Cluster and Cloud Computing Assignment 1

HPC Twitter Processing

Yaoyi Chen 970325 yaoyic@student.unimelb.edu.au

Hangyu Pan 1050937 hjpan@student.unimelb.edu.au

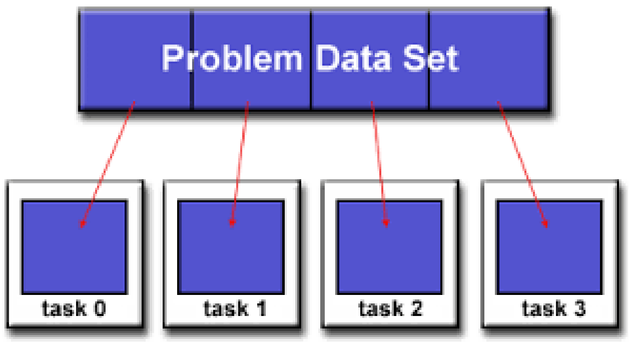
Introduction

This assignment is to search the large Twitter data set and identify the top 10 most commonly used hashtags, languages as well as the number of times they appear. Due to the large size of data, the process should use parallelized approach; which is MPI in this assignment, and implement the application on HPC facility SPARTAN.

Generally, as bigTwitter has around 20GB size, it can implement the iterator to cope with each data by lines and process data through method of map reduce. However, it does take more than 3 mins in 8 cores. To improve the efficiency of computing power, the report is going to deliver the approach of multi-core parallel computing.

Approach of multi-core parallel computing

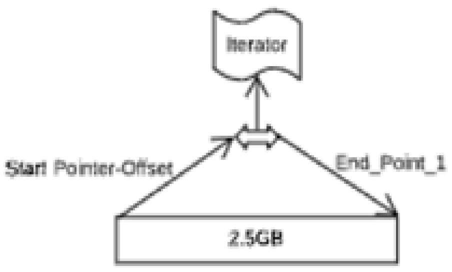
With the design of parallelizing in this project, firstly, each process can be set to its part of similar

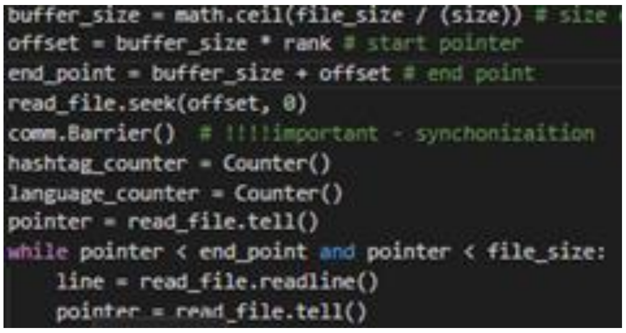


work in the parallelizing part. Secondly, the data can be exchanged through the stable and reliable protocol within all processes. Thirdly, the program has to ensure that it will not run out of memory and try to get a more efficient way of memory use. To parallelize code, it implements two significant techniques which are MPI and the Pointers to read files.

To set part of the task for each process, the file is divided into several pieces with the same size depending on the number of processes/cores and the file size. Here are 8 cores and an around 20GB dataset in json. Therefore, each process is arranged to cope with individual 2.5 GB tasks respectively.

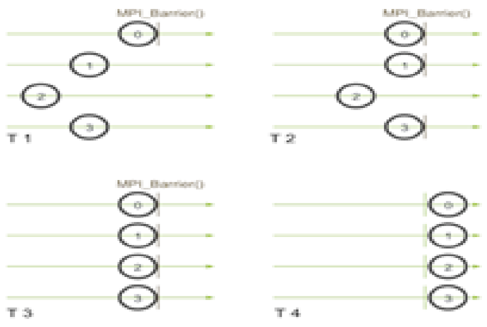
For each process, although it handles various parts of the file, with the design of parallelizing, it sets to do the same jobs before the collection of information. The pointer implemented here is to help each process identify their own parts of the file allocated to its job. As the code and the graph show, offset can be considered as the start pointer of parts of file and end point is to reflect the end of reading file task and parsing data

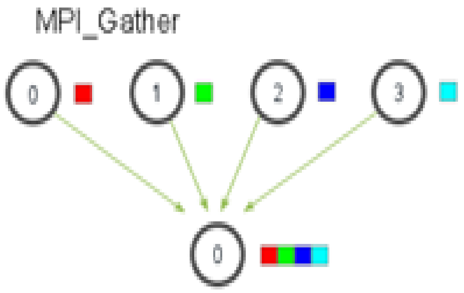



```
buffer_size = math.ceil(file_size / (size)) # size
offset = buffer_size * rank # start pointer
end_point = buffer_size + offset # end point
read_file.seek(offset, 0)
comm.Barrier()  # !!!!important - synchonizaition
hashtag_counter = Counter()
language_counter = Counter()
pointer = read_file.tell()
while pointer < end_point and pointer < file_size:
    line = read_file.readline()
    pointer = read_file.tell()
```

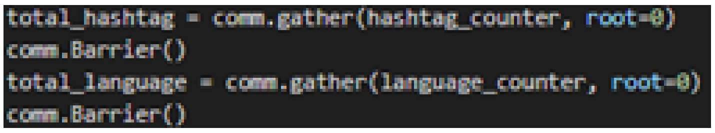
task. Through the method of “readline” regarded as an iterator to load data from part of file into memory, on one hand, it helps better parse the data, on the other hand, it ensures that the program does not run out of memory instead of loading the whole part of file.

In addition, the “comm.Barrier()” is to support synchronization among processes. It ensures the process can reach the same point before the next step when parallelizing.



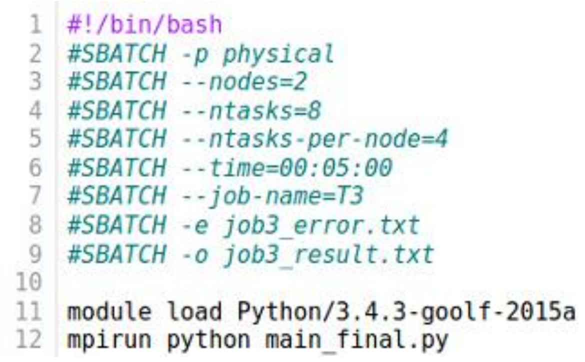


After parallelizing tasks are all completed among processes, the processed data from each process will be gathered and sent to the root (the root here is rank 0) for the collective information purpose. Finally, through the root process, the collective information will be delivered.


```
total_hashtag = comm.gather(hashtag_counter, root=0)
comm.Barrier()
total_language = comm.gather(language_counter, root=0)
comm.Barrier()
```

SPARTAN implementation

The SPARTAN implementation codes show as below.



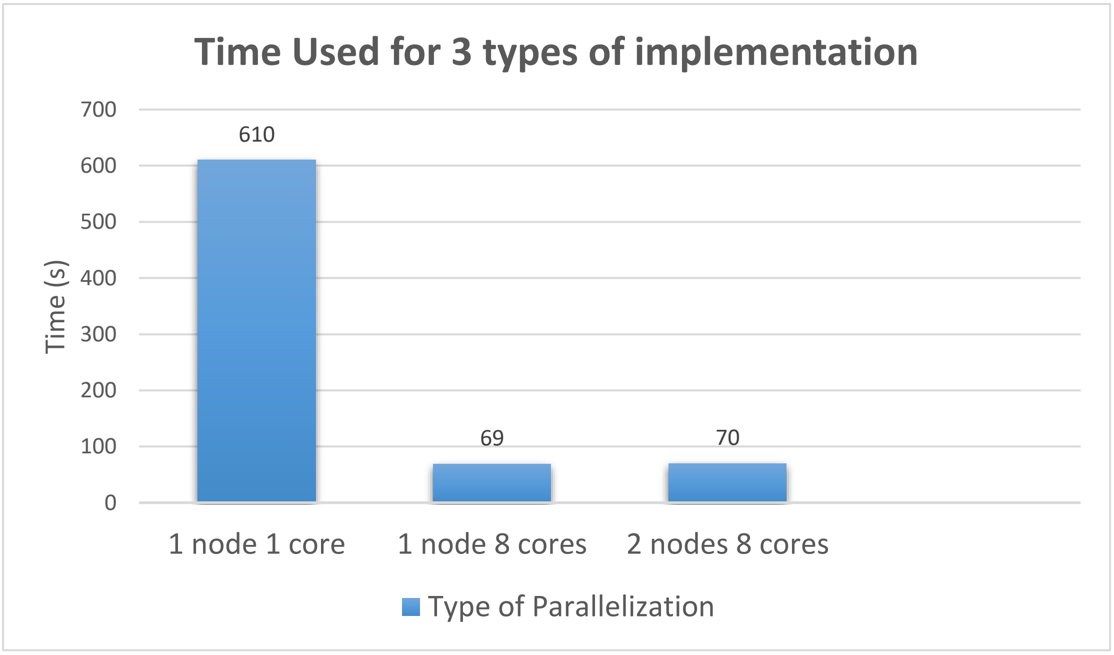
```
#!/bin/bash
#SBATCH -p physical
#SBATCH --nodes=2
#SBATCH --ntasks=8
#SBATCH --ntasks-per-node=4
#SBATCH --time=00:05:00
#SBATCH --job-name=T3
#SBATCH -e job3_error.txt
#SBATCH -o job3_result.txt

module load Python/3.4.3-goolf-2015a
mpirun python main_final.py
```

This is an example of using MPI to run a python script on physical SPARTAN with 2 nodes, 8 cores and 4 cores in each node under a specific python version.

As the script shown, the task or job called T3 is going to run under the partition with 8 cores and 2 cores (4 cores on the 2 nodes). The error output will be saved in job3_error.txt and the result is in job3.result.txt. The wall time is in 5 mins.

Comparison

The time used for 3 types of implementation show as below graph.



It is very obvious to see the efficiency of parallelization. In this project, both multi-core and multi-node approach can reduce processing time about **88%**. However multi-node approach is a litter bit slower than just using multi-core approach. Because the process communication among different nodes is slightly slower than communication within a single node.