Introduction

This report discusses the approach used to implement a parallelized application built in Python to determine the top 10 occurring hashtags and languages (along with their respective counts) used in a large Twitter dataset for Sydney. The application leverages the High-Performance Computing (HPC) facility SPARTAN provided by the University of Melbourne. The application is run on different configurations utilising different resources to solve the problem.

Specifications

Platform: The application is written in Python 3 and uses mpi4Py package to parallelize the application. It is run on 'physical' partition of the SPARTAN cluster. The application runs on bigTwitter.json dataset.

Output: The output of the application is recorded in the following format:

```
Top 10 Hashtags:
('#hashtag1', Count)
('#hashtag2', Count)
Top 10 Languages used:
('Language1', Count)
('Language2', Count)
```

High Performance Computing: The application is run on the following resources in SPARTAN:

- 1 node and 1 core
- 1 node and 8 cores
- 2 nodes and 8 cores (with 4 cores per node).

Application Invocation

Scripts used for submitting the jobs on SPARTAN:

1) Symbolic Link to bigTwitter.json file using the script below:

```
ln –s /data/projects/COMP90024/bigTwitter.json
```

2) Create a new slurm file using nano editor
3) Slurm file for 1 node and 1 core configuration:

```
#!/bin/bash
#SBATCH --time=0-30:0:00
#SBATCH --nodes=1
 #SBATCH --ntasks=1
 #SBATCH --partition=physical
```

```
# Load required module
module load Python/3.5.2-intel-2017.u2
time mpiexec -n 1 python Assignment1.py - bigTwitter.json
```

4) Slurm file for 1 node and 8 cores configuration

```
#!/bin/bash
#SBATCH --time=0-30:0:00
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --partition=physical
# Load required module
module load Python/3.5.2-intel-2017.u2
time mpiexec -n 8 python Assignment1.py -i bigTwitter.json
```

5) Slurm file for 2 nodes and 8 cores configuration

```
#!/bin/bash
#SBATCH --time=0-30:0:00
#SBATCH --nodes=2
#SBATCH –ntasks-per-node=4
#SBATCH –ntasks=8
#SBATCH --partition=physical
# Load required module
module load Python/3.5.2-intel-2017.u2
time mpiexec -n 8 python Assignment1.py -i bigTwitter.json
```

6) Script to submit /queue Slurm jobs

```
sbatch assignment1.slurm
```

Methodology

- **Parallelization Technique and logic**

The technique we have used to achieve parallelism is ***SPMD (Single Program, Multiple Data)*** in which there are multiple processes that execute parallelly on same code but utilizes different parts of the data. Each node executes the program and communicates the results with the master node by sending and receiving messages. This communication technique is called **Master worker/ Slave model**. In the technique, the master node gathers the partial results from all the slave nodes and computes the final result.

- **Program Flow**
    - **Parallel File Reading**

The bigTwitter.json dataset is extensively large (more than 20GB) and reading it sequentially would consume a lot of resources and memory. Therefore, we have split the dataset according to the number of nodes (given through Get_size() function of the class MPI_COMM_WORLD) and each process is designated a unique rank using the Get_rank() function. Therefore, each node will be reading and processing (Total data/number-of-nodes) amount of data parallelly. The dataset is read as a dictionary.

- **Parallel Data Processing**

Each node executes the logic to compute two lists (Hashtags and its count; Language codes and its count) on a part of the total dataset that is read by that node. The Hashtags are extracted by traversing to the “Entities” key in the dataset. Similarly, the Language codes are extracted from the “lang” key in the dataset.

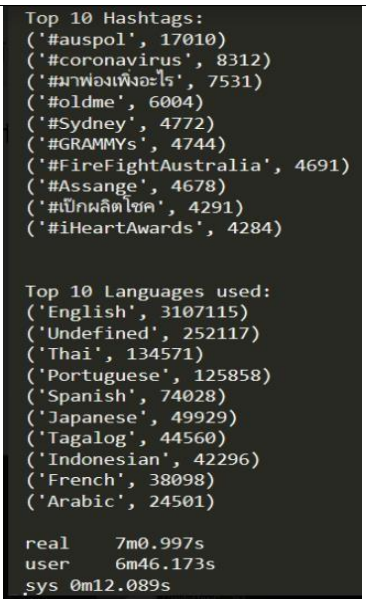
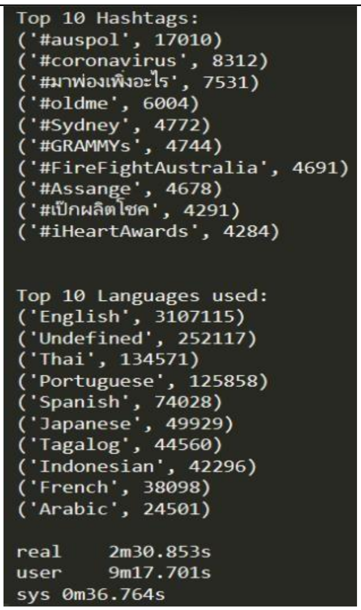
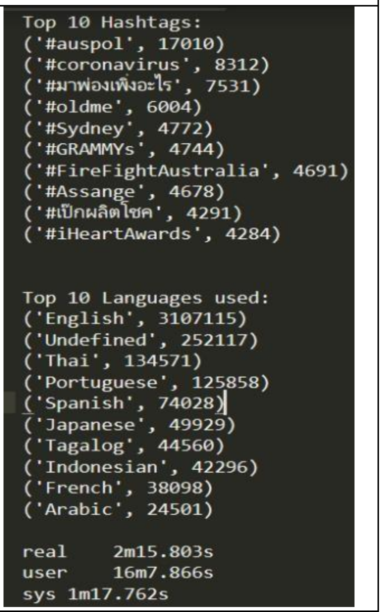
- **Merging results from different parallel processes**

All the ranks except the master rank (rank 0) sends their results to the master rank by using the **mpi.send()** function of the MPI library. The master rank receives the parallelly processed results from all the nodes using **mpi.recieve()**.

The master node computes the final list of Hashtags and Language codes by merging the individual lists received from each node. It calculates the total count of Hashtags and Tweets of different languages, sorts the compiled result and filters the top 10 Hashtags and Languages used it tweets.

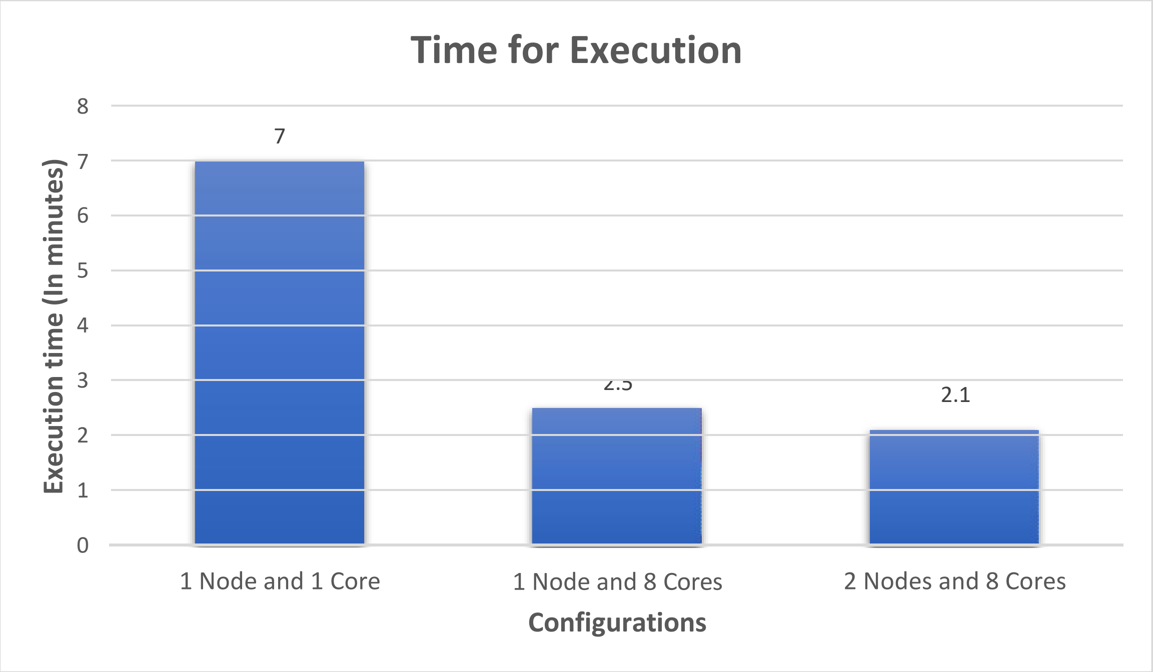
Results

The results of running the application on 3 configurations are given below**:**

Configuration	**1 node- 1 core**	**1 node- 8 cores**	**2 nodes- 8 cores**
Time Taken	**7 mins**	**2 mins 30 seconds**	**2 mins 15 seconds**
Output Screenshot	 Top 10 Hashtags: ('#auspol', 17010) ('#coronavirus', 8312) ('#มาพ่องเพิ่งอะไร', 7531) ('#oldme', 6004) ('#Sydney', 4772) ('#GRAMMYs', 4744) ('#FireFightAustralia', 4691) ('#Assange', 4678) ('#เป๊กผลิตโชค', 4291) ('#iHeartAwards', 4284) Top 10 Languages used: ('English', 3107115) ('Undefined', 252117) ('Thai', 134571) ('Portuguese', 125858) ('Spanish', 74028) ('Japanese', 49929) ('Tagalog', 44560) ('Indonesian', 42296) ('French', 38098) ('Arabic', 24501) real 7m0.997s user 6m46.173s sys 0m12.089s	 Top 10 Hashtags: ('#auspol', 17010) ('#coronavirus', 8312) ('#มาพ่องเพิ่งอะไร', 7531) ('#oldme', 6004) ('#Sydney', 4772) ('#GRAMMYs', 4744) ('#FireFightAustralia', 4691) ('#Assange', 4678) ('#เป๊กผลิตโชค', 4291) ('#iHeartAwards', 4284) Top 10 Languages used: ('English', 3107115) ('Undefined', 252117) ('Thai', 134571) ('Portuguese', 125858) ('Spanish', 74028) ('Japanese', 49929) ('Tagalog', 44560) ('Indonesian', 42296) ('French', 38098) ('Arabic', 24501) real 2m30.853s user 9m17.701s sys 0m36.764s	 Top 10 Hashtags: ('#auspol', 17010) ('#coronavirus', 8312) ('#มาพ่องเพิ่งอะไร', 7531) ('#oldme', 6004) ('#Sydney', 4772) ('#GRAMMYs', 4744) ('#FireFightAustralia', 4691) ('#Assange', 4678) ('#เป๊กผลิตโชค', 4291) ('#iHeartAwards', 4284) Top 10 Languages used: ('English', 3107115) ('Undefined', 252117) ('Thai', 134571) ('Portuguese', 125858) ('Spanish', 74028) ('Japanese', 49929) ('Tagalog', 44560) ('Indonesian', 42296) ('French', 38098) ('Arabic', 24501) real 2m15.803s user 16m7.866s sys 1m17.762s

Performance Comparison

The comparison of execution time for all the three configurations is given below:



Reason for Performance Variation

From the above results, the execution time follows the trend: **2 nodes and 8 cores configuration** (**with 4 tasks allocated to each node) < 1 node and 8 cores < 1 node and 1 core configuration**.

The reason for better performance on 2 nodes and 8 cores is because distributing the processing load over multiple nodes drastically reduces the overall load. Thus, Horizontal Computational Scaling (i.e. parallel processing of the application by running the same code on multiple nodes using different parts of the data) ensures that the overall performance is increased by parallel processing whilst utilising the shared resources by all nodes. Furthermore, the processing time of two nodes and one node configuration is quite comparable because further parallelising the application on two nodes increases the communication overhead between the processes and affects its overall execution time.