The University of Melbourne

COMP90024 Cluster and Cloud Computing
2020 Semester 1
Assignment 1

HPC Twitter Processing

Yawei Sun	1050317	yaweis@student.unimelb.edu.au
Rongbing Shan	945388	rashan@student.unimelb.edu.au

1. Introduction

Due to rapidly grow use of twitters, it becomes harder to extract information from Twitter. Parallel computing becomes an essential part in processing data in large volume such as twitters. The purpose of this assignment is to identify the languages which are most commonly used and top 10 hashtags which are most frequently occurring in a large twitter file through high performance computing. In this report, we will explain in detail how we extract the desired information from large twitter file though MPI, and the performance by executing the program on Spartan, an HPC platform by the University of Melbourne.

Tools and platforms:

- SPARTAN: the HPC platform hold by The university of Melbourne.
- Python and MPI4py: The program is coded with python and MPI4py is used to implement parallel computing.

2. Implementation

The program aims to find the top 10 hashtags in the twitters, also, the top 10 most frequent used languages of the twitter. In this section, we will explain how we extract such information from the twitter and how the program rank hashtags and languages in parallel.

Data extraction algorithms

Hashtags will be extracted from Json -> '**doc**' -> '**entities**' -> '**hashtags**'. Here we will consider only the hashtags in actual tweets that are in the file. For a tweet 'b' that is a retweet of some other tweet 'a', we will not add up the hashtags that is in original tweet a (which are located in 'retweet_status'->'entities'), since we would not want to count those source tweets multiple times.

Language codes for the twitter will be extracted from Json -> '**doc**' -> '**metadata**' -> '**iso_language_code**' and then the language code will be mapped to real languages based on twitter's developer guide on supported language codes.

Steps of processing

In this part, we will explain the steps of processing the twitter file in parallel computing method with MPI.

1) Reading file and split data across nodes
   In our implementation, all nodes will open the file separately with readonly mode and get its own allocated data based on line number. This is because of

that, in our json file, each line contains exactly one twitter data block, except for the starting and ending line. As demonstrated in figure 1, we will split data based on the remainder of line number (*line_num*) divided by total number of nodes(*comm_size*), and check if it equals node number (*comm_rank*).
Based upon this, if there are 4 nodes in total, node 0 will process line 4,8,16,… and node 1 will process line 1, 5, 9, etc.

```
for line in file0:
    line_num = line_num + 1
    # divide the file to be processed by different node
    # the data is split to each node by remainders of line number divided by node numbers
    if(comm_rank == line_num % comm_size):
```

Figure1. data partitions based on line number

Then each node will process its allocated data, and return two dictionaries, one for hashtags with number of appearances and another one for languages with number of appearances.

2) Master node data collection

We acknowledge the node rank 0 as the master node.
After all nodes have finished processed their data, master node will gather all the dictionaries from each node and merge them into two complete dictionaries, containing data from all twitters in the file.

```
# gather results from all nodes to master node
all_hashtags = comm.gather(hashtags, root=0)
all_language = comm.gather(language, root=0)
```

Figure2. master node gather data

Then both dictionaries will be sorted based on number of appearances and return the top 10, as the target for the program.

```
# master node process all results and find top 10 hashtags and languages
if comm_rank == 0:
    # merge all dicitionaries from each node to single one
    final_hashtags = merge_dict(all_hashtags)
    final_language = merge_dict(all_language)
    # sorted the reusult by appearances of hashtags and languages
    sorted_hashtags = sorted(final_hashtags, key=final_hashtags.get, reverse=True)[0:10]
    sorted_language = sorted(final_language, key=final_language.get, reverse=True)[0:10]
```

Figure3. Find top 10 results

3. Execution on SPARTAN

The program will be run on HPC platform SPARTAN for three different settings: 1 node 1 core; 1 node 8 cores; and 2 nodes 8 cores. We will submit the job to SPARTAN through slurm files, an example is shown in figure 4.

```
#!/bin/bash
#SBATCH -p physical
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --time=00:30:00
#SBATCH --output=n2c8.out
#SBATCH --error=err_n2c8.out

echo ' '
echo 'Run with 2 node 8 cores'

module purge
module load Python/3.6.4-intel-2017.u2-GCC-6.2.0-CUDA9
mpiexec python rankTwitter.py -f bigTwitter.json
```

Figure4.
2 nodes 8 cores slurm job

4. Evaluation and Conclusion

In this section, we will discuss the execution results and lessons learned.

Evaluation

The execution results for the three categories are listed below:

```
Run with 1 node 1 core
Top hashtags:
 1. #auspol, 19878
 2. #coronavirus, 10110
 3. #มาพ่องเพิ่งอะไร, 7531
 4. #firefightaustralia, 6812
 5. #oldme, 6418
 6. #sydney, 6196
 7. #scottyfrommarketing, 5185
 8. #grammys, 5085
 9. #assange, 4689
10. #sportsrorts, 4516
------------------------------
Top languages:
 1. English(en), 3107115
 2. Undefined(und), 252117
 3. Thai(th), 134571
 4. Portuguese(pt), 125858
 5. Spanish(es), 74028
 6. Japanese(ja), 49929
 7. Tagalog(tl), 44560
 8. Indonesian(in), 42296
 9. French(fr), 38098
10. Arabic(ar), 24501

Execution time is 540.84432 seconds
```

```
Run with 1 node 8 cores
Top hashtags:
 1. #auspol, 19878
 2. #coronavirus, 10110
 3. #มาพ่องเพิ่งอะไร, 7531
 4. #firefightaustralia, 6812
 5. #oldme, 6418
 6. #sydney, 6196
 7. #scottyfrommarketing, 5185
 8. #grammys, 5085
 9. #assange, 4689
10. #sportsrorts, 4516
------------------------------
Top languages:
 1. English(en), 3107115
 2. Undefined(und), 252117
 3. Thai(th), 134571
 4. Portuguese(pt), 125858
 5. Spanish(es), 74028
 6. Japanese(ja), 49929
 7. Tagalog(tl), 44560
 8. Indonesian(in), 42296
 9. French(fr), 38098
10. Arabic(ar), 24501

Execution time is 128.20838 seconds
```

```
Run with 2 node 8 cores
Top hashtags:
 1. #auspol, 19878
 2. #coronavirus, 10110
 3. #มาพ่องเพิ่งอะไร, 7531
 4. #firefightaustralia, 6812
 5. #oldme, 6418
 6. #sydney, 6196
 7. #scottyfrommarketing, 5185
 8. #grammys, 5085
 9. #assange, 4689
10. #sportsrorts, 4516
------------------------------
Top languages:
 1. English(en), 3107115
 2. Undefined(und), 252117
 3. Thai(th), 134571
 4. Portuguese(pt), 125858
 5. Spanish(es), 74028
 6. Japanese(ja), 49929
 7. Tagalog(tl), 44560
 8. Indonesian(in), 42296
 9. French(fr), 38098
10. Arabic(ar), 24501

Execution time is 130.96161 seconds
```

Figure5. Outputs by different settings

The running time for three different settings are:

- 1 node 1 core: **541** seconds
- 1 node 8 cores: **128** seconds
- 2 nodes 8 cores: **131** seconds

For a better illustration, we put the results in a bar chart as in figure 6.

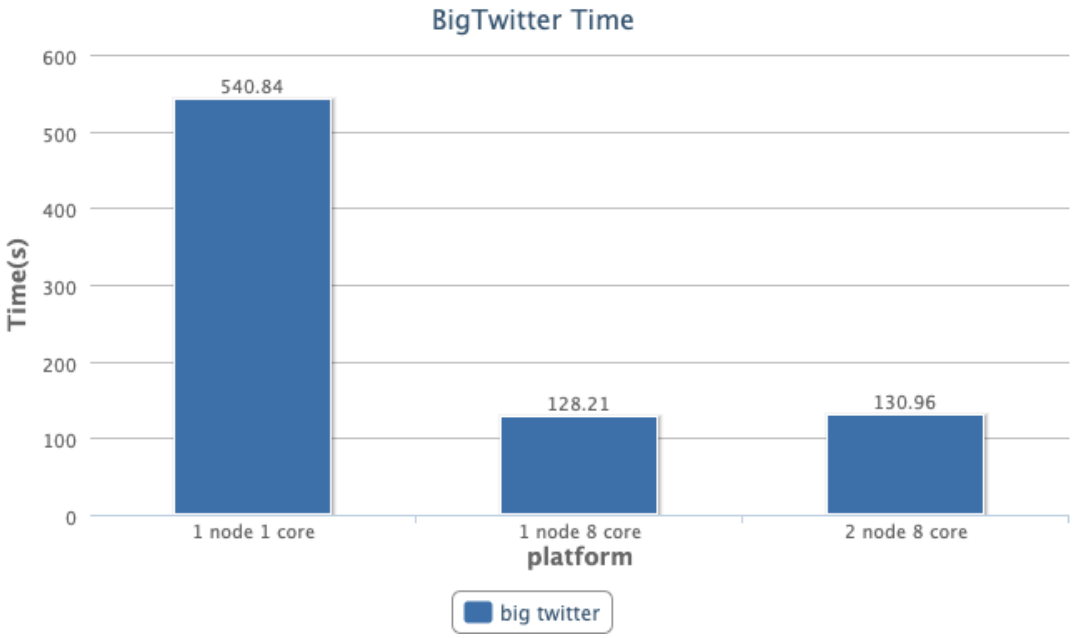

Figure 6. Execution time for different settings

Conclusion

It can be found that, compared to 1 node 1 core, parallel computing can improve the performance by a large amount, reducing the running time by about 3-5 times. This is because all worker nodes can read and process data at the same time. In other words, we have divided a large problem into multiple smaller ones, hence utilizing the calculation power for each worker nodes, and this is the advantage for parallel computing. In such way, all resources can be used effectively.

However, the performance on 2 nodes 8 cores is not good as it on 1 node 8 core. We believe it is caused by the communication between two nodes, where accessing the memories in another node takes more time then accessing one node's own memory.

In general, from the results, we can prove that parallel computing saves time, given a better performance for complicated problems.