**Modifying Metrics Collection Script**

**Last week's task:**

1. **Collecting BWA-mem system metrics from *top* command**

**Next week's task we will collect system metrics of:**

1. **Samtool**
2. **Picard SortSam**
3. **Picard MergeSamFiles**
4. **Picard MarkDuplicates**
5. Overview

In the previous report it was discussed that there was a problem with Slurm, where there were missing values in the memory usage job which elapsed time was less than 30 seconds. To follow up on this issue, system metrics collection is performed using the top command. However, the script for collecting collection metrics needs to be modified. To speed up the modification process, I chose the Python programming language because it is more familiar than using a shell script. And now the program is still running so there is no data analysis that can be displayed.

1. Script

One of the main new methods in the metrics collection script is generate\_packing\_combination as shown in Figure 1. The goal is that the program can directly control the size of the pack so that it can record the size in a file. This method will receive input in the form of a list of lists of variant metric values. In this case there are 3 types of metrics, which are number of threads, memory allocation, and cpu allocation. Then, this method will return output in the form of a combination of how to pack jobs into an instance in the form of a list. This list contains several packs of sets of arguments. Each pack contains several sets of arguments. One set of arguments contains 3 values for the parameters threads, memory allocation and cpu allocations which will become one job.

The way to produce this output is to generate all possible sets of arguments. Then the list of every possibility will be randomized so that each set of arguments is not only associated with the same pack size continuously. This is related to the algorithm used to generate all possible packs. The algorithm is quite simple, the program will iterate over a list of all possible sets of arguments and make several sets of arguments into a pack so that the total memory allocation and cpu allocations of the pack do not exceed the capacity of the instance.

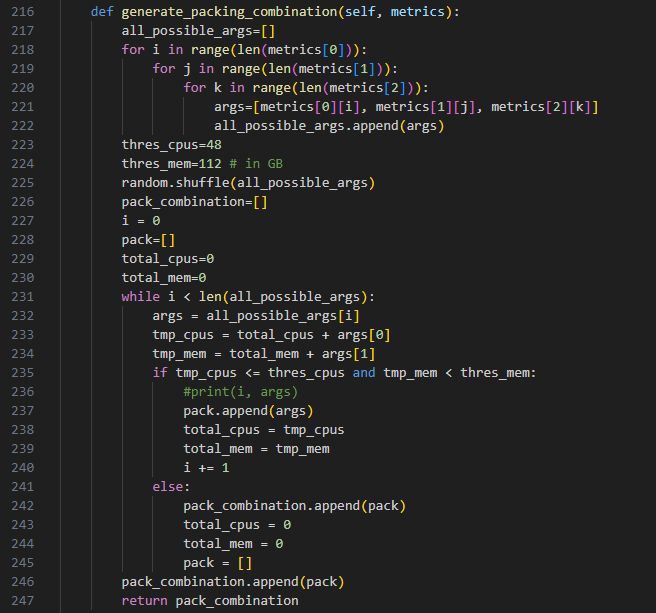


Figure 1. Generate packing combination method.

For each submitted job, resource usage data will also be collected using the *top* command (Figure 2). This was done because there was a problem with Slurm that it could not capture memory usage if the job elapsed for under 30 seconds. Data collection is done by running the command on the instance that processes a job remotely. Based on the information about the job pid of a job id (id from Slurm) obtained from the *scontrol listpids* command, the master node will execute 'top -b -p {job\_pid} -n1 | tail -1' on the node that processes the job pid and produces output in the form of 1 row of resource usage of a job in a particular time.

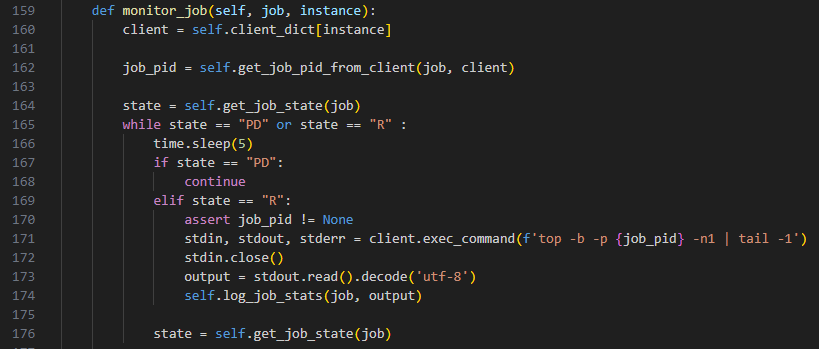


Figure 2. Monitor job method to collect system metrics from *top* command

1. Result

The result of this program is a file that contains a collection of resource usage snapshots for each job (Figure 3). As mentioned earlier, we also record the pack size of the batch where the job originates. This number is compiled in column n\_pack. Not only that, there is an additional column in the form of pack\_id which aims to provide information on other jobs that are packed with that job. It is possible that there is 1 other job that has failed so that what was previously an n\_pack with a value of 3 becomes 2.

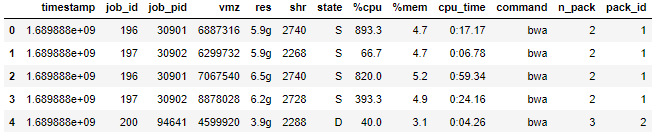


Figure 3. Resource usage snapshots of jobs.

So far, the program is still running to collect data points. Each combination of arguments is executed 10 times. The combination of arguments is a combination of the values in Table 1. So the number of combinations is 10 \* 5 \* 5 \* 4 = 1000. And the number of executions is 10 \* 1000 = 10000. Thus, we will get 10,000 data points, whether they are completed jobs or failed jobs.

| SRA files | 1. SRR2059426 2. SRR2059427 3. SRR2059428 4. SRR2059429 5. SRR2059430 6. SRR2059431 7. SRR2059432 8. SRR2059433 9. SRR2059434 10. SRR1658393 |
| --- | --- |
| threads | 2, 4, 8, 16, 32 |
| CPU allocation | 2, 4, 8, 16, 32 |
| Memory allocation | 8, 16, 32, 64 |

Table 1. Variation in the value of the job execution parameter