**COMP90024 Cluster and Cloud Computing**

**Assignment 1 – Social Media Analytics**

**Saad Sheikh (saad.sheikh /saadullahs/1384242)**

**Shibu Thanadan Francis (sthanadanfra /** **sthanadanfra/1441017)**

Table of Contents

[**Purpose and Objectives** 2](#_Toc163367167)

[**DataSets provided** 2](#_Toc163367168)

[**Intended outcomes** 2](#_Toc163367169)

[**Computational Scenarios** 2](#_Toc163367170)

[**Slurm and data parallism code** 2](#_Toc163367171)

[**Our Approach for the project** 3](#_Toc163367172)

[**Approach#1: Imputation by data parallelism processing** 3](#_Toc163367173)

[**Approach#2: Imputation by summary of parallelism** 3](#_Toc163367174)

[**Data results** 4](#_Toc163367175)

[**Data results for 120GB on 1 Node/1Core** 4](#_Toc163367176)

[**Data results for 120GB on 1 Node/8 Cores** 4](#_Toc163367177)

[**Data results for 120GB on 2 Node /8 Cores** 4](#_Toc163367178)

[**Results comparison** 5](#_Toc163367179)

[**Processing time** 5](#_Toc163367180)

[**CPU Utilization** 6](#_Toc163367181)

[**RAM Utilization** 6](#_Toc163367182)

[**Compute Idle Utilization** 6](#_Toc163367183)

[**Results comparison** 7](#_Toc163367184)

[**Appendix** 7](#_Toc163367185)

[**Appendix1 : Python Code (shsa\_comp90024\_as1\_v1.86.py)** 7](#_Toc163367186)

[**Appendix2 : Slurm Job Results** 9](#_Toc163367187)

**Figures**

[Figure 1 Results output format 3](#_Toc163479198)

[Figure 2 Example of sbatch with srun scripts. 4](#_Toc163479199)

[Figure 3 Memory Error Logs for Approach#1 4](#_Toc163479200)

[Figure 4 Memory Utilization for Approach#1 5](#_Toc163479201)

[Figure 5 Results of Approach#2 with default memory allocation 6](#_Toc163479202)

[Figure 6 Results of Approach#2 with custom memory allocation of 200MB per CPU 6](#_Toc163479203)

[Figure 7 Processing results for 120GB on 1 Node/1 Core 7](#_Toc163479204)

[Figure 8 Processing results for 120GB on 1 Node/8 Core 7](#_Toc163479205)

[Figure 9 Processing results for 120GB on 2 Node/8 Core 8](#_Toc163479206)

[Figure 10 Data analytics results. 8](#_Toc163479207)

[Figure 11 Processing time results 8](#_Toc163479208)

[Figure 12 CPU Utilization rate 9](#_Toc163479209)

[Figure 13 RAM Utilization rate (Default allocation 3.91GB/CPU Core) 9](#_Toc163479210)

[Figure 14 RAM Utilization rate (allocation 200MB/CPU Core) 10](#_Toc163479211)

[Figure 15 Idle Utilization rate 10](#_Toc163479212)

# **Purpose and Objectives**

The objective of this project is to apply the learnings on parallel processing to a social media dataset of different sizes and understand computational relationships with data parallelism.

## **DataSets provided**

There are three datasets provided for analysis purpose

* Twitter datasets of 1mb storing ~1K records used for initial testing
* Twitter datasets of 50mb storing ~50K records used for testing
* Twitter datasets of 120GB+ used for final testing and report.

## **Intended outcomes**

The aim of the project is to perform an efficient way of data analytics to report the following four metrics from the data given.

1. **The happiest hour ever** =hour having most +ve accumulative value of sentiments
2. **The happiest day ever=** day having most +ve accumulative value of sentiments
3. **Most active hour ever=** hour having most number of tweets across the whole data
4. **Most active day ever=** day having most volume number of tweets

Below is the output format for results (Refer to Fig 1)

A close-up of a message

Description automatically generated

Figure 1 Results output format

## **Computational Scenarios**

The analysis will be run on three computational scenarios having different processing resources.

* 1 Node and 1 Core – associated with 1 Node /1 Physical computing system only
* 1 Node and 8 Cores– associated with 1 Node /1 Physical computing system only
* 2 Node and 8 Cores– associated with 2 Node /2 Physical computing system only

## **Slurm Script with data parallism**

The code will be run on University HPC system (SPARTAN) environment using workload scheduling by Slurm which is a simple Linux utility for resource management (i.e., ‘srun’ and ‘sbatch’ commands). Refer to Fig2 a case to run job on 2 Nodes /4 Core with total 8 processes.

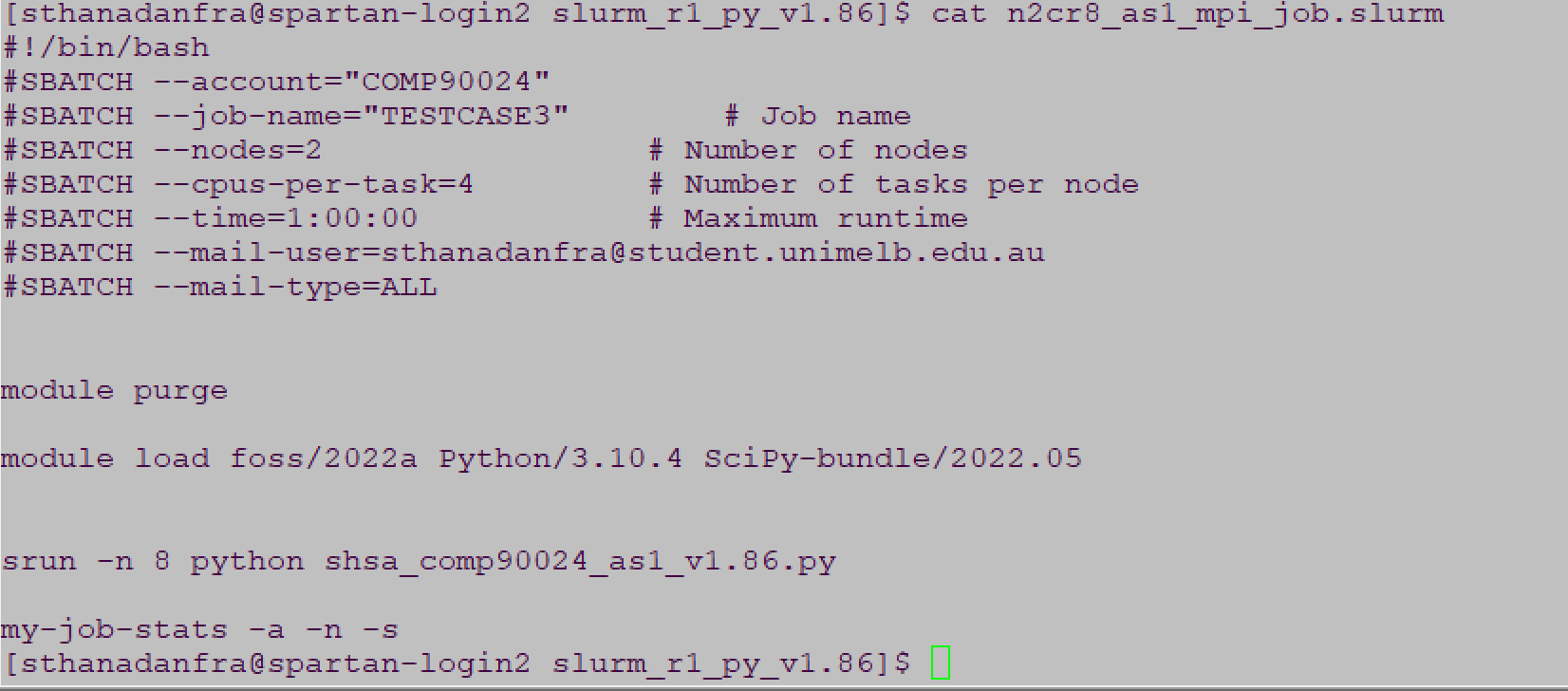


Figure 2 Example of sbatch with srun scripts.

# **Our Approach for the project**

We used multiple approach to optimize the computational objective using different software design approaches, to understand the impact and trade-offs between data set sizes vs resources. Below are the key approaches used in the program to achieve the results.

## **Approach#1: Imputation by data parallelism processing without any summarization technique while parsing the large json file.**

At first stage, we tried to parallelize the data processing by filtering the data of interest (json key & its values) as follows.

1. We sent the file across the node(s) and CPU(s) to process json lines based on file line index mapped to cpu ranking and size with an offset value (i.e, multiple of the sum of cpu rank+size) of the provided environment set by Spartan srun/sbatch). In this case the associated process will parse its processed line to create an array of each relevant json keys ('created\_at’ & ‘Sentiment’) and its associated values. Each processor will send its array to main process (ranking 0) to compile the array for calculation. This approach has resulted to achieve parallel CPU Efficiency at ~98.84%
2. Though, the method ran perfectly for 1MB/50MB test cases, but we got into “Memory crash” issue when we run it for 120GB twitter json file. It was noticed that the granularity level of data parsing based on ‘created\_at’ key value included minutes and seconds while creating the data array hence the memory crash occurred due to large rows of data sets (i.e, the job status shows - Memory Utilized: 35.40 GB (estimated maximum), Memory Efficiency: 113.29% of 31.25 GB (3.91 GB/core)). Refer to Fig3 and 4 for the memory Error and memory utilization while running the job.

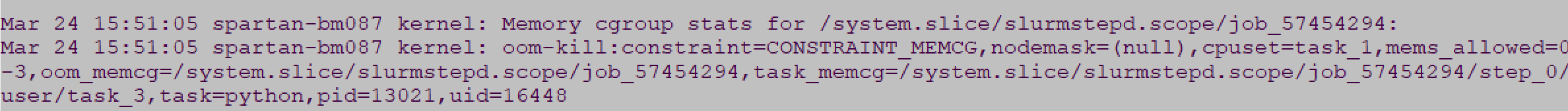


Figure 3 Memory Error Logs for Approach#1

A screenshot of a computer

Description automatically generated

Figure 4 Memory Utilization for Approach#1

## **Approach#2: Imputation by data parallelism processing with summarization technique while parsing the large json file.**

To avoid memory error due to excess utilization, we further optimized our approach by first making an array list of pre-aggregated summary of interested data keys and values to reduce the memory usage before processing it for data analysis as follows:

1. **Data pre-processing based on CPU ranking** : While the parallel processing logic remains same as Approach#1 (i.e, parsing file lines by line index mapped to multiples of the sum of cpu rank+size), using a python function called ***process\_twitter\_json\_file()*** , the json file was processed to aggregate and summarize the data using key index (i.e, ‘Year’, ‘Month’,’Day’ and ‘Hour’ ) to capture the associated tweet Counts and appending the sum of Sentiment values of each tweets belongs to the same key indexes. This approach helped us to reduce the memory usage and process the data more efficiently this case, the data was parsed at granularity level set at hourly basis rather than storing the values at minutes and seconds basis. This has resulted to achieve parallel CPU efficiency at ~ 99.46% and memory utilization ~164.48MB/CPU core (which was around 4.11% of default allocated memory 3.91 GB/CPU core in SPARTAN) by giving room for processing further large file using same code algorithms.

As the memory utilization of the code and algorithm on 120GB twitter json file was only ~164.48MB/CPU , the code was further analysed by running with custom memory allocation in the slurm script (using *#SBATCH --mem-per-cpu=200M* ) to observe the memory utilization increasing ,i.e, >80%.

1. **Data parallism and communication gathering from Multiprocessing:** The distributed dataset (i.e processed json lines based on CPU ranking and size index from 120GB data file ) gets collected at main cpu ( with ranking 0 ) for further consolidating and grouping by key index columns ‘Year’, ‘Month’ and ‘Day’ to calculate the sum of tweets and sentiment scores. The result was getting stored in a data frame (eg. ‘**result\_df)** for further data analysis using a python data function called ***“project\_data\_analysis”*** to meet the assignment objective.

The returned results are shown in Fig 5 and 6.

**A screenshot of a computer

Description automatically generated**

Figure 5 Results of Approach#2 with default memory allocation

**A close up of a message

Description automatically generated**

Figure 6 Results of Approach#2 with custom memory allocation of 200MB per CPU

# **Data results**

Below are the data processing results for each scenario:

## **Data results for 120GB on 1 Node/1Core**

Results for 1 Node and 1 Core

|  |  |
| --- | --- |
| A close-up of a receipt  Description automatically generated | |
|  |  |

Figure 7 Processing results for 120GB on 1 Node/1 Core

## **Data results for 120GB on 1 Node/8 Cores**

Results for 1 Node and 8 Cores

|  |  |
| --- | --- |
| A close up of a message  Description automatically generated | |
| A screenshot of a computer  Description automatically generated | A screenshot of a computer  Description automatically generated |

Figure 8 Processing results for 120GB on 1 Node/8 Core

## **Data results for 120GB on 2 Node /8 Cores**

Results for 2 Node and 8 Cores

|  |  |
| --- | --- |
| A close up of a computer screen  Description automatically generated | |
|  |  |

Figure 9 Processing results for 120GB on 2 Node/8 Core

# **Results comparison**

Below are the data processing results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No | Scenario | CPU  Efficiency % | RAM Used | RAM % Used as per Job status.  (Default Allocation -3.91 GB/core) | CPU Idle Time % | Processing Time  (Secs) |
| 1 | 1 Node / 1 Core | 99.46% | 164 MB | 4.11% | ~0.2% | 1854.812 |
| 2 | 1 Node / 8 Cores | 97.8% | 1.31 GB | 4.2% | ~2.3% | 672.977 |
| 3 | 2 Nodes 8 Cores | 99.01% | 1.31 GB | 4.19% | Node1: ~2.8%  Node2: ~2.0% | 678.497 |

Figure 10 Data analytics results.

## **Processing time**

Refer to Fig 11

Figure 11 Processing time results

From the above charts, it is found:

* As compute resource power increases the time reduces exponentially and no linear relation found.
* As per **Amdahl’s law**  , it is found that the parallelizing computation with 8 processors (N=8) has given speedup (S) improvement gain of 2.75613 (i.e, 1854.812/672.977). Hence ~61% (i.e, α=0.6097) of the program is found to be parallelized and remaining 39% of the code was dependent on main cpu.
* It is also observed that running this data set on a single node architecture has given more optimum results (672.977 secs) than with 2 Node (678.497 secs). However, both are executed almost with same processing time.

## **CPU Utilization**

Refer to Fig 12

Figure 12 CPU Utilization rate

From this chart we deduce the following

* As compute resource power increases ([A.K.A](https://www.oxfordreference.com/display/10.1093/oi/authority.20110803095405686#:~:text=Quick%20Reference,Operating%20system%20X%20AKA%20Crapware'.) CPU cores) in a node the CPU utilization drops
* Scheduling is important decision as we can see for case of 1N/8C and 2N/8C there is slight difference in CPU% , for large data this can be key decision

## **RAM Utilization**

It found the code was using only 164MB/CPU core during program run which was ~ 4.11% to 4.2% as the default memory allocation was ~3.91GB/CPU Core. We did regressive testing to optimize RAM utilization by allocating 200MB per CPU core instead of default allocation to improve the memory efficiency > 80%.

Refer to Fig 13

Figure 13 RAM Utilization rate (Default allocation 3.91GB/CPU Core)

From this chart we deduce the following

* Based on our project execution we found HPC systems are memory intensive hence the program algorithm should be optimized when processing large data sets and files to avoid memory crash (as we observed during Approach#1, the memory was exceeding allocated 3.91GB per CPU while parsing and storing values from 120GB file). Using aggregated array with required key granularity (i.e, ‘Year’,’Month’,’Day’ and ‘Hour’), the memory usage has significantly reduced to 164MB per CPU as shown in Fig 14.

A screenshot of a computer

Description automatically generated

Figure 14 RAM Utilization rate (allocation 200MB/CPU Core)

* For our case we applied data parallelism (SIMD) and RAM utilization was optimized, this is certainly a key learning to plan optimized AI infrastructure

## **Compute Idle Utilization**

Refer to Fig 14

Figure 15 Idle Utilization rate

From this chart we deduce the following

* We can clearly see that more resources mean compute idle time % will increase, we need to optimize it to ensure we get best ROI from our infrastructure.
* Surprisingly adding more nodes do not improve idle % after “N” is reached, in our case 2N/8C is 2.4% idle compared to 1N/8C which is 2.30% idle.

# **Conclusion**

In conclusion, in line with Amdahl’s law, if 95% of the program can be parallelized, the theoretical maximum speedup using parallel computing would be 20×, no matter how

many processors are used. In our program code , we could observe ~61% of the code processing was parallized for computing and ~31% of the code processing was depending on main CPU resulting speedup gain of 2.75613 times (i.e 1CPU Core vs 8CPU Core). Hence the analysis suggests that for CPU-intensive tasks like the one at hand, leveraging multi-core and multi-node configurations can substantially improve processing efficiency and reduce execution times, however beyond certain “N” adding more resources may not help.

#########################################################################################

**References**

1) Use a consistent referencing scheme: <https://library.unimelb.edu.au/recite>

2) SLURM Documentation, URL: <https://slurm.schedmd.com/documentation.html>

3) PYTHON MPI Documentation, URL : <https://mpi4py.readthedocs.io/en/stable/>

4) Python Documentation, URL : <https://www.python.org/>

5) Pandas Python documentation, URL : <https://pandas.pydata.org/>

6) Amdahl’s law , URL :https://en.wikipedia.org/wiki/Amdahl%27s\_law

7) SIMD architecture <https://en.wikipedia.org/wiki/Single_instruction,_multiple_data>

**Appendix**

## **Appendix1 : main () Python Code (shsa\_comp90024\_as1\_v1.86.py)**

A computer screen shot of text

Description automatically generated