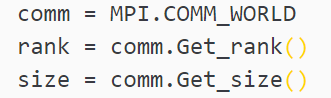
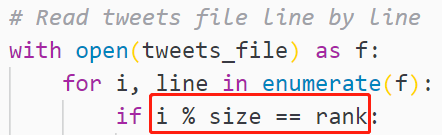
The implementation of our application

(The code follows the idea of the given example in the workshop7-8.1)

This application uses **point to point communication** (supported by MPI/mpi4py module) to achieve the parallelization. The main idea is, if there are n processes running, each process should only process tweets and let one of the processes (master) gather all the result from other processes (slaves).

Each process would know how many processes there are and whether it is a master or a slave through:

Every process uses serial numbers generated during file reading, processes size, and its rank to determine if this tweet belongs to it to process:



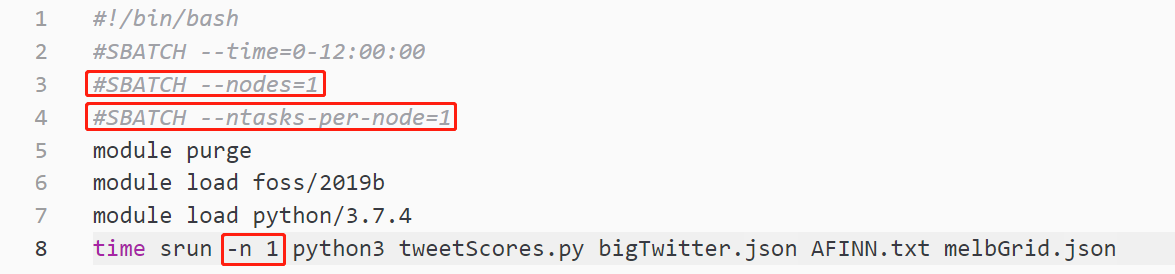
Once the master process finished its own task, it would gather results from other slave processes by sending a ‘return\_data’ signal to all other slave processes and kill them by sending an ‘exit’ signal when it receives all results successfully.

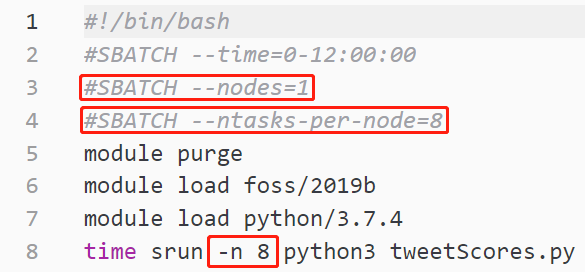
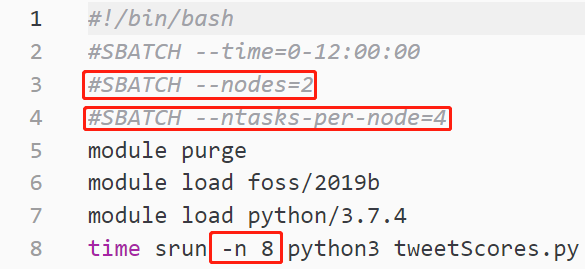
The whole process can be generalized as:

|  |  |  |
| --- | --- | --- |
| Sequence | Master | Slave |
| 1 | Get rank, size, and process tweets | Get rank, size, and process tweets |
| 2 | Send ‘return\_data’ signal | Receive ‘return\_data’ signal |
| 3 |  | Send processed result |
| 4 | Receive and process all the results |  |
| 5 | Send ‘exit’ signal, Print results | Receive ‘exit’ signal |
| 6 | Finished | Finished (exited) |

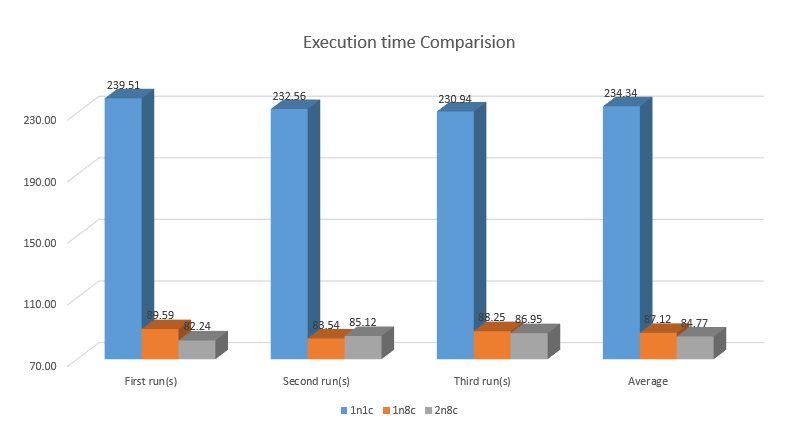
Spartan submitting scripts.

The application should be running under the environment of foss/2018b and python/3.7.4, with number of nodes and cores being specified separately in three .slurm files.



Performance analysis:



It will take 234 seconds to run this program on 1n1c, and it takes about 87 seconds and 85 seconds to spend on 1n8c and 2n8c respectively, which is about one third of 1n1c.

We can find that using 8 cores does not increase the processing speed by 8 times. This is because even if 8 cores are used to run the program, each core still needs to traverse 20g of data completely although only one-eighth of it needs to be loaded, while the single core only needs a complete traversal and loading. And due to the limitations of MPI, the master needs to receive the results in order, and some cores may be slow to process, which will cause the overall time delay (Cannikin's law).

The results of 1n8c and 2n8c are similar, because they are basically the same except for the number of computers that make up them.