COMP90024 Cluster and Cloud Computing
Assignment 1 Report

Student ID: 1323782, 1204405
Name: Taylor Tang, Lingling Yao

1. Overview

This report aims to demonstrate the use of parallel processing through High Performance Computing (HPC) for JSON file processing tasks, and to analyse the performance differences across various resource configurations and methods.

2. Deploy processing command on Spartan

Slurm scripts used in this report is explained table 1 below:

	Slurm scripts	**Explanation**
Resource allocation	#SBATCH --node=x #SBATCH --ntask-per-node=y	To require corresponding resources, x=node, y=core
Environment	module load mpi4py/3.0.2-timed-pingpong module load python/3.7.4 source ~/virtualenv/python3.7.4/bin/activate	To load working environment, virtualenv is installed with ijson package
Run scripts	time mpiexec -n core_amount python3 scripts.py	To run mpi4py with corresponding python file
Record usage and email notification	my-job-stats -a -n -s #SBATCH --mail-user=xxx@yyy.zz #SBATCH --mail-type=FAIL, BEGIN or END	To record system usage and create email notifications when jobs start and end.

Table 1 - slurm scripts usage

A screenshot of 2 node 4 cores each node setting slurm script is shown diagram 1 below:

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --time=0:15:0
#SBATCH --mail-user=chaojunt@student.unimelb.edu.au
#SBATCH --mail-type=FAIL
#SBATCH --mail-type=BEGIN
#SBATCH --mail-type=END
module load mpi4py/3.0.2-timed-pingpong
module load python/3.7.4
source ~/virtualenv/python3.7.4/bin/activate
time mpiexec -n 8 python3 ~/MPI_v5.py -s sal.json -t bigTwitter.json
deactivate
my-job-stats -a -n -s
```

Diagram 1 – screenshots of 2n8c configuration

3. Parallel processing

Two methods are chosen to process a big JSON file in parallel.

- **Method 1**, having all cores read through the JSON document (using ijson package for JSON object streaming), each core picks its own jobs based on a mod calculation, then processes the results. At the end, all results are collected and merged at one core.

  The drawback of this method is that each core needs to read through the whole JSON document while they only need to process a part of the document. Reading through all documents is unnecessary, hence extra effort spent on redundant tasks. If processing JSON objects doesn't take long, the benefits of parallel processing are limited (refer to performance analysis section).

- **Method 2**, having each core read only part of the JSON document (by using python seek() method), parsing JSON objects, process the results, and send the result to the master core at the end. The master core collects all processed data from other cores and merges them into the final result (as illustrated in diagram 2 below).

  This method mitigates the drawback of method 1 with each core only reads a part of the document. However, the performance is limited by the slowest core among all cores and the amount of work can be parallelised (in this case, read json and parse address, but not merger and final output).

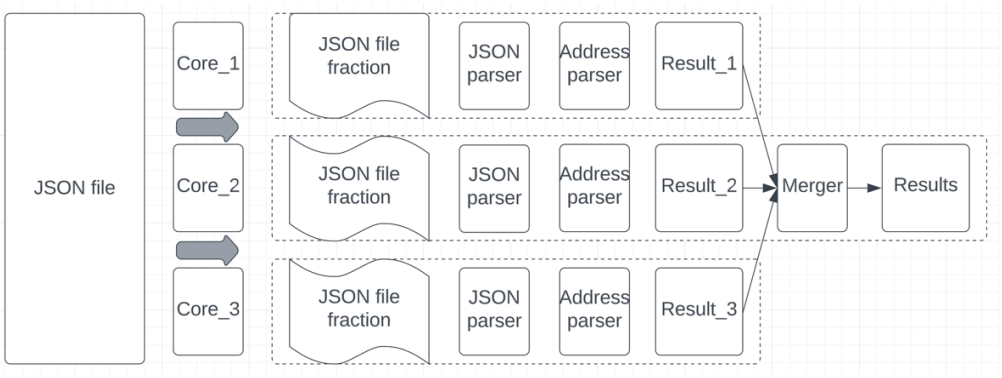

Diagram 2 – parallel processing method

4. Address data parsing

- **Step 1:** extract the place information from JSON object, split by comma, strip & lower-case it into a list of strings. The index 0 component is most likely to be the suburb while the rest of the list can contain city, state or country information (as shown in diagram 3 below).

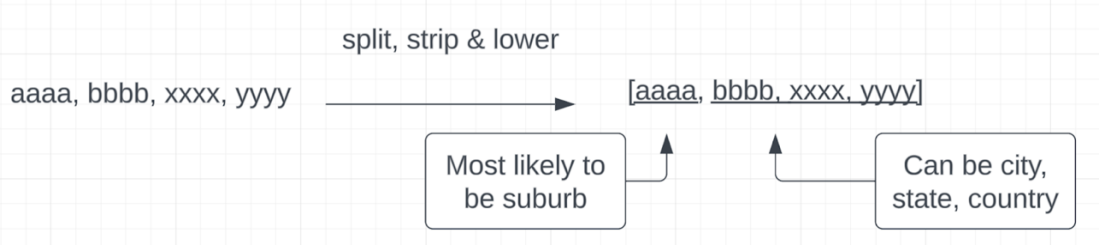

Diagram 3 – address data parsing process

- **Step 2:** Iteratively checking if the list of strings contains state or city information. This is being done by comparing each string to pre-defined city_dict and state_dict. If we successfully identified this place as belonging to one capital city, we can safely stop and output the city name.

- **Step 3:** If we cannot find city information in the list of strings, we then use the suburb information in association with sal.json to lookup this suburb's belonging city. First, we manipulate the sal.json into a dictionary with structure shown below, so we could perform an exact string match to find the true city information:

```
Gcc_code_dict = {"abbotsbury (nsw): "sydney",
                 "abbotsford (nsw): "sydney",
                 "abbotsford (vic.): "melbourne,
                 …}
```

- **Step 4:** if we only have suburb information (no city and state information), we look it up from the original sal.json file to find if this suburb belongs to a great capital city area or not. If not found, we conclude that this address is not in a great capital city area.

5. Final statistical result for each task

Task 1 – Most number of tweets made by the individual:

Rank	Author ID	Number of Tweets Made
#1	1498063511204761601	68477
#2	1089023364973219840	28128
#3	826332877457481728	27718
#4	1250331934242123776	25350
#5	1423662808311287813	21034
#6	1183144981252280322	20765
#7	1270672820792508417	20503
#8	820431428835885059	20063
#9	778785859030003712	19403
#10	1104295492433764353	18781

Task 2 – Most number of tweets made in greater capital cities

Rank	Great Capital City	Number of Tweets Made
#1	2gmel(Greater Melbourne)	2286780
#2	1gsyd(Greater Sydney)	2119039
#3	3gbri(Greater Brisbane)	858182
#4	5gper(Greater Perth)	589186
#5	4gade(Greater Adelaide)	460979
#6	8gact(Greater ACT)	202655
#7	6ghob(Greater Hobart)	90672
#8	7gdar(Greater Darwin)	46390
#9	9oter(Great Other Territories)	160

Task 3 - Most number of unique city tweeted

Rank	Author ID	Number of Unique City Locations and #Tweets
#1	1429984556451389440	8(#1920 tweets - #1880gmel, #13acte, #10gsyd, #7gper, #6gbri, #2gade, #1ghob, #1gdar)
#2	17285408	8(#1209 tweets - #1061gsyd, #60gmel, #40gbri, #23acte, #11ghob, #7gper, #4gdar, #3gade)
#3	702290904460169216	8(#1165 tweets - #294gsyd, #261gmel, #229gbri, #152gper, #121gade, #46acte, #41ghob, #21gdar)

#4	87188071	8(#394 tweets - #110gsyd, #86gmel, #65gbri, #51gper, #34acte, #28gade, #15ghob, #5gdar)
#5	774694926135222272	8(#272 tweets - #38gmel, #37gbri, #37gsyd, #36ghob, #34acte, #34gper, #28gade, #28gdar)
#6	1361519083	8(#260 tweets - #193gdar, #36gmel, #12gsyd, #9gade, #6acte, #2ghob, #1gper, #1gbri)
#7	502381727	8(#250 tweets - #214gmel, #10acte, #8gbri, #8ghob, #4gade, #3gper, #2gsyd, #1gdar)
#8	921197448885886977	8(#205 tweets - #58gmel, #46gsyd, #37gbri, #28gper, #24gade, #7acte, #4ghob, #1gdar)
#9	601712763	8(#146 tweets - #44gsyd, #39gmel, #19gade, #14gper, #11gbri, #10acte, #8ghob, #1gdar)
#10	2647302752	8(#80 tweets - #32gbri, #16gmel, #13gsyd, #5ghob, #4gper, #4acte, #3gdar, #3gade)

6. Performance analysing

Here we tested both method 1 and method 2 on spartan, and the performance of processing the bigTwitter.json file in different resource setting are shown in diagram 4 below:

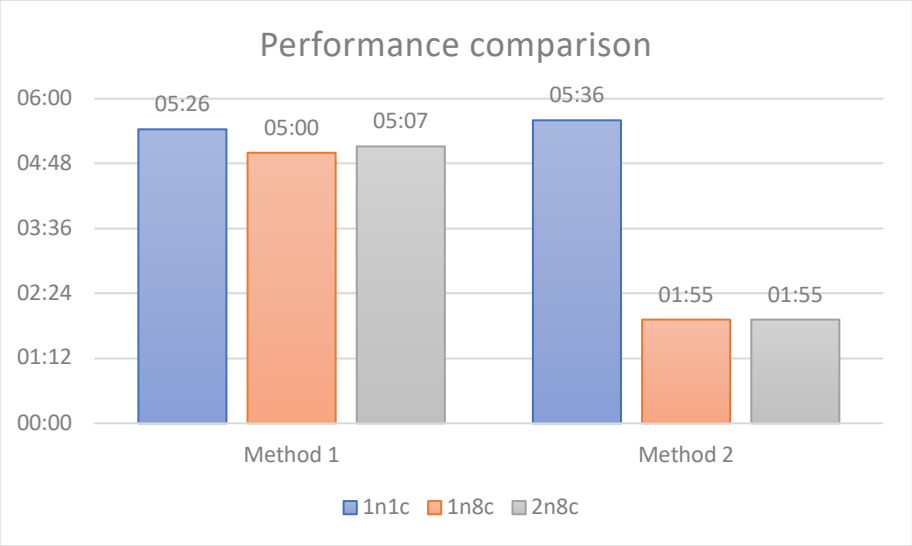

Diagram 4 – configuration and performance comparison

In method 1, three different resource configurations don't provide significant variance in terms of process time. This is because each core still needs to iteratively read through all the file content as previously discussed. Although each core needs to process less JSON objects, due to the actual processing being not quite computationally intensive, only marginally speed gains are spotted when using method 1.

In method 2, both parallel processing configuration provide a much faster performance than 1 node 1 core. By using method 2, each core only needs to read and process one eighth of the file. Hence, when having 8 cores working together, the total workload is averaged out and the total time needed is reduced. This is also proved by analysing the time usage (refer to slurm-46257497.out file attached in zip): real time is 1m41s while the user time is 12m44. This means it would take one core much longer time to process same file.

It is also worth noting that 8 cores configuration is not 8 times faster than 1 core configuration. This is because 1) in Amdahl's law: the performance gain through parallel processing is limited by set of operation that must be run in series (such as merging final result), and 2) we are using ijson to read json file in 1 core configuration and using python default read() and seek() in multicore configuration.

Comparing 1-node 8-cores and 2-node 8-cores, both provide a similar performance. This matches the mechanism of method 2, which is having all cores directly read the JSON file. As each core works independently most of the time and only finalized results are passed between cores in the last step, they provide a similar performance (variances caused by allocated CPU and different CPU usage).