Cluster and Cloud Computing
COMP90024

Assignment 1
HPC Twitter GeoProcessing

Aaron Robins 694098 robinsa1@student.unimelb.edu.au
Nakia Silva 796504 nakias@student.unimelb.edu.au

1. Purpose of report:

This report will outline the problem at hand, the approach taken, challenges faced and then analyse the performance and results of the system on a varying number of nodes and cores.

2. Problem:

Our task is to take a large JSON file of data, 11 GB, and implement a simple and parallelised program on Melbourne Universities HPC known as Spartan.
In order to process this amount of data fast, it is not possible to use one single computers processing power as the job is too large. Therefore, the purpose of this assignment is to understand how to implement a system that can take advantage of the processing power of a clustered system of computers, but complete the job and provide results as if the job had been completed by a single computer.
Our application will take geocoded Twitter data and identify Twitter usage around Melbourne and the most frequently occurring hashtags that are being sent from those areas.
Further to this, we will analyse the difference in performance between 1 node with 1 core, 2 nodes with 8 cores and 2 nodes with eight cores.

3. Challenges:

ID	Name	Description
1	Python Knowledge	As the assignment was strongly recommended to me completed in Python, our group decided to use Python. One member had some limited experience while the other team member was new to Python. This proved a challenge as we had to overcome difficulties with syntax and coding knowledge rather than just focussing on the problem statement at hand.
2	MPI	This was the groups first exposure to Message Parsing Interface programming. Traditional coding practices such as pointers, recursion, loops and global variables etc prove to be challenging when using MPI. For example, can a global variable be maintained over a clustered machine? If so how do we update it and make sure multiple processes aren't trying to update the same variable at the same time.
3	How to Segment the data	One of the key blockers that we faced was figuring out a way to segment the data to the individual processors. We tried to read the data line by line and scatter it to nodes, however this did not work as the Master node has to wait for all jobs to be complete before gathering the information back. We also considered a solution which helps the node decide weather to process the line based on the Modulus of the ID number and the Rank number. Finally we came to a solution which involved splitting up the JSON file into smaller files, where each node would process their assigned file.

4. Implementation:

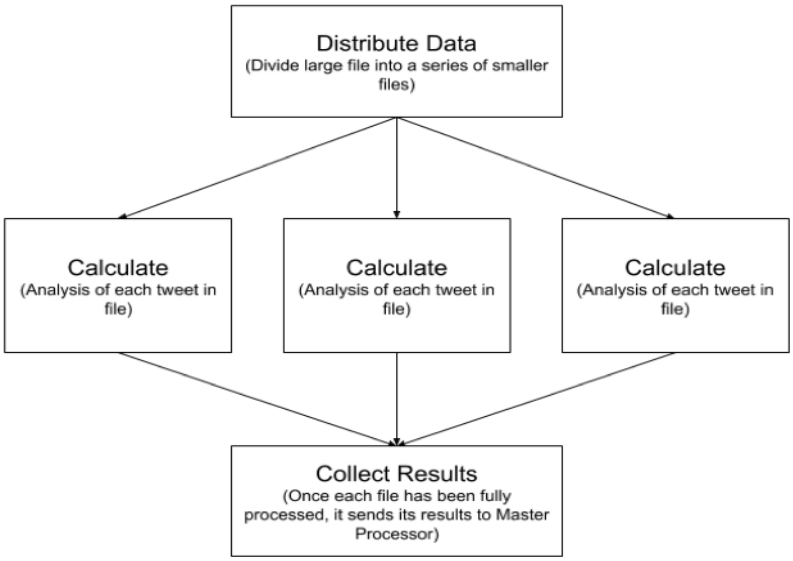
4.1 Reading melbGrid.json file

The coordinates for each grid were extracted from melbGrid.json file on running the function create_grid. The function, named create_grid takes an input of file path to the JSON melbGrid.json file. A list of size pertaining to the number of grids was created where each row corresponded to the grid coordinates in the order they were represented in melGrid.json file. Simultaneously another json object is created with all the distinct grid names. Whenever we need to check coordinates of a tweet to assign it to a grid ID, we call this function. The representation of the grid coordinates and the grid names is included in the appendix A.

4.2 Reading Twitter data:

One Node One Core :
When the slurm job is submitted to perform 1 task using one node , the master node performs the task. The data is not splitted but read line by line and processed to update a json object which holds the result consisting of grid name, number of tweets and top 5 hashtags and their count.

One Node Eight cores And Two Node Eight cores (4 cores per node) :
To process this amount of data we used a paradigm known as Message Passing Interface, specifically we used the mpi4Py Library. After considering many approaches to handle data in case of nodes more than 1, we chose to split the large JSON Twitter file into small files, then store the name of these files in an Array. This is an extension of Single-Program Multiple-Data programming and can be visualised in the following way.



The below command is used to split the bigTwitter json file into smaller files equal to the number of cores or number of cores +1 depending upon the output of split function .

```
split -l $((`sed -n '$=' bigTwitter.json`/(no_of_cores)) bigTwitter.json -d bigTwitter_
```

Incase if the division does not produce a definite output then we will be an extra file.

```
if rank == 0 :
    # We are master
    print ("Master start")
    master_tweet_processor(comm, [input_file[0],input_file[-1]], grids_file)
else:
    # We are slave
    print ("Slave start")
    slave_tweet_processor(comm, [input_file[0]],grids_file)
```

Our Master processor will process input_file[0] and the last file (input_file[-1]). Once it has finished processing this file it will then wait for all of the Slave processors to finish processing their files by sending a message asking to return data. Once the slave receives this message and has finished processing the data, it will then respond to the message by sending the results. Once the results have been received the Master will send an a message which will end the slave process.
At the end the small files created are removed using script command.
The script to split the bigTwitter file and remove the small files are written in slurm file, But they can be alternatively included in the python code.

5. Slurm scripts for submitting jobs

The below Script is used to run the three slum jobs on spartan.

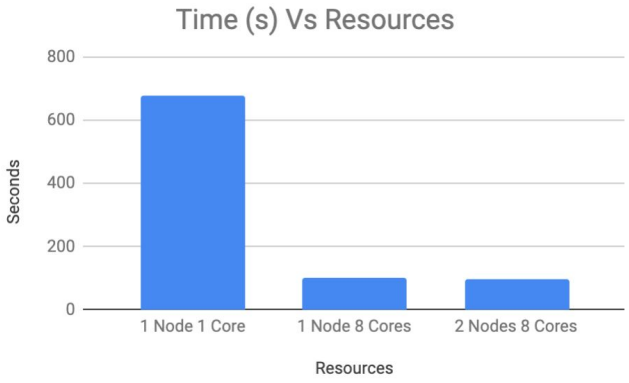
```
#!/bin/bash
#SBATCH --nodes=(number of nodes)
#SBATCH --ntasks=(number of nodes)
#SBATCH --time=0-1:00:00
#SBATCH --partition=cloud

# Load required modules
 module load Python/3.6.4-intel-2017.u2

# 1 node 1 core
time mpiexec -n 1 python3 GridAllocation.py bigTwitter.json melbGrid.json

# 1 node 8 cores
split -l $((`sed -n '$=' bigTwitter.json`/8)) bigTwitter.json -d bigTwitter_
time mpiexec -n 8 python3 GridAllocation.py bigTwitter_ melbGrid.json
rm bigTwitter_*

# 2 node 8 cores
```

```
split -l $((`sed -n '$=' bigTwitter.json`/8)) bigTwitter.json -d bigTwitter_
time mpirun python3 GridAllocation.py bigTwitter_ melbGrid.json
rm bigTwitter_*
```

There are 3 slurm scripts submitted with the assignment. Each slurm file for each job respectively.

6. Analysis of performance:

Resources	Time taken for execution
1 Node 1 Core	677.5512 seconds
1 Node 8 Cores	100.8878 seconds
2 Nodes 8 Cores	94.5867 seconds



Our results have shown that there is a great improvement in performance from a 1 Node 1 Core system versus a 1 node 8 core system, down from 678 seconds to 101 seconds. This falls in line with Amdahl's law which establishes the maximum improvement when only part of the system has been improved on a fixed data set. However, it should be considered that there are certain serial overheads that can't be serialised which are running in the code, which is one of many reasons the maximum theoretical improvement has not been reached.

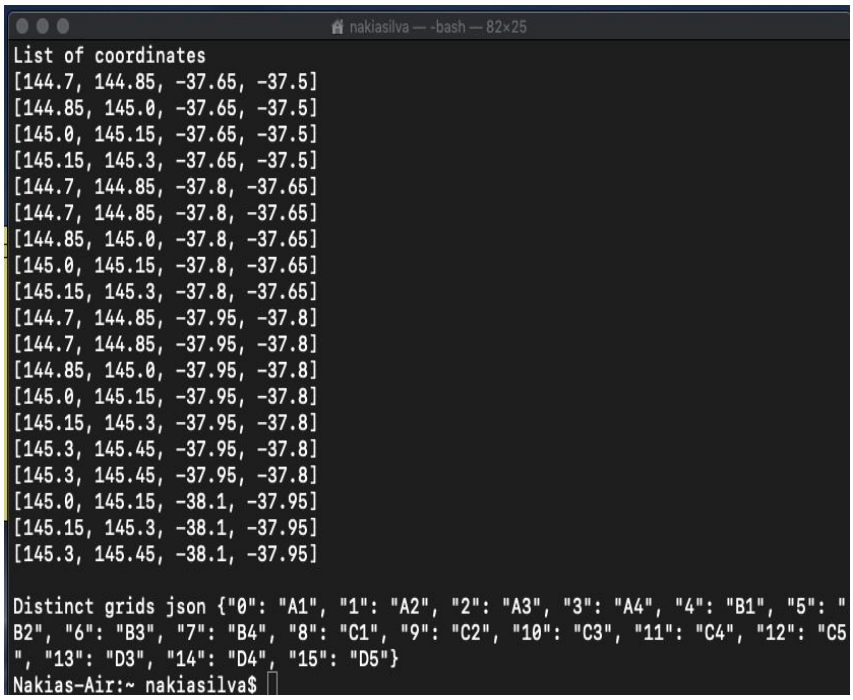
On running the program with 2 nodes and 8 cores we saw a further improvement to 95 seconds. Spartan is a clustered HPC which has a head node for users to login on, and then multiple other nodes as resources for users to consume. Having access to a distributed computing system with 2 nodes improves the available resourcing compute power, processing the job in a faster time.

7. Conclusion

Solving this problem has been both challenging and rewarding. Our challenges included a low level understanding of Python, limited experience with Linux systems and no experience with MPI programming. Whilst getting the program to run on 1 node 1 processor did not take that much time, we ran into further challenges when having to find a way to segment the data amongst the processors. Once this problem was solved we were happy with the results however do think there is room for improvement in making our code more efficient. Specifically the section of the code that allocates the grid and has many loops.

Appendix A:

1.


nakiasilva — -bash — 82×25
List of coordinates
[144.7, 144.85, -37.65, -37.5]
[144.85, 145.0, -37.65, -37.5]
[145.0, 145.15, -37.65, -37.5]
[145.15, 145.3, -37.65, -37.5]
[144.7, 144.85, -37.8, -37.65]
[144.7, 144.85, -37.8, -37.65]
[144.85, 145.0, -37.8, -37.65]
[145.0, 145.15, -37.8, -37.65]
[145.15, 145.3, -37.8, -37.65]
[144.7, 144.85, -37.95, -37.8]
[144.7, 144.85, -37.95, -37.8]
[144.85, 145.0, -37.95, -37.8]
[145.0, 145.15, -37.95, -37.8]
[145.15, 145.3, -37.95, -37.8]
[145.3, 145.45, -37.95, -37.8]
[145.3, 145.45, -37.95, -37.8]
[145.0, 145.15, -38.1, -37.95]
[145.15, 145.3, -38.1, -37.95]
[145.3, 145.45, -38.1, -37.95]
Distinct grids json {"0": "A1", "1": "A2", "2": "A3", "3": "A4", "4": "B1", "5": "
B2", "6": "B3", "7": "B4", "8": "C1", "9": "C2", "10": "C3", "11": "C4", "12": "C5
", "13": "D3", "14": "D4", "15": "D5"}
Nakias-Air:~ nakiasilva$