Cluster and Cloud Computing: (COMP90024) Assignment 1

Han Liu
Student ID: 906075
Email: h.liu71@student.unimelb.edu.au

1. Introduction

1.1 Assignment Objection

The objective of the assignment is to develop a simple parallelised program to parse a big Instagram dataset, and count Instagram posts in Melbourne according to the posts' geographical location.
This task is implemented on the University of Melbourne's cloud infrastructure Spartan, with different set of resources: one node one core, one node eight cores and two nodes eight cores. For student, the target of the experiment is to let us getting familiar with manipulation on the HPC platform and the common tools such as MPI, as well as comparing the performance of parallel computing with different settings.

1.2 Experiment Setting

- HPC platform: Spartan
- Workload Manager: Slurm (in bash script)
- Developing Language: Python (version: Python/2.7.9-goolf-2015a)
- Python library imported: json, mpi4y, pprint, numpy, time

2. Scripts

Following are the scripts written in python and bash to execute the job.

2.1 Code Sturcture

```
├── data
├──── melbGrid.json         # Melbourne coordinates in grids
├──── BigInstagram.json     # array of BigInstagram json data
├── output                  # output result from slurm job
├── slurm-job
├──── 1node_1core.slurm     # slurm job with 1 node and 1 core configuration
├──── 1node_8cores.slurm    # slurm job with 1 node and 8 cores configuration
├──── 1node_8cores.slurm    # slurm job with 2 nodes and 8 cores configuration
├── rank.py                 # main python script
```

2.2 rank.py

rank.py is the main code, which perform the the following tasks.

1. Read Json file line by line into a list, which contain all the coordination/geographical location.
2. Split the data into N parts, N should be the same number of processors specified in Slurm. Here I tried two ways of spliting files:
- one is by python's library numpy
- another is by using loop

There seems no obvious difference in efficiency of the two approach.

3. Scatter the files to different processors.
4. Each processor perform the computing task
5. Gather the data from processors.

The scatter and gather flow is described in Figure 1. One thing to note here is these routines need to specify a root process and all processes must specify the same root.

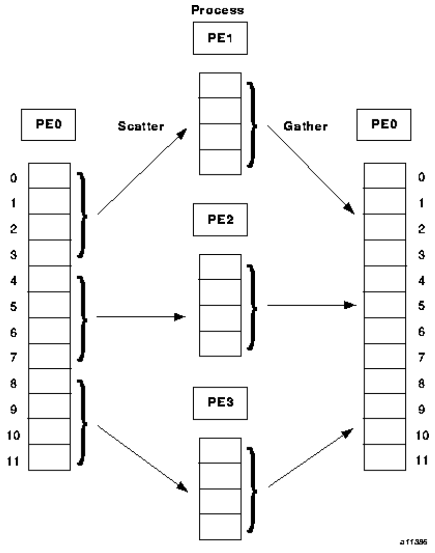


Figure1: Scatter and Gather Process

2.3 Slurm(in bash script)

The slurm script states all the requirements including wall time, number of nodes, cores and error file name. To be mentioned, the working directory was set to be the directory from where the job is submitted to queue. There are several confusing conception here, such as "--ntasks" and "ntasks-per-node". By defination, "--ntasks" advises the Slurm controller that job steps run within the allocation will launch a **maximum** of number tasks, while "--ntasks-per-node" Requests that ntasks be invoked on each node. To figure out the difference, I run several jobs and find that "--ntasks = 8" has shorter run time than "--ntasks-per-node=8".

Resources	Time
2 node 8 cores(ntasks-per-node)	2'34''
2 node 8 ntasks	2'28''
2 node 8 cores 8 nstasks	2'30''

Table 1. Comparison on execution time between "--ntasks" and "ntasks-per-node"

3. Result Analysis

Here is the outcome of running time of different jobs

Resources	Time
1 node 1 core	2'32''
1 node 8 cores	2'34''
2 node 8 cores	2'34''

Table 2: Variations in total execution time on different resources.

Surprisingly, the executing time on different cores and nodes are similar, with little variation. To find the difference in detail, I run another two jobs, with setting timepoint after each part of the job complete, such as reading file, scattering and gathering.

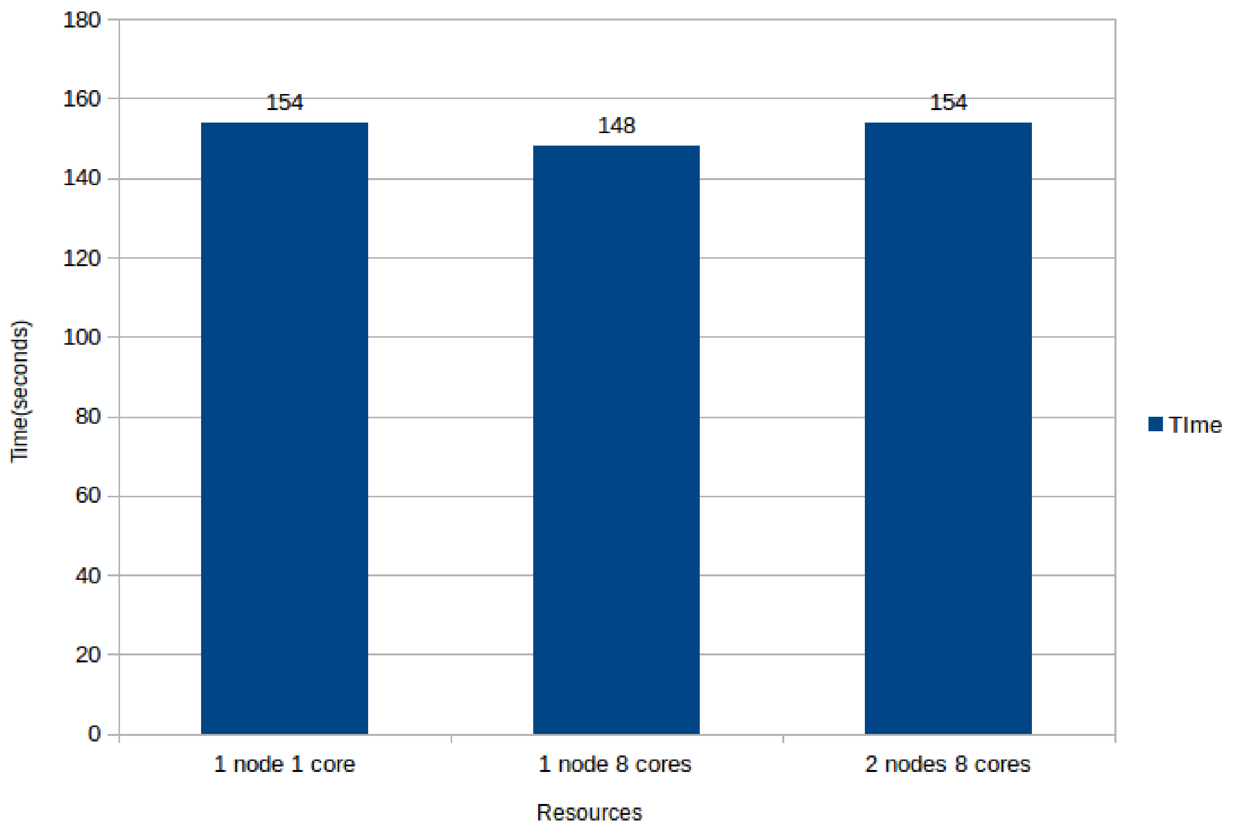


Figure 2: Variations in total execution time on different resources.

Now we can see that even though the total time on 1 node 1 core, 1 node 8 cores, 2 nodes 8 cores are similar, but most of the time on 3 jobs are spent on reading big json file into the coordination list. In terms of computing time, the 1 node 1 core costs about 4 seconds, while the multicore jobs cost almost zero. But with the growing of cores, more time were spent on scattering.

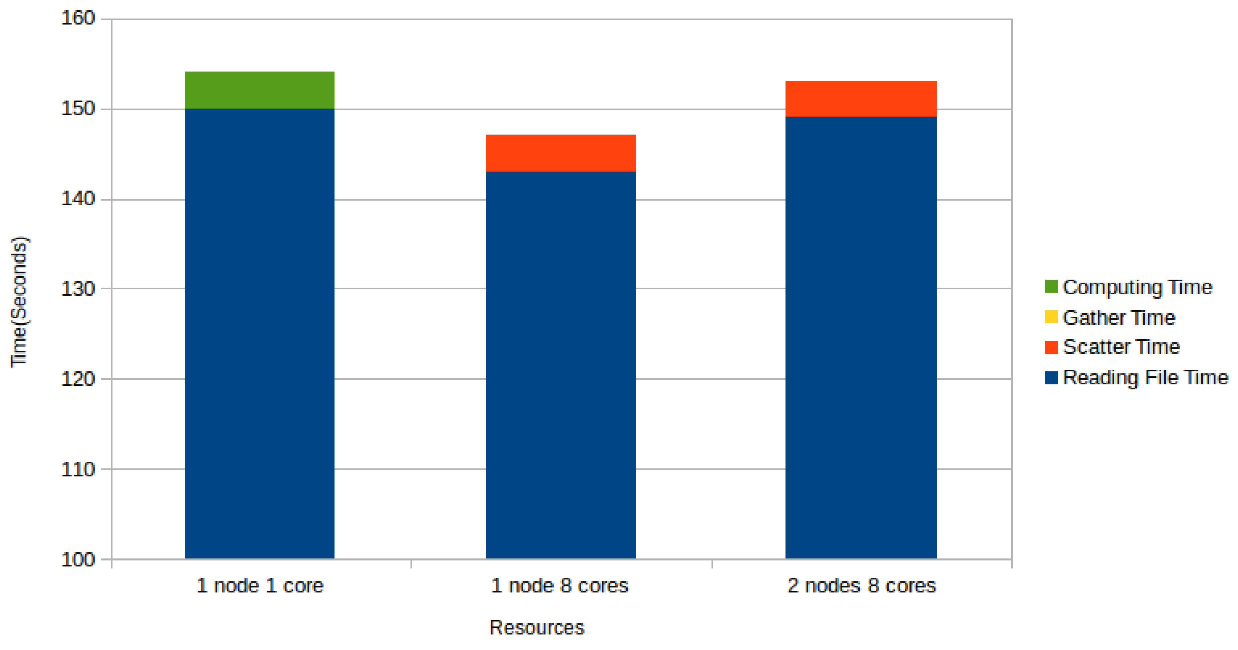
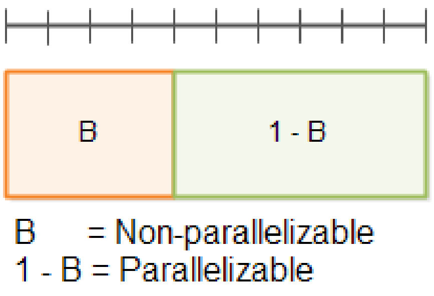


Figure3: Variations in distribution of execution time on different resources.

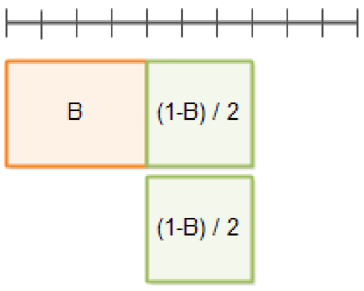
4. Conclusion

This experiment is a verification of Amdahl's law: Speedup is limited by the total time needed for the sequential (serial) part of the program. The success in improving performance of parallel computing lies in figuring about as many as process can be paralleled.

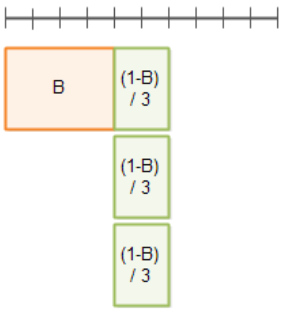
> *In parallelisation, if B is the proportion of a system or program that can be made parallel, and 1-B is the proportion that remains serial, then the maximum speedup that can be achieved using N number of processors is 1/((1-B)+(B/N).*



Amdahls law illustrated.



Amdahls law illustrated with a parallelization factor of 2.



Amdahls law illustrated with a parallelization factor of 3.