Cluster and Cloud Computing Assignment 1

HPC Twitter Processing

University of Melbourne

Jack Crellin - 1168062

Hoang Viet Mai - 813361

15th April, 2020

Abstract

The purpose of this project is to investigate the reduction of time cost through the implementation of a parallelised application using a High Performance Computing (HPC) cluster. A 20.7GB dataset extracted from Twitter was to be curated and processed to return the most common hashtags and languages used. This was conducted using differing levels of parallelisation and the time taken to complete was measured and compared. The results of this task were as hypothesised, as the time required to complete the task was reduced as more nodes were introduced. The key takeaway is that using a parallelised application can reduce the time required to execute a given task.

Introduction

The main objective of this project is to be able to carry out the processing of the data in parallel with different parameters.
The twitter dataset is a JSON file that contains a list of tweets and its associated metadata. Each tweet may have a designated language and hashtags that were used in the tweet as well as those in the reply thread. These are to be counted and aggregated so as to find the ten most common of each within the dataset.

To carry out the task, access to the University of Melbourne's HPC cluster, SPARTAN, was allowed. On this cluster, jobs could be scheduled and run as instructed by the user. In this case, the process was to be completed three times with different levels of computing power. These were as follows:

- 1 node with 1 core.
- 1 node with 8 cores.
- 2 nodes with 4 cores each.

Implementation

Our application was written in Python and made use of the Mpi4py library. We decided to have each worker read and process their assigned part of the file in parallel using MPI-IO, then aggregate and display the results at the master process. To coordinate the parallel reading process, we assigned each worker a unique offset, based on their respective rank and the size of the input file. That is, each worker will start reading the file at a certain *i*th byte, where *i* equals the worker rank multiplied by the length of their respective equally assigned part, which is the length of the whole file divided by the number of workers. Additionally, to prevent memory error, we have each worker further divide their part into multiple chunks and process them iteratively.

We let a given chunk and the next chunk have some overlapping data, whose length is equal to the amount of byte that we think the longest newline-delimited substring will have (we gave this number a default value of 20KB in our implementation, it can be adjusted if necessary). Afterwards, each worker will start at the first newline character from the start of their assigned part and stop at the first newline character in the overlapping region. This method ensures that no json object is broken and therefore skipped, or being processed more than once.

We decided to use regular expressions to extract the only ASCII-based hashtags from the tweets' texts. That is, any hashtags written non-English letters and hashtags in other locations such as user profiles or retweets, will be skipped.

In case of ties between two or more hashtags/languages, they will all be given the same ranking. For example, if there are two second-ranked hashtags, the next rank number being assigned will be three, not four. All hashtags/languages that are tied for the tenth place will be included, resulting in more than ten hashtags/languages being displayed.

Scripts for job submission

1 node and 1 core:

```bash
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --time=0-02:00:00

# Load required modules
module load Python/3.7.3-spartan_gcc-8.1.0

# Launch application on 1 node and 1 core.
echo 'COMP90024 Assignment 1, 1 node and 1 core.'
time mpirun python3.7 assignment1mpi.py bigTwitter.json
```

1 node and 8 cores:

```bash
#!/bin/bash
```

```
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --time=0-02:00:00

# Load required modules
module load Python/3.7.3-spartan_gcc-8.1.0

# Launch application on 1 node and 1 core.
echo 'COMP90024 Assignment 1, 1 node and 8 cores.'
time mpirun python3.7 assignment1mpi.py bigTwitter.json
```

2 nodes and 4 cores each:

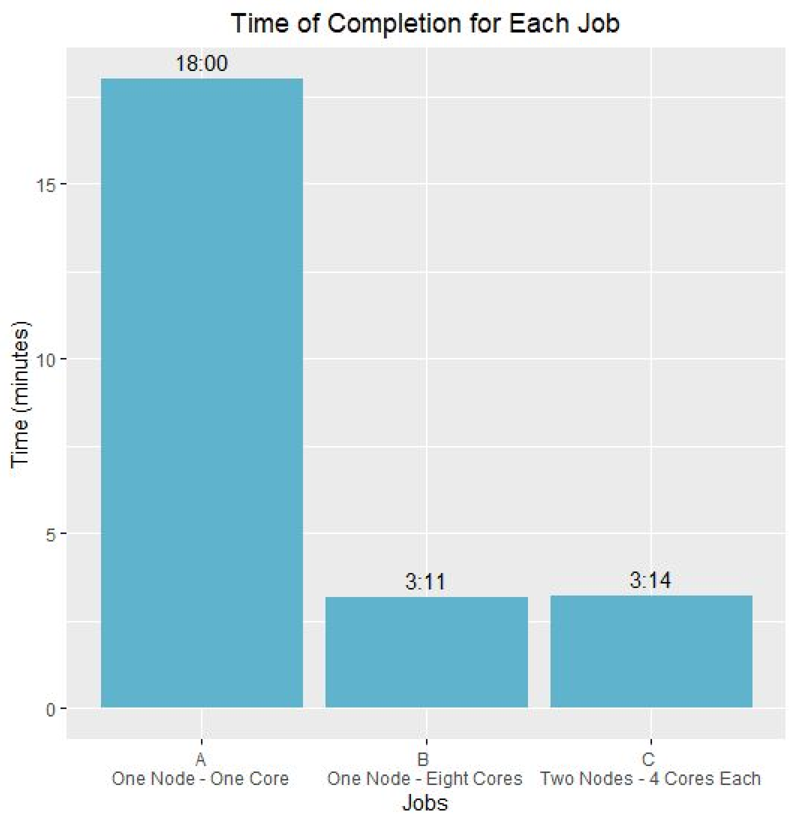
```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --time=0-02:00:00

# Load required modules
module load Python/3.7.3-spartan_gcc-8.1.0

# Launch application on 1 node and 1 core.
echo 'COMP90024 Assignment 1, 2 nodes and 4 cores each.'
time mpirun python3.7 assignment1mpi.py bigTwitter.json
```

To run each of the three scripts above once on SPARTAN, use command `bash submit_jobs.sh`, where `submit_jobs.sh`:

```
sbatch assignment1_1node_1core.slurm
sbatch assignment1_1node_8cores.slurm
sbatch assignment1_2nodes_4cores_each.slurm
```

Performance

The time of completions are as shown in the figure below. Both jobs B and C provided a significant reduction in when compared to job A. By increasing the number of workers available to run in parallel, the time taken dropped from 18 minutes to 3 minutes 11 seconds and 3 minutes 14 seconds, respectively.



The results that were recorded were as expected: increasing the number of cores and/or the number of nodes would speed up the process. This is because there is more computing power available to complete the task.

Although the job with a single node and 8 cores had a slightly faster completion time in comparison to the job with two nodes and 4 cores each, it is unlikely that there is any real difference between the two, it is possible that any difference stems from causes that are unrelated to the scaling of the task.

For the jobs with multiple workers, there was approximately a 6-fold reduction in execution time. This is of note because it shows that the improvement made is not linear to the number of cores available: the number of cores increased 8-fold for both. This is to be expected for parallelised application: there will be sections where the process cannot be parallelised. In our implementation, it was the aggregation and preparation of results to be displayed at the master process.

Conclusion

Through the completion of this project, the effects of implementing a HPC cluster has been successfully demonstrated. The task of processing Twitter data to aggregate hashtags and languages on different resources has shown that an increase in computing power saw a reduction in execution time. However, this reduction may not be linear, due to parts of the process that cannot be parallelised.