COMP90024 Cluster and Cloud Computing
Assignment 1 Report

Student ID: 813174 Chengeng Liu
Student ID: 964135 Xinze Li

1.1 Introduction

The objective of the project is to write a parallel program to compute total tweets and hashtags in the scope of Melbourne grid, based on the information provided in an 11 GB file, named 'bigTwitter.json'. The bottleneck of the project is dealing with io operation, since the large amount of data will be hard to handle. Implementing a Message Passing Interface(MPI) across a set of clusters and nodes will be helpful to overcome the difficulty of io operation. This program uses MPI4PY for parallelizing the io operation and data exchange within the program for each process. This program is written in python and running on the University of Melbourne High Performance Computing platform called Spartan.

1.2 Technology used in the project

Development language —— Python (Python/3.6.4-intel-2017.u2-GCC-6.2.0-CUDA9)
Python package —— mpi4py, json, time
Platform —— Spartan
Workload manager —— Slurm

1.3 Assumption made in the project

Hashtag is extracted from 'text' in 'doc' field of the json file. It is considered to be a valid hashtag if and only if it has a pattern of SPACE#STRINGSPACE.
We distinguish uppercase letter and lowercase letter when extracting hashtag. For example, ' #Helloworld ' is not the same as ' #helloworld '.

2. Scripts —— Slurm

Since the requirements specify that three different ways of running the program is required, there are three Slurm scripts used for the program. 1node1core.slurm, 1node8cores.slurm and 2nodes8cores.slurm.

i. 1node-1core script
```
#!/bin/bash
#SBATCH --partition cloud
#SBATCH --time=00:5:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=1
#SBATCH --job-name=1node_1core

echo “1node 1core "

module load Python/3.6.4-intel-2017.u2-GCC-6.2.0-CUDA9
time mpirun python3 Assignment1.py
```
ii. 1node-8cores script

```
#!/bin/bash
#SBATCH --partition physical
#SBATCH --time=00:15:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=1
#SBATCH —job-name=1node_8cores
echo “1node 8cores”
module load Python/3.6.4-intel-2017.u2-GCC-6.2.0-CUDA9
time mpirun python3 Assignment1.py
```

iii. 2nodes-8cores script

```
#!/bin/bash
#SBATCH --partition cloud
#SBATCH --time=00:50:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --cpus-per-task=1
#SBATCH --job-name=2nodes_8cores
echo “2nods 8cores"

module load Python/3.6.4-intel-2017.u2-GCC-6.2.0-CUDA9
time mpirun python3 Assignment1.py
```

The Slurm scripts state all requirements such as wall time, cores needed, number of nodes and job name etc. ‘cpus-per-task’ indicates how many cpus are required. For all of the scripts, the value is set to be 1. ‘ntasks-per-node’ specifies how many processes will be initiated for the program. It should be the same as the number of cores. ‘nodes’ define the number of nodes as required.

3. Approaches and algorithms used

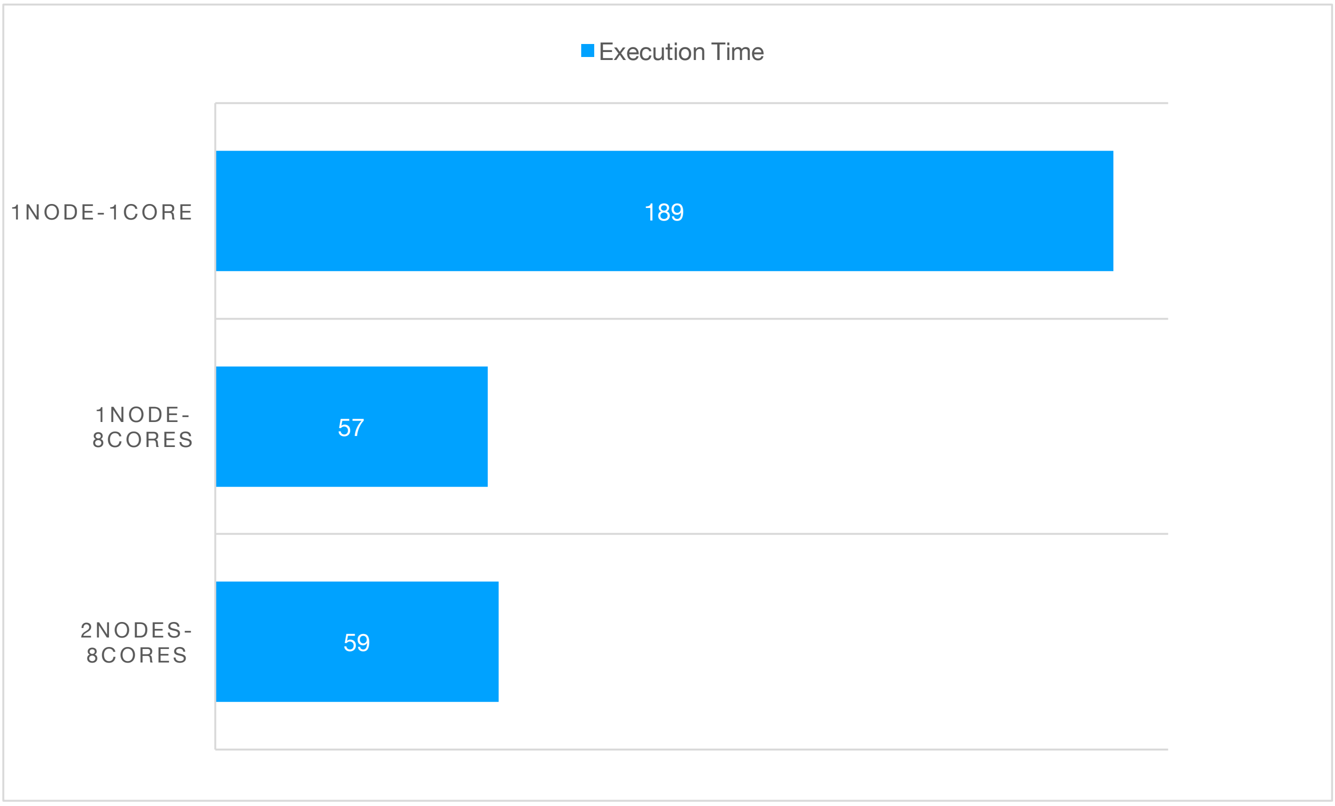
a. pre-processing and filter data
One of the most important feature when implementing a program involving large amount of data is to do text analysis and filter out ‘noise’ data. There is 11 GB data in the ‘bigTwitter.json’ file and it contains 2.5 million lines of json strings. Firstly, it would be impossible to load the whole file into memory. Secondly, loading such a big data into the program will be super slow and inefficient. After analyzing the data provided, we found out that if we filter out useless data, such as data without geolocation or out of scope of the scope of Melbourne grid, then there is a lot of time and space saving. By doing data filtering, 2.5 million lines of json strings are decreased to about 0.4 million lines of json strings. Combining data pre-processing and implementing MPI will increase the efficiency of reading the file.

b. Master-slave idea using MPI
Master-slave idea is compatible with Message Passing interface, especially when there are multiple cores operating on the same piece of data. For every line in the data, each core will only operate on and proceed if and only if line number mod number of cores equals current core counter. This will assign each core, or slave node, an equal amount of data to proceed on. Then in the master node which has rank of 0, all the data proceeded by other slave nods will be collected. By using master-slave model, the io cost is decreased. However, there do exist other better way of using MPI. For example, there is a lot of waste for all nodes to read in the same piece of data. We could have divided the file into 8 chunks, given the fact that we have 8 cores. Each node will only read in 1/8 of the

original data and proceed in that way. We aren't able to do so because we cannot figure out how to deal with file pointers at first and time limits our further development.

4. Program execution and conclusion

For the sake of other people using Spartan, we didn't do too many benchmark test but only running for debugging program. The results may vary after running for many times, since there is difference between the computability of different nodes. The time taken for 1node-1core, 1node-8cores and 2nodes-8cores are shown below:



From the bar chart above we can know that, 1node-8cores is the fastest way of proceeding the file. The time consumed by 1node-8cores, comparing with 1node-1core, is almost 1/3 of the latter. This is a huge efficiency improvement. The reason why the 1node-8cores time is not 1/8 of 1node-1core is that there is always time consumed that cannot be parallelised. Those linear operation time decide how fast and how much can parallel computing improve.
Another fact is that 2nodes-8cores is slightly slower that 1node-8cores. This is what we have expected. Since two nodes require more communication between nodes and this can lead to delay in processing data and communication with other cores.
Using physical partition in workload manager instead of cloud partition also boosts our program. Information maybe delayed during the way being transmitted and this is one of the bottleneck of cloud computing.