BioHPC workshop "Parallelization and load balancing": Exercises Part 2

This part of the workshops is devoted to using SLURM scheduler, especially as a tool for parallelizing and load-balancing of multiple independent tasks. BioHPC offers a 'SLURM on demand' service that allows any user to spin up a simple instance of SLURM cluster on one or more machines the user has reserved through the BioHPC reservation calendar. This simple procedure is described in https://biohpc.cornell.edu/lab/userguide.aspx?a=software&i=689#c. Unfortunately, since one machine can only be part of one SLURM cluster, it will not be possible for everybody to practice the procedure in the workshop environment, where each machine is shared by multiple users. Instead, we have created a SLURM cluster ahead of time, consisting of all workshop machines, with access given to all participants. The name of the cluster is `cbsum1c2b002` - same as the machine where the SLURM control daemon `slurmctld` is running.

For all SLURM-related exercises it will be convenient to create a dedicated folder in your home directory (not /workdir this time!).

```
cd
mkdir slurm_test
cd slurm_test
```

Smooth operation of SLURM requires requires a 'working directory' present on all cluster nodes, and a subfolder of home directory, network-mount on all nodes, is the obvious choice. Note that this does not mean that the actual computations will be performed there. We will use this directory to store scripts, submit jobs, and collect results. Copy the sample SLURM submission scripts from the workshop directory

```
cp /shared_data/Parallel_workshop/*.sh .
```

Exercise 1: BLAST as a SLURM job

Remember the BLAST exercise (Exercise 2) from part 1? We will now submit our BLAST search to a SLURM cluster. Open the script `blast_slurm.sh` in a text editor and carefully examine its content and comments.

Submit this script (with `/workdir/your_id/slurm_test` as your current directory), requesting your choice of number of threads and memory, e.g.,

```
sbatch -N 1 -n 4 --mem=2G blast_slurm.sh
```

Run `squeue` or `squeue_l` to see the status of the job. What is its job ID? Is it still pending? Is it running? If so, on which machine? Do you see the job's output file (STDOUT + STDERR combined) among your files? It should contain outputs from all echo and date commands scattered throughout the script. This may help you figure out which stage the job is currently at. Periodically run `squeue` and watch for output files to catch when the job ends.

Exercise 2: BWA alignment as a SLURM job

Now use the script `bwa_slurm.sh` to submit a BWA alignment job (Part 1, Exercise 3). Take a look at the script in a text editor and make sure you understand its content and submit with options of your choice. Note that the most important SLURM options are already requested in the #SBATCH header lines. If you want to use those options, the submission command is simple:

```
sbatch bwa_slurm.sh
```

You can also modify some of these parameters without changing the script, e.g.,

```
sbatch --ntasks=4 bwa_slurm.sh
```

will run run the job using 4 threads rather than 8 requested via `#SBATCH --ntasks=8`. After submitting, check `squeue` to find out where the job is running. Log in to that machine via ssh and start `htop` (`-u your_id` option will be useful). Is the job (i.e., all its processes combined) using all allocated resources? Le the job run to completion.

Now edit the script and change the number of threads in '`-t`' option of bwa to `8`. Submit the job again, but requesting a smaller number of cores, e.g., 2:

```
sbatch -N 1 -n 2 --mem=2G bwa_slurm.sh
```

Again, log in to the node where the job is running (check it first with `squeue`), and fire up `htop` to examine your processes. In `htop`, press `H` and `t` keys to toggle the tree and thread view. How many `bwa` threads are running? How much combined `%CPU` are they all consuming? What is the total CPU and memory consumption of your job, as reported by `htop`? Are the core and memory limits imposed by SLURM being obeyed?

Exercise 3: Multiple jobs vs job arrays

The simple script gzip_slurm.sh takes one of the files BBB_1, BBB_2, BBB_3 from the workshop directory, compresses it using gzip, and copies the result to $HOME/slurm_test (examine the script in the text editor to verify this). The script takes one argument, an integer, intercepted as $1, which it uses to construct the file name. All important SLURM options are given in #SBATCH header lines. Submit three separate jobs to compress the three files, one right after another, e.g.,

```
sbatch gzip_slurm.sh 1
sbatch gzip_slurm.sh 2
sbatch gzip_slurm.sh 3
```

Instead, you could use the shell's for loop:

```
for i in {1..3}
do
sbatch gzip_slurm.sh $i
done
```

Right after submitting the three jobs, use `squeue` to verify their status. As any of the jobs complete, the corresponding compressed file will appear in your submission directory.

Instead of submitting 3 separate jobs using three separate `sbatch` commands, the process may be simplified with the help of SLURM's job array mechanism. The script `gzip_array.sh` is a slight modification of `gzip_slurm.sh`, utilizing the SLURM-provided environment variable `SLURM_ARRAY_TASK_ID` to parametrize the input file instead of the external script argument. This variable is defined within a SLURM job if the `--array` option is used at submission:

```
sbatch --array=1-3 gzip_array.sh
```

will have the same effect as the three separate `sbatch` commands above, i.e., will result in three separate jobs being submitted. In `squeue` output, these jobs will be showed collectively in one line when in pending (PD) state, and separately on different lines when in running (R) state. Their job IDs will have the form similar top `1234_5`, where `1234` is the job ID of the first job of the array, and `5` is the index of the array.

```
sbatch gzip_slurm.sh 2
```

Exercise 4: Examine jobs' accounting information

Use `sacct` and its longer form `sacct_l` commands to sow information on jobs you just submitted. What can you say about the efficiency of each job? Were the number of cores and memory requested adequate?