COMP90024 Cluster and Cloud Computing
Project 1 Report

STUDENT NAME: Yunlu Wen, Yiming Zhang

1. Introduction

The assignment task is to implement a parallelized application leveraging the University of Melbourne HPC facility SPARTAN. The purpose of the application is to identify twitter usage from a large geocoded Twitter dataset. In this report, it will first describe how to submit a job to SPARTAN. Then it will briefly explain the parallel methodologies that are used in the application. Finally, the results of the performance of the application on the different number of nodes and cores will be discussed.

2. SPARTAN Scripts

To create and submit a job to SPARTAN, the SLURM scripts should be written. There are three scripts for "1 node 1 core", "1 node 8 cores" and "2 nodes 8 cores" respectively. The script specifies the partition of SPARTAN, the number of nodes, the number of cores, the number of cores per node and the maximum wall time. In addition, it specifies the modules that need to be loaded, the analysis code (ccc1.py) and the dataset (melbGrid.json and bigTwitter.json).

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

# Load required modules
module load Python/3.5.2-intel-2016.u3

# Launch multiple process python code
echo "1 node 1 core big twitter dataset"
time mpirun -np 1 python ccc1.py /home/yimingz8/melbGrid.json /home/yimingz8/bigTwitter.json
```

Figure 1: SLURM Script "1 node 1 core"

```
#!/bin/bash
#SBATCH -p physical
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=8
#SBATCH --time=0-12:00:00

# Load required modules
module load Python/3.5.2-intel-2016.u3

# Launch multiple process python code
echo "1 node 8 cores big twitter dataset"
time mpirun -np 8 python ccc1.py /home/yimingz8/melbGrid.json /home/yimingz8/bigTwitter.json
```

Figure 2: SLURM Script "1 node 8 cores"

```
#!/bin/bash
#SBATCH -p physical
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --time=0-12:00:00

# Load required modules
module load Python/3.5.2-intel-2016.u3

# Launch multiple process python code
echo "2 nodes 8 cores big twitter dataset"
time mpirun -np 8 python ccc1.py /home/yimingz8/melbGrid.json /home/yimingz8/bigTwitter.json
```

Figure 3: SLURM Script “2 nodes 8 cores”

3. Parallel Methodologies

3.1 Preprocessing

The first step of the application is the pre-process of the bigTwitter.json dataset. In order to do it, we wrote the “data-feeder” function which can process the JSON file into buffers and then we partitioned the data and only saved partial useful data such as coordinate, hashtags into a list. The buffer size is set to 10000. The design of this is to avoid load the whole file in the master node, instead, it will process it in chunks, and it is more efficient.

```
def data_feeder(file_path):
    fp = open(file_path, mode='r')
    # chunk = [None] * BUFFER_SIZE
    fp.readline()
    while True:
        try:
            chunk = [None] * BUFFER_SIZE
            for i in range(BUFFER_SIZE):
                # next_item = next(new_items)

                next_item = json.loads(fp.readline().replace(',\n', '\n'))
                if not next_item['doc']['retweeted']:
                    chunk[i] = (list(map(hash_proc, next_item['doc']['entities']['hashtags'])),
                                next_item['doc']['coordinates']['coordinates'],
                                1)
                else:
                    chunk[i] = None
                # ([tag1, tag2, tag3...], [x, y], count)
```

Figure 4: Data Preprocessing

3.2 Master-Worker Model

The application applies master-worker model. Master decomposes the problem into small tasks, distributes to workers and then gathers partial results to produce the final result. More specifically, we use scatter and gather with MPI using MPI4py and python. MPI_Scatter is a process that the master node sends the chunks of data to all worker nodes. In this application, after the master node reads a batch of data, scatter will divide the data into pieces and sends them to the worker nodes. The worker nodes will count the number of tweets and hashtags. Then MPI_Gather takes the results from workers and gathers them to one result.

```
while True:
    data = comm.scatter(batch, root=0)

    if data == FLAGS.END_OF_ITERATION:
        break
    reduced_dict = reduce(data)
    comm.gather(reduced_dict, root=0)
```

Figure 5: Scatter and Gather

4. Experiments and Results

The output of the application is in the appendix. It briefly shows that C2 area has the largest number of tweets (137594 posts) and the top hashtags in this area are: '#melbourne', '#job', '#australia', '#jobs' and '#hiring'.

Three experiments are taken to compares the performance of the application on different numbers of nodes and cores. The results of the time are shown at the table and chart. From the result, it shows that '1 node 8 cores' and '2 nodes and 8 cores' is has the similar result, and '1 node 1 core' has the worst performance.

Resources	Real Time(s)
1 node 1 core	209.802
1 node 8 cores	196.553
2 node 8 cores	195.242

Table 1: Time Performance

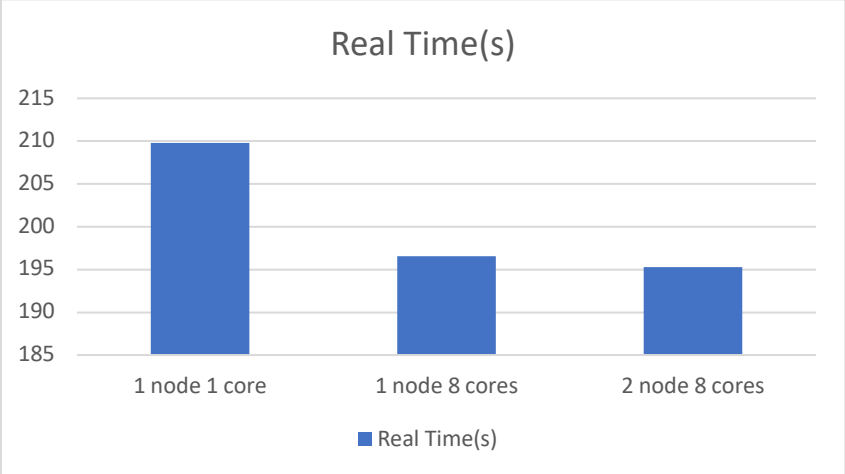

Figure 6: Time Performance Chart Bar

5. Conclusion

To conclude, the report firstly explains how to submit the job to SPARTAN, then discusses the programming details such as JSON preprocessing, master worker model and scatter, gather in python MPI4py. Finally, the performance of the application on different numbers of nodes and cores are compared. The result shows that the '1 node 8 cores' and '2 nodes 8 cores' has a better performance than '1 node 1 core' and '1 node 8 cores', '2 nodes 8 cores' have the similar performance. The reason is that parallel programming is more effective as it could run different processes at the same time.

6. Appendix

```
[yimingz8@spartan-login1 CCC1]$ cat slurm-7826315.out
2 nodes 8 cores big twitter dataset
C2 :  145096 posts
B2 :  79633 posts
C3 :  53299 posts
B3 :  26550 posts
C4 :  19699 posts
B1 :  16507 posts
D4 :  14497 posts
D3 :  13388 posts
C1 :  8427 posts
B4 :  5450 posts
A3 :  5036 posts
A2 :  4165 posts
C5 :  4106 posts
D5 :  3248 posts
A1 :  2533 posts
A4 :  310 posts
C2 :  [('#melbourne', 13448), ('#job', 2127), ('#australia', 1781), ('#jobs', 1300), ('#hiring', 986)]
B2 :  [('#melbourne', 1027), ('#auspol', 354), ('#incident', 314), ('#victraffic', 239), ('#firealarm', 237)]
C3 :  [('#melbourne', 612), ('#incident', 317), ('#victraffic', 297), ('#firealarm', 232), ('#structurefire', 219)]
B3 :  [('#incident', 222), ('#melbourne', 213), ('#structurefire', 177), ('#nonstructurefire', 159), ('#firealarm', 158)]
C4 :  [('#auspol', 279), ('#incident', 195), ('#structurefire', 179), ('#savedallas', 166), ('#victraffic', 144)]
B1 :  [('#melbourne', 226), ('#structurefire', 134), ('#nonstructurefire', 128), ('#incident', 111), ('#vote5sos', 106)]
D4 :  [('#nowplaying', 1730), ('#karishmatannacraze', 83), ('#victraffic', 79), ('#structurefire', 70), ('#noblepark', 69)]
D3 :  [('#melbourne', 99), ('#incident', 67), ('#beach', 66), ('#victraffic', 63), ('#structurefire', 62)]
C1 :  [('#incident', 112), ('#structurefire', 98), ('#camto4mill', 73), ('#melbourne', 69), ('#christmasfollowparty', 62)]
B4 :  [('#melbourne', 59), ('#nonstructurefire', 55), ('#incident', 49), ('#structurefire', 49), ('#fire', 31)]
A3 :  [('#raw', 39), ('#structurefire', 38), ('#nonstructurefire', 30), ('#saveconstantine', 26), ('#epping', 21)]
A2 :  [('#krazykristmas', 177), ('#nonstructurefire', 72), ('#structurefire', 70), ('#rcl1milliongiveaway', 59), ('#egsholidaygiveaway', 57)]
C5 :  [('#melbourne', 60), ('#victraffic', 31), ('#photo', 20), ('#australia', 18), ('#1000steps', 18)]
D5 :  [('#lost', 30), ('#cat', 28), ('#somebodytocon', 27), ('#melbourne', 25), ('#findjasper', 23)]
A1 :  [('#melbourne', 15), ('#bbau', 15), ('#tattoo', 14), ('#worldjudo2014', 11), ('#vobis', 11)]
A4 :  [('#auspol', 7), ('#qldvotes', 6), ('#lnp', 6), ('#libspill', 5), ('#vicfires', 3)]
NoneType

real    3m15.242s
user    0m0.025s
sys     0m0.040s
```

Figure 7: Screenshot of SPARTAN (2 nodes 8 cores) Output