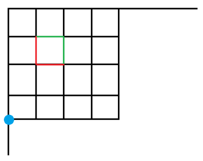

Figure 1: grid cell

Based on the coordinates and grid cells, calculate the number of languages and tweets. To do this, traversing all Json file is well needed. But considering the time complexity, *Dictionary* data structure is the most appropriate choice to search. So if a tweet is in one grid, then add it to the Dictionary which belongs to its grid. The *language_dict* feature in *json* file is combined with[twi22] and [Wik22] language tag, to decide which language belongs to. The method is very useful to allow us to process in the next steps.

2.2 Slurm

In this application, it is mainly submitted by slurm. Slurm not only can load the running environment which the program exactly needs, like Python, it also asks the program to which situation should it stand. For example, telling the program, which partition it will use, how many node/task it will occupy. Considering the actual circumstance, multi-process is the question we should concern, instead of multi-thread. However, context switch of multi-process consumes lots of time and resources. Hence, we assume that core is the same thing as process. And in the same time, there is only one process operating on one node.

2.3 Parallel Strategies

The reason why parallel strategies are quite crucial is very simple. The volume of *bigTwitter.json* is too large. If processing it directly, time and memory consuming are unacceptable. Not to mention that directly reading this huge dataset into memory is impossible. Therefore, choosing wise and suitable strategies is very important.

2.3.1 One Node One Core

The first model is one node one core system. We attempted several methods. Firstly, we attempted to implement a model which could distinguish all the data in one grid. Then, iterating until all the grids are calculated. Finally, there is a function that could read the result and generate the pandas' table. In terms of the second model, we implemented a system which could analyze one-line data instead of all the files. Therefore, the system will not crash when it encounters a big file.

```
#Read whole file at one time

read all the data in grid json file
read all the data in twitter json file
for all twitter data
    if the coordinate value of one line of twitter data is not null
        add this line to the filtered list

initial a grid dictionary
for all grids
    store the grid id in specific grid coordinates key

initial a list which store the result
for all grids coordinates in
    initial a twitter dictionary
    for all lines of twitter data
        if the coordinate in this grid
        the number of this language in dictionary add 1
    sort the dictionary by appeared times
    store the result of this grid in result list

print the result
```

```
#Read data line by line

read all the data in grid json file
initial a grid dictionary
for all grids
    store the grid id in specific grid coordinates key

for all lines in tweet data set
    if this lines coordinate is not null and coordinate within the grid
        analysing the language used in this line and the grid id that line located
print the result in required format
```

In the first line of *bigTwitter.json*, it shows the number of lines which need to be processed. Meanwhile, the last line of *bigTwitter.json* is slightly different from *smallTwitter.json*. It should be treated as an exception to process

```
# First line
{"total_rows":215443567,"offset":211386044,"rows":[
#Last line
]}
```

Generally speaking, reading and processing data line by line instead of processing them as a whole. At the end, gathering all processed data in the *language_dict*

2.3.2 Multiple Cores

In terms of one node eight core model, we used the *MPI* function in the *MPI4PY* package which is a *MPI* library for python. In this model, we used some paralleling strategies. Firstly, we used the grid analysing part, result printing part and one line reading part in analysing data line by line function that we implemented before. Then, we implemented the tweets processing function which could split the data into 8 parts and each core processing one part. Next, all the data would be generated in the main core, and use the result print function to process all the data that was analysed by all the cores. Finally, use the pandas DataFrame to print a table of the result.

```
# Pseudocode of the 8 cores

read all the data in grid file
initial an dictionary
for all the data in grid file
    store the grid id and according grid coordinate in dictionary

for all the data in twitter file
    if the line number mod number of cores equals to the core number
        If this line of the data contains coordinates and in the grid area
            Add to the list
Combining the data each core generated
For all the data from the former list
    Add the coordinate and language to the specific grid id key in dictionary
Analysing the data to the required format
Print the result
```

The multiple cores section is implemented on the One Node One Core section, specifically keeping the reading and processing data line by line. To verify which core a certain line should be sent to, we use $row\_number\%process\_size == rank$. In this part, master-slave model is deployed, slave core are processing the data from the file, at the mean time master core is also responsible for processing part of data, same as other slave cores. Each core initialise a list, and it would append the new line of data to the list. Firstly, master will send eight lines data to itself and other slave cores. Once these cores receive, processing data in local. Then master will repeat this step, until all data has been processed. Overall, each core processes $\frac{1}{8}$ of dataset. The advantage is obvious, this strategy divides a massive data part into many small parts, it will enhance the performance and speed. The disadvantage is, the master core undertake too many processing mission, it will definitely cost lots of time and resources.

2.4 Optimization

In this part we will illustrate some strategies which could be used to optimize the whole system. Furthermore, there will be some evaluations of the model that we implemented.

2.4.1 Time & Resource Comparison

In terms of Time and Resource Comparison, the 1 node 1 core model uses more time than 1 node 8 cores and 2 node 8 cores. Because more processors tend to have more computing power and much faster to process the data. However, more processor means requires more cores to complete this task, which may take extra resources. For instance, in the second model, we used 8 cores to do this task. By contrast, in the first model we used 1 core to do this task. Moreover, the communication between different cores will cost extra more time and resources too.

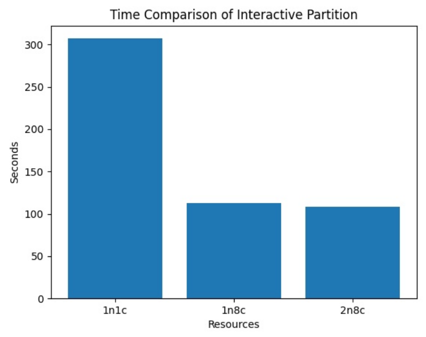

Figure 2: Time Comparison of Interactive Partition.jpg

2.4.2 Evaluation and Optimization Strategy

In terms of evaluation, one node eight cores almost spends the same amount of time as two nodes eight cores, which may because differences in performances across nodes. Hence, two nodes eight cores runs on the high performance core, but one node eight cores model runs on the low performance core. Moreover, 1 node 8 cores and 2 node 8 cores runs almost 4 time faster than 1 node 1 core. This result shows the MPI could enhance the efficiency of the data processing. More cores performance better than less, it largely demonstrates the correctness of the Amdahl's law.

We use the MPI function in multiple cores model and it does perform better than one node one core. However, this model has several weaknesses. Firstly, this model requires processing the output data from eight ranks, which may take $O(n)$ time. Therefore, we could use some strategies to improve it. Secondly, in this model we used eight cores, but increasing the number of cores could also improve the performance.

3 Conclusion

In conclusion, these project is a good way to know more about MPI programming. we used read line by line and $MPI4PY$ package to solve analysing a great amount of data task. Furthermore, in this report we mentions some strategies to improve the model and evaluate the performance of the models.

References

[GLS14] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI portable parallel programming with the message-passing interface*. The MIT Press, 2014.

[twi22] Supported languages and browsers of twitter developer platform. https://developer.twitter.com/en/docs/twitter-for-websites/supported-languages, 2022.

[Wik22] Wikimedia Foundation. Ietf language tag. https://en.wikipedia.org/wiki/IETF_language_tag, 2022.

4 Appendix

4.1 One Node One Core Script

```
#!/bin/bash
#SBATCH --partition=interactive
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=0-00:10:00
#SBATCH --output=1n1c.out
#SBATCH --error=1n1c.err

# Load required modules

module load gcccore/8.3.0
module load foss/2019b
module load python/3.7.4
module load numpy/1.18.0-python-3.7.4


time srun -n 1 python3 ass1.py
echo "Completed!...................................."
```

4.2 One Node Eight Cores

```
#!/bin/bash
#SBATCH --partition=interactive
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=8
#SBATCH --time=0-00:05:00
#SBATCH --output=1n8c.out
#SBATCH --error=1n8c.err

# Load required modules

module load gcccore/8.3.0
module load foss/2019b
module load python/3.7.4
module load numpy/1.18.0-python-3.7.4


time srun -n 8 python3 ass1.py
echo "Completed!...................................."
```

4.3 Two Nodes and Eight Cores

```
#!/bin/bash
#SBATCH --partition=interactive
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --time=0-00:05:00
#SBATCH --output=2n8c.out
#SBATCH --error=2n8c.err

# Load required modules

module load gcccore/8.3.0
module load foss/2019b
module load python/3.7.4
module load numpy/1.18.0-python-3.7.4


time srun -n 8 python3 ass1.py
echo "Completed!..................................."
```