COMP 90024 – Cluster and Cloud Computing
Assignment 1

Suryadi Tjandra
shtjandra@student.unimelb.edu.au
1112164

Dongfang Wang
dongfangw@student.unimelb.edu.au
906257

Introduction

In the past, a single machine could process all information it needs for a job. As time went, the amount of data kept growing, but so did the capabilities of a machine. In recent times, however, physical limitations have slowed down the advancements of a machine's processing power. Meanwhile, the amount of information keeps getting larger and larger. To achieve higher performance in the face of large amount of data, it is necessary to let the job be done by multiple processes or machines at once in parallel.

In this assignment, we perform a series of tests on the performance of parallel computing on a large dataset on a computer cluster, and then record the time it takes for the cluster to finish processing the data.

Methodology

A. Cluster and Dataset

- The computer cluster used is SPARTAN, a High Performance Computing (HPC) cluster operated by the University of Melbourne. It uses SLURM Workload Manager to schedule its jobs.
- The dataset used is the file *bigTwitter.json*. It is a JSON file containing around 20 Gigabyte of Twitter posts.

B. Algorithm

The job that is run on SPARTAN is an algorithm to analyse the Twitter dataset, count the occurrences of each hashtag as well as the occurrences of each language, and announce the 10 most used hashtags and languages. The code is contained in the file *TwitterCount.py* and written in Python.

To parallelize the job, we use Message Passing Interface (MPI) system, using the implementation of the *mpi4py* Python library. It is assumed that each Twitter post require the same amount of processing power. Therefore, the distribution follows a Round Robin pattern so that each machine is given an equal amount of work. For example, if there are three machines, the first machine processes the first post, fourth post, seventh post, and so on, while the second machine processes the second, fifth, eighth, and so on.

C. Running the Tests

The algorithm will be run on SPARTAN three times each with different conditions.

The first test is run with only a single instance with a single core. This is to show the performance of a fully serial process -- i.e., no parallelization at all. To run this test, the following script in the file *1node1core.slurm* is invoked.

```
#!/bin/bash
#SBATCH --partition=physical
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=0-00:20:00
#SBATCH --output=1node1core.out

module load Python/3.4.3-goolf-2015a
time mpirun -np 1 python TwitterCount.py
```

The second test is run with a single instance with eight cores to reflect the performance of a multicore process. In this case, one machine can process eight tasks at once in parallel. To run this test, the following script in the file *1node8core.slurm* is invoked.

```
#!/bin/bash
#SBATCH --partition=physical
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=0-00:20:00
#SBATCH --output=1node8core.out

module load Python/3.4.3-goolf-2015a
time mpirun -np 8 python TwitterCount.py
```

The third test is run with two instances with four cores in each instance to reflect the performance of a distributed system. The total number of cores is eight, same as the second test, so that the results can be compared. The following script in the file *2node8core.slurm* is invoked to run this test.

```
#!/bin/bash
#SBATCH --partition=physical
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --time=0-00:20:00
#SBATCH --output=2node8core.out

module load Python/3.4.3-goolf-2015a
time mpirun -np 8 python TwitterCount.py
```

Results

A. Result

-----In data bigTwitter.json :-----
-----The top 10 hashtags are:-----
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

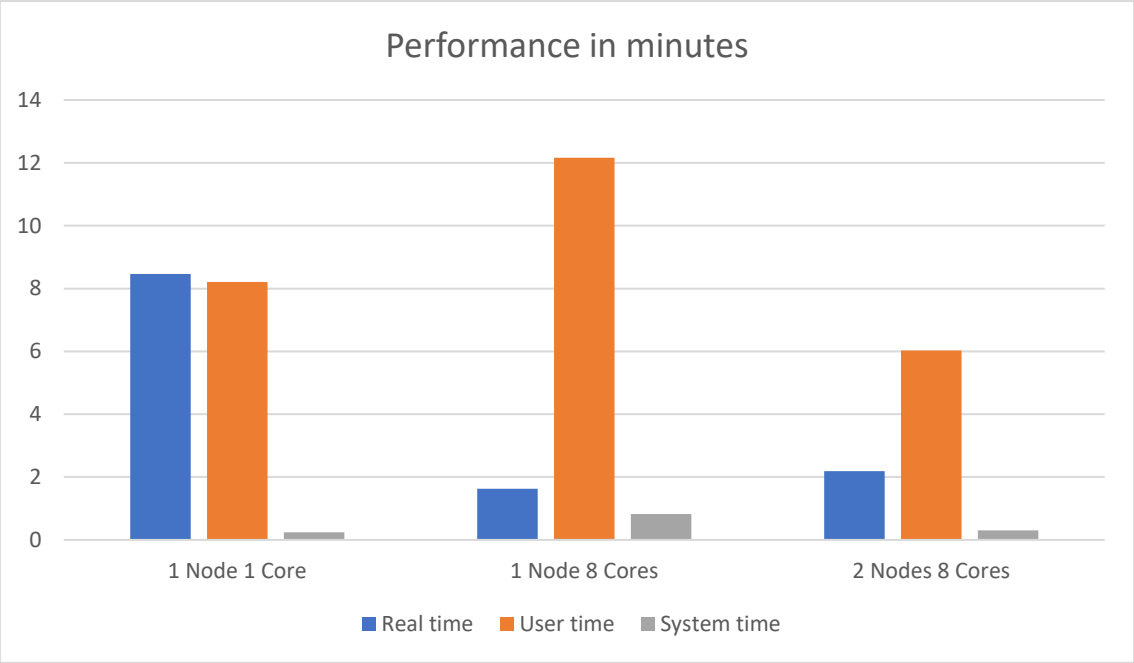
-----The top 10 languages used are:-----
1. English(en), 3107115
2. Undefined(und), 252117
3. Thai(th), 134571
4. Portuguese(pt), 125858
5. Spanish(es), 74028
6. Japanese(ja), 49929
7. Tagalog(tl), 44560
8. Indonesian(in), 42296
9. French(fr), 38098
10. Arabic(ar), 24501

B. Partition VS Scroll

The results of the tests are measured in real time, user time, and system time, as given by the outputs of the above scripts.

- Real time is how many seconds have passed from the start of the job until it finishes, as measured by the system's clock.
- User time is how many seconds is spent by processes to finish processing the user algorithm (in this case, to finish *TwitterCount.py*). For multiple cores, the user time is the total of all cores.
- System time is how many seconds in total spent doing kernel operations such as forking or joining.

The results are contained in the files *1node1core.out*, *1node8core.out*, and *2node8core.out*. They are summarized in the chart below:



In the first test, real time and user time are almost equal, both taking around 8 minutes. This makes sense, as there is only one core doing the entire job. In contrast, system time is minimal because the kernel does not have much to do other than initializing and terminating the job.

In the second test, real time is significantly reduced to less than 2 minutes. This proves that multicore processing can deliver significantly higher performance than fully serial processing. On the other hand, user time and system time are increased. This could be caused by several issues within concurrent computing, such as access locking to avoid race conditions and overheads. The increase in system time might be because the kernel needs to fork and join multiple processes. These problems increase the time for each process to finish, and so the real time is higher than the ideal (real time of first test divided by eight).

In the third test, real time is also significantly reduced compared to the first test. However, it is slightly higher than the second test. Because the third test includes multiple machines, this can be explained by network latency. On the other hand, user time and system time is reduced by half compared to the second test. This might be because there are two machines doing the job, so the workload of each machine is cut by half.

Conclusion

As the results have shown, utilizing parallelization and multiple processors and machines can significantly increase the performance of computers for processing a large amount of data, as opposed to the traditional fully serial processing. However, parallelization comes with its own issues that may prevent it from reaching its ideal performance.