COMP 90024 - Assignment 1 HPC Twitter GeoProcessing

Jing Du 775074 du2@student.unimelb.edu.au
Zhijia Lu 921715 zhijial@student.unimelb.edu.au

This report is written to deliver the tools and ideas behind assignment 1 and furthermore evaluate the results obtained by the experiment.

1. Introduction

The main purpose of assignment 1 is to implement a parallel computation over HPC facility, which ranks the grid boxes based on relative twitter number and lists hashtags in each grid box in Melbourne with ordering five hashtags in each grid. The datasets and run environment are demonstrated in *Table1:*

Datasets	melbGrid.json
	bigTwitter.json
Programming Language	Python3
Parallel programming standard	MPI
Computation Platform	SPARTAN (University of Melbourne)

Table1

2. Methodology

2.1 Single Program Multiple Data (SPMD)

SPMD is a technique designed to achieve parallelism. Tasks are split up and run simultaneously on multiple processors with different input in order to obtain results faster. SPMD is the most common style of parallel programming. The programming model of SPMD is shown below:

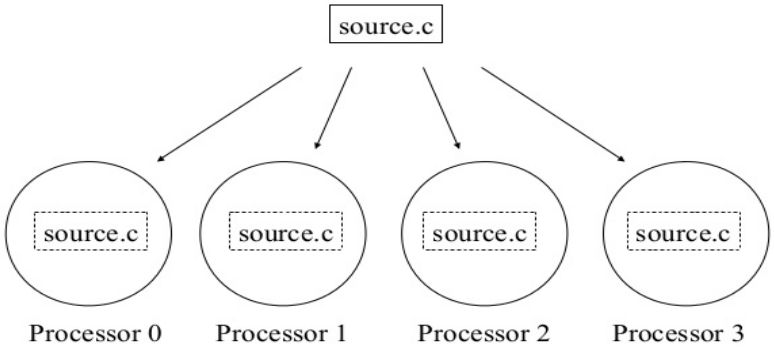

Fig.1: SPMD Programming Model

2.2 Message Passing Interface (MPI)

MPI, a standard of parallel programming for message delivering, is chosen to constrict computing. It is generally used to build highly reliable, scalable and flexible distributed applications because of its well-designed scalability and complete asynchronous communication function. Basic functions invoked in the assignment are shown in *Table2:*

Basic function	Function meaning
MPI_Init	initialize

MPI_Gather	Gather message to the root node
MPI_Comm_size	Get the number of processes
MPI_Comm_rank	Gets the process id
MPI_Bcast	Broadcast message

Table2: MPI basic functions

2.3 Bash Script

In order to guide HPC to execute our tasks, Slurm (Simple Linux Utility for Resource Management) as a workload manager will allocate resources (computer nodes) to each job by the specific batch script. The parameters used for this assignment are explained as follow.

[--partition=<partition_names>] requests a specific partition for resource allocation. Cloud and Physical as two options can be chosen

[--time=<time>] specifies the run time wall of each job allocation. This parameter is associated with the priority, which means a shorter time limit leads to a higher priority.

[--nodes=<minnodes[-maxnodes]>] accepts a minimum or a maximum number of nodes to be implemented in the job. If only one number is specified, the nodes in this number will be allocated.

[--ntasks-per-node=<ntasks>] will invoke ntasks on each node.

[--output=<filename pattern>] defines a file to record both standard output and standard error.

By applying the optional parameters, the job can be executed under different environments, and the performances associated with any modifying parameter can be observed directly.

3. Program Design

3.1 Program Framework

There are four basic classes in this program: Grid, Twitter, Mpi, and Utility. The UML diagram of this program is shown in *Fig.2,* where **Grid** and **Twitter** provide basic data structures storing the information for grid boxes and helpful attributions in each twitter. **Mpi** enables communication among cores based on the functions in the module 'mpi4py' and **Utility** provides the methods to process dataset.

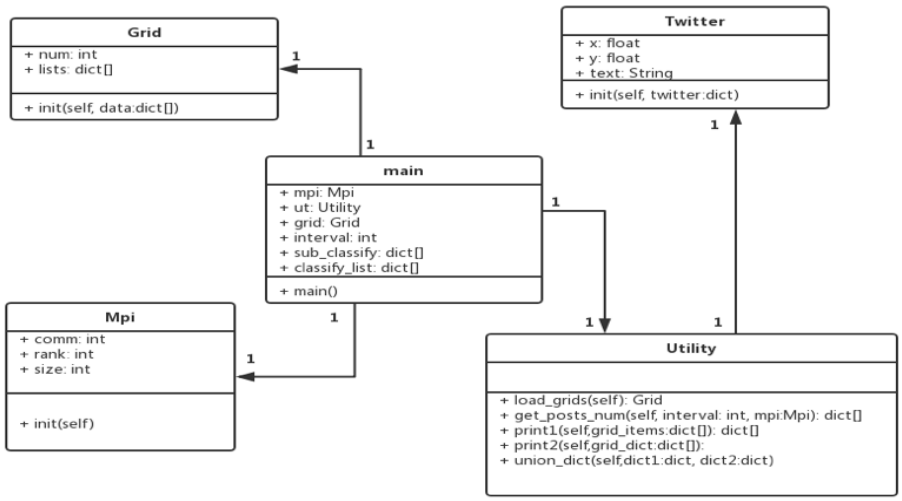

Fig.2: UML

When the program starts running, all nodes will create objects of Mpi and Utility for further communication and calculation.

3.2 Parallel task assignment

As the basic tool to drive a parallel algorithm, MPI plays a vital role in this assignment. We tried two methods: processing Twitters with partition reading and scroll reading.

Partition method equally split all Twitters based on the number of parallel processes. The root process calculates the number of rows of twitter and gets the number of processes by calling Mpi function *MPI_Comm_size*. Then divide the number of rows by the number of processes to get the size of the twitter block that each process needs to deal with. After that, *MPI_Comm_bcast* is called to get each process informed. After obtaining the size of the twitters partition block, each process calculates its task interval and processes them individually.

```
def get_posts_num(interval, current_rank, size):
    line_index = 1
    start_index = current_rank * interval
    end_index = (current_rank + 1) * interval
```

For scroll method, all processes take turns reading and processing twitters. Each process decides whether to work by comparing process ID with the remainder of the row index and the total number of processes.

```
def get_posts_num(self, mpi):
    line_index = 0
    with open("bigTwitter.json", 'r', encoding='utf-8') as load_f2:
        for twitter_l in load_f2:
            line_index += 1
            if mpi.rank == line_index % mpi.size:
                try:
                    # Line structures are different between the last line and others
                    if twitter_l[-2:-1] == ',':
                        twitter_d = json.loads(twitter_l[:-2])
                    else:
                        twitter_d = json.loads(twitter_l[:-1])
```

3.3 Hashtags Matching

A valid hashtag is defined as a term with structure [<Space>#<String><Space>], but the 2nd hashtag followed by another hashtag like [<1st hashtag> <2nd hashtag>] always be neglected because of a shared space. For retrieving neglected hashtags, intersection operation for two result lists from matching structure[<Space>#<String>] and [#<String><Space>] is applied.

```
# Hash tags are defined as with the structure of'<Space><String><Space>'
tag1 = re.findall(r'\s#\w+', twitter.text)
tag2 = re.findall(r'#\w+\s', twitter.text)
tag3 = [t[1:] for t in tag1]
tag4 = [t[:-1] for t in tag2]
tag = list(set(tag3).intersection(tag4))
```

4. Performance Evaluation

4.1 Result

From the running results, the results of single-process processing and multi-process processing are consistent, which indicates that parallelism does not affect the results of the program operation.

```
C2: 145096 posts, ((#melbourne, 10385),(#australia, 1534),(#job, 1390),(#jobs, 964),(#hiring, 469))
B2: 79633 posts,  ((#melbourne, 840),(#australia, 156),(#auspol, 148),(#brunswick, 94),(#fitzroy, 88))
C3: 53299 posts,  ((#melbourne, 488),(#australia, 144),(#coffee, 102),(#oneorigin_specialtycoffee, 71),(#food, 70))
B3: 26550 posts,  ((#melbourne, 168),(#australia, 60),(#fitness, 54),(#pocus, 50),(#fashion, 41),(#health, 41))
C4: 19699 posts,  ((#alishas1million, 96),(#vote5sos, 85),(#melbourne, 82),(#auspol, 75),(#savedallas, 45))
B1: 16507 posts,  ((#melbourne, 176),(#kca, 72),(#australia, 60),(#bds, 49),(#vote5sos, 43))
D4: 14497 posts,  ((#nowplaying, 1730),(#melbourne, 30),(#job, 26),(#jobs, 22),(#smallzyvampsfirsttix, 20))
D3: 13388 posts,  ((#melbourne, 80),(#beach, 59),(#summer, 29),(#australia, 23),(#followmenash, 21))
C1: 8434 posts,   ((#camto4mill, 72),(#christmasfollowparty, 62),(#jamesyamm, 58),(#askbeau, 53),(#melbourne, 52))
B4: 5450 posts,   ((#melbourne, 33),(#smallzywelcomeshome5sos, 26),(#fire, 19),(#vicfires, 19),(#socialmedia, 18))
A3: 5047 posts,   ((#vicfires, 19),(#saveconstantine, 13),(#girls, 10),(#love, 8),(#raw, 8))
A2: 4165 posts,   ((#krazykristmas, 165),(#egsholidaygiveaway, 57),(#rcl1milliongiveaway, 53),(#vdaywithbeth, 37),(#egholidaygiveaway, 31))
C5: 4106 posts,   ((#melbourne, 53),(#australia, 17),(#dandenongs, 14),(#travel, 13),(#1000steps, 12))
D5: 3248 posts,   ((#somebodytocon, 23),(#melbourne, 22),(#lost, 20),(#voteukvampettes, 18),(#findjasper, 12),(#kca, 12))
A1: 2534 posts,   ((#melbourne, 14),(#tattoo, 11),(#worldjudo2014, 9),(#judo, 8),(#design, 8),(#my, 8),(#vobis, 8),(#drawing, 8))
A4: 310 posts     ((#lnp, 4),(#vicfires, 3),(#qldvotes, 3),(#laughterclass, 2),(#joy, 2),(#mytribe, 2),(#market, 2),(#community, 2),(#birthday, 2))
```

4.2 Partition VS Scroll

The codes are invoked on different resources: 1 node 1 core, 1 node 8 cores, and 2 nodes 8 cores (4 cores per node) with time consumption in *Fig 3*. Compared to the processing with partition reading, scroll

reading method does not need to get the total number of rows of all twitters, thus reducing the running time of the program.

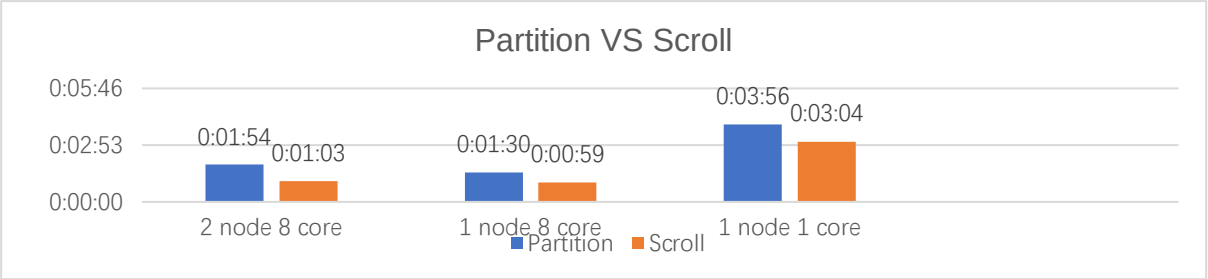

Fig.3: Partition VS Scroll

4.3 1core VS 8 core

Under the same amount of node, with the number of cores increasing from 1 to 8, the system processes faster significantly. With only one core, all the tasks are undertaken by one root process serially, which takes a considerable amount of time. On the other hand, multiple cores can alleviate the burden of root core, in which the run time decreases apparently. However, compared with 1core, the running time of 8core does not decrease by a corresponding multiple. This can be explained by Amdahl's law, which states that the parallel acceleration rate cannot exceed the number of parallels.

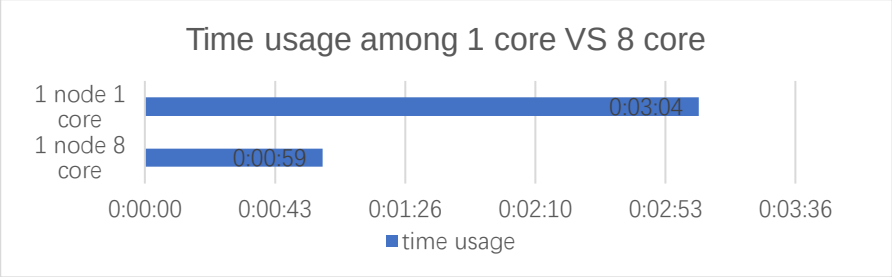

Fig.4: 1core VS 8 core

4.4 1 node VS 2 node

However, within the same number of cores, the time usage decreases along with the growth of node quantity. Specifically speaking, in the HPC system, communication on the same node is conducted in memory whereas it is achieved by Ethernet on different nodes. Therefore, marshaling and unmarshaling of messages and transmission delay may affect time efficiency.

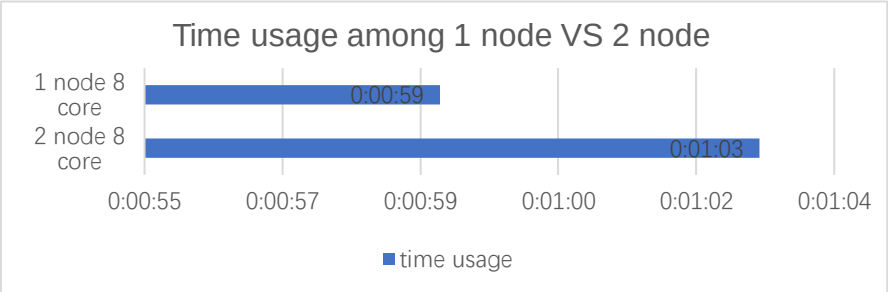

Fig.5: 1 node VS 2 node

In conclusion, parallel computing literally improves calculation efficiency by invoking more resources. However, since only three experiments are conducted in this assignment, the performance variation under different parameters should still be examined further.