Clusters

From CCBR IT Wiki

Contents

Brief introduction to our clusters

Penguin cluster - Mixed CPU/GPU nodes - Setting Up

- **Slurm cluster system:** Use sbatch, srun, salloc, scancel, sinfo, squeue, scontrol to manage jobs.
- **Cluster partitions (sub clusters):** "cc" & "gc" for CPU & GPU partitions, default is "cc", if you don't specify partition when submitting jobs, CPU nodes will be assigned, and you MUST specify "gc" partition when submitting GPU jobs.
- **Master node:** penmaster.ccbr.utoronto.ca, **VM, DO NOT RUN CPU/MEMORY HEAVY LOAD PROCESS ON IT.**
- **CPU nodes:** cc[01-50].ccbr.utoronto.ca, **4 nodes cc[01-04]** are configured and available now.
  - **Hardware:** 60 Lenovo SD530 servers, each has 2xIntel Xeon Gold 6148 CPU Cores, 80 vCores, 2x20Cx2T @2.4-3.7GHz, 224GB DDR4 2666 MT/s memory.
  - **OS & System App:** Rockylinux9, R, Bioconductor, Miniconda3, localcolabfold, chrombpnet, Boltz-1/Alphafold3, cellprofiler4, Singularity, etc.
- **GPU nodes:** gc[01-12].ccbr.utoronto.ca, **3 nodes gc[01-03]** are configured and available now.

- **Hardware:** 12xIBM Power9 AC922 servers,each has 4xV100 32GB GPUs,2xPower9 CPU,128 vCores,2x16Cx4T@2.3-3.8GHz,256GB DDR4 2666 MT/s CPU memory.
- **OS & System App:** Almalinux8 ppc64le, nvidia driver 550, cuda-toolkit 12.4, cudnn9, conda 25, python3.11
- **Note:** Nvidia stops supporting ppc64le from RHEL9, RHEL8 and installed apps are the most recent version for our hardware.
- **Bio Research & Machine Learning Software use GPUs:**
  - **Conda channel for ppc64le:** https://anaconda.org/rocketce/repo
  - **pytorch (2.4.1,2.5.1,2.7.0), boltz1(Alphafold3) and their dependencies:** compiled for PowerPC CPU by Matej Usaj and installed.
  - **tensorflow-2.14.1-cuda12.2_py311_1 installed from rocketce conda channel.**
  - **localcolabfold, chrombpnet:** will do research and try to compile and install them on PowerPC CPU node gc01.
  - **Cellpose:** Cell and nucleus segmentation (https://github.com/mouseland/cellpose)
    - Installation instruction: Studied and tested by Myra Masinas,
      - Clone the repository, git clone https://github.com/BooneAndrewsLab/cellpose.git
      - cd cellpose && git checkout v3.1.1.1-ccbr
    - Setup conda environment
      - conda env create -f cellpose-3.1.1.1_environment.yml
      - conda activate cellpose
      - pip install .
    - Example Usage: cellpose --use_gpu --gpu_device 0 --dir /home/myra/cellpose_test/input/Screens_Cdc11 --savedir /home/myra/cellpose_test/output/Screens_Cdc11 --diameter 70 --chan 2 --chan2 2 --flow_threshold 0.0 --cellprob_threshold -1.5 --pretrained_model cyto --save_tif --save_outlines --no_npy --verbose

DC (Donnelly) CPU Cluster

- **Please SUBMIT JOBS to cluster with sbatch, srun, salloc whenever possible**
- **To CHANGE PASSWORD, run "passwd" on MASTER node DC10, new password will be propagated to all nodes within one hour**
- **Hardware:** 8 nodes, 544/2176 CPU Cores/Threads, 1.6TB ram, good for large number of simultaneous jobs, max 180GB ram for one job.
- **OS & Cluster server** Rocky Linux 9.4 (RHEL9.4) and Slurm cluster
- **Local Storage:** 28TB SSD, 8 partitions, /home[2,3,5,6], /scratch[1,4,7,8]
- **Scratch Storage:** available for everyone per request with reasonable time line
- **Slurm scripts:** sbatch, srun, salloc, scancel, sinfo, squeue, scontrol to manage jobs
- **PBS compatible scripts:** qsub, qstat, qdel, qhold, qalter, qrerun, qrls, pbsnodes
- **Master node:** dc10, DO NOT RUN CPU/MEMORY INTENSIVE processes on master, it is a VM.
- **Slave nodes:** dc[01-08], recommended to login & work on slaves
- **Login session limit:** Users are restricted to 2 ssh login sessions on each node
- **Apps:** list of some installed
  - R-4.4.2 https://www.r-project.org/
  - Bioconductor-3.19 https://www.bioconductor.org
  - Boltz-1/Alphafold3 https://github.com/jwohlwend/boltz
  - Localcolabfold/Alphafold2.3 https://github.com/YoshitakaMo/localcolabfold
  - Chrombpnet https://github.com/kundajelab/chrombpnet
  - Rstudio Server https://posit.co/, access at http://dc0N.ccbr.utoronto.ca
  - Singularity https://github.com/sylabs/singularity
  - Cellprofiler4 https://cellprofiler.org/
  - Python-3.9 (System) and 3.12 (Conda) https://www.python.org/
  - Miniconda https://docs.anaconda.com/miniconda/
  - Perl-5.38.2 https://www.perl.org/

    - BioPerl-1.7.8 https://bioperl.org/
    - Go-1.23.2 https://go.dev/
    - Aws-cli & google-cloud-cli https://aws.amazon.com/cli & https://cloud.google.com/sdk
    - Java openjdk https://www.java.com/
    - Samtools, Bcftools, Htslib, Parallel
- **Note:** localcolabfold is a better version of cloned Google Alphafold, both localcolabfold and chrombpnet run faster with GPUs, but we don't have GPU on DC cluster. However, since there are so many cores/threads (68/272) on each node, we can take advantage of running dozens of concurrent jobs, although each individual job maybe slow, but time to finish many jobs may be shortly than run one by one on GPU server. I don't have the data and skills to fully test them on the new DC cluster, they don't have problem in my simplest test run. Hope some users can test them.

RC (Rocky) CPU Cluster

- Rocky CPU cluster has 17 nodes, 488 Cores, 1.8TB ram. OS is RockyLinux8.10(RHEL8.10)
- **To CHANGE PASSWORD, run "passwd" on MASTER node rc01, new password will be propagated to all nodes within one hour**
- Master node: rc01.ccbr.utoronto.ca
- Other nodes:
    - rc[02-12].ccbr.utoronto.ca
    - rc14.ccbr.utoronto.ca
    - grendel.ccbr.utoronto.ca
    - rao.ccbr.utoronto.ca
    - banting.ccbr.utoronto.ca **(decommissioned)**
    - best.ccbr.utoronto.ca **(decommissioned)**
- **Note:** Support (not recommended) jobs with huge memory (<480GB) and good for common jobs too.
- Apps includes Python3.9, R4.4, Conda23, Cellprofiler3 & 4, Java1.8, Bioconductor3.19, BioPerl1.7.8, Wine, etc
- Rstudio server installed on two nodes, http://grendel.ccbr.utoronto.ca and http://banting.ccbr.utoronto.ca.
- Useful scripts submitjob, jobstatus, nodes, nodes.pbs, qstat, qdel, qsub, tracejob(only on rc01), pbsnodes.
- Storage is same as BC cluster, mounted remote NFS servers.

Boltzmann GPU cluster

- Hardware: 40 CPU Cores/Threads, 128GB Ram, 4 GPUs, each has 12GB Ram.
- **To CHANGE PASSWORD, run "passwd"**
- Software: Ubuntu 22.04 LTS, Cuda-Toolkit-12.5, Cudnn-9.2, Python3, R4, Conda, Alphafold2, Cellprofiler4, Java1.8, Bioconductor3.15, BioPerl1.7.8
- Slurm cluster server, scripts include "sbatch", "srun", "salloc", "scontrol", "sinfo", "squeue", "scancel", "sacct".
- OpenPBS compatible cluster commands are installed, qsub, qalter, qdel, qstat, pbsnodes, etc.
- DO NOT manually run long & CPU/Memory intensive job, always submit job to cluster.
- Sample GPU job script file can be found at /etc/slurm/GpuJob.sh.sample
- ONLY submit GPU jobs, use other clusters for CPU jobs.
- Alphafold with GPU is supported, sample commands listed below.
    - conda activate alphafold
    - cd /usr/local/alphafold/
    - ./run_alphafold -d /home/home2/alphafold/ -o ~/alphafold/dummy_test/ -f ./example/query.fasta -t 2020-05-14

BC (Banting) Cluster - Decommissioned on Feb 3, 2023

- Banting cluster (to be decommissed soon): 8 nodes, 128 Cores/Threads, 800GB ram, lab owned and shared storage (1+PB), OS is CentOS7 (RHEL7)
- Master node bc.ccbr.utoronto.ca

Beagle Slurm Cluster

- Private Kim lab cluster with 16 Nodes including 2 GPU nodes.
- Master node beagle04.ccbr.utoronto.ca

Gold Cluster

- Private Greenblatt lab cluster, 10 Nodes (5 active) with total 240 (120) CPU Cores/Threads, 480GB (240GB) ram, 20TB disk.
- Master node gold.ccbr.utoronto.ca

Rao Cluster - Decommissioned

- Former Morris lab Single server cluster with 64 CPU Cores/Threads, 256GB ram, 24TB disk, to be reconfigured and add to RC cluster.

Silicon Cluster - Decommissioned

- All 32 slave nodes are decommissioned, master node used as 6.5TB storage server for Emili lab

Best Cluster - Decommissioned

- All nodes are Decommissioned, 24TB storage will be decommissioned too

Descartes Cluster - Decommissioned

- All 28 nodes are Decommissioned and removed, master node to be used for other purpose

Cluster scripts and job policy

Outside queue - Manually run jobs

Manually run jobs are not recommended and should only happens if you have difficult to submit particular jobs to cluster system (PBS) or quick testing your jobs. Please be informed that no manually run CPU/Memory intensive jobs allowed on management nodes BC/BC2, they will be killed upon noticed. If you need to run these jobs manually, please login to less busy nodes (run "nodes") and run them there.

Queue setup

Jobs submitted to the BC (a.k.a banting) and DC clusters are routed to one of three queues based on their requested run time. The 'long' queue is for jobs that require between 24 and 1000 hours to complete, the 'medium' queue is for jobs taking between 1 and 24 hours and the 'short' queue is for jobs taking less than one hour. The purpose of having three queues rather than one is to set upper limits to number of long-running jobs so that they do not block the cluster for a long time. For example, in the configuration as of April 2010 there are ~400 available processors

but only 230 of those are available for jobs in the 'long' queue (only 180 when the cluster is heavily used). The total number of jobs in the medium and long queues together also cannot exceed 350, so that 50 cpu's are always available for short jobs taking less than one hour. Note that the various job limits can be different in the current cluster setup. When submitting your jobs to the cluster you should estimate your job run time as accurately as possible. Shorter run times will mean that jobs will get scheduled earlier when the cluster is very busy and you will be able to run more jobs at the same time.

Cluster fairness

When the cluster is not occupied, each user can run up about 80 jobs (processor equivalent) on BC cluster and about 250 jobs on DC cluster. When more people are using the cluster at the same time, the scheduler will try to ensure that each user gets an equal share (fairshare) of the available processors. However, the 'fairshare' percentage is only one part of the calculation that determines the job priority (and thus the order in the queue). Other elements include the requested run time, the amount of memory, the type of nodes requested, and the amount of time a job has already been queued. So when multiple users are running many jobs, the person with shorter jobs will have more jobs running on average.

submitjob script

There are two version of **submitjob** script, submitjob on DC cluster is the same as submitjob2 on BC cluster, this is the latest version with newly added advanced features. The "submitjob" on BC cluster is old version. Job's stdout and stderr are saved in your home in pbs-output folder.

New version of submitjob,

```
usage: submitjob [-h] [-w WALLTIME] [-m MEM] [-c CPU] [-N NODE]
              [-e CONDA_ENVIRONMENT] [-f FILE] [-l] [-L LOG_PATH]
              [-E EMAIL] [-n NAME] [-a ARGS] [-b BATCH_SIZE] [-p]
              CMD OPTIONS INPUT
```

Old version of submitjob

```
Usage: submitjob [walltime] [-m -p -c] <command>
```

```
    walltime
       The expected run time of the job, measured in hours. Default: 24
    -m -mem --mem
       The maximum amount of memory used by the job in Gb.  Default: 2
    -p -property --property
       Currently disabled
       #Select nodes by properties, choices are OS6,OS7. Default: OS7
    -c -cpu --cpu
       The number of CPUs required on a single node. Default: 1
```

```
    command
       The command to run on the cluster. Note that any output redirection
       or pipe symbols must be escaped, i.e.   \> or \|
```

```
  Job STDERR is merged with STDOUT and redirected to /root/pbs-output/
  Any job exceeding the run time and memory limits will be killed automatically.
```

check job status scripts

There are several commands that can be used to check the status of running and queued jobs. **tracejob** only available on master node of each cluster, e.g, bc, dc01, rc01. **showq** only available master node of on BC and DC clusters.

- **jobstatus** best and advanced program to get summary of all running and queued jobs per user

usage: jobstatus [-h] {details,archive} ...

Check job status. If no subcommand is specified it prints out a summary of all jobs.

optional arguments:

```
-h, --help         show this help message and exit
```

Available subcommands:

```
For detailed subcommand help run: <subcommand> -h.
```

```
{details,archive}
  details          Show details of my jobs.
  archive          Archive finished jobs.
```

- **qstat** and **showq** give a listing of all running jobs

```
qstat                  => Full listing of all running jobs
showq                  => Full listing of all running jobs (different format)

qstat -u <username>    => Show only jobs for user <username>
qstat -n               => Show on which nodes the jobs are running
```

- **checkjob** and **tracejob** give a detailed overview of the status of each job, checkjob not available on RC

```
checkjob -v <jobid>    => Detailed status overview for running or queued jobs
tracejob <jobid>       => This command will also work for jobs that have already finished
tracejob -n 4 <jobid>  => By default 'tracejob' will only look back one day, specify more days with -n if needed
```

- **lastjoboutput** shows the output of the most recently finished job(s). Only available on BC and DC clusters. The script assumes that the job output is kept in the folder pbs-output in your home folder, i.e. this is the default output folder for the 'submitjob' script. The script has several options that can be viewed by executing 'lastjoboutput -h'.

```
lastjoboutput          => Shows the job output file for most recently finished job
lastjoboutput -n 5     => Shows the output for the last 5 finished jobs
lastjoboutput -i       => Select most recent jobs by jobID rather than time
```

check node status script

nodes

Usage: nodes [-h] [-o] [-s FILTER_STATES]

Check nodes status.

optional arguments:

```
-h, --help            show this help message and exit
-o, --show-job-owners
                      List jobs running on nodes
-s FILTER_STATES, --filter-states FILTER_STATES
                      Display only nodes in FILTER_STATES (comma separated).
```

Delet jobs

Use the **qdel** command to delete individual jobs or **deleteallmyjobs** to delete all your jobs at once. Note that you can only delete jobs that belong to you.

```
qdel <jobid>           => Delete job with ID <jobid>
deleteallmyjobs        => Will ask for confirmation before proceeding, only available on BC and DC clusters.
```

Interactive jobs

Sometimes it is necessary to run jobs interactively. Rather than running these jobs on the head node, it is possible to request an interactive session through the queueing system:

```
qsub -I              => Will start an interactive session through the queueing system
```

Set up email forwarding

Instead of reading email notifications about aborted jobs at the command line, you can have all mail forwarded to an external email account. To do this execute the following command, obviously replacing your.email@address with your actual email address:

```
echo your.email@address > ~/.forward
```

Temporary storage

If your jobs need to write to temporary files then it is preferable to use the $TMPDIR environment variable. $TMPDIR resolves to a temporary folder set aside by the queueing system for your job. It is created when a job starts and automatically deleted when a job ends. The main advantage of using $TMPDIR is that accessing local storage is much faster than writing to your home disk over the network and it also reduces cluster overhead. The second advantage is that the temporary files are automatically cleared at the end of a run, reducing the risk that a node runs out of disk space due to stray temporary job output. In perl I generally set the $TMPDIR variable as follows so that it will work outside the queueing system as well:

```
$ENV{TMPDIR} ||= '/tmp';      => Set environment variable TMPDIR unless it is already defined
```

PBS Exit Code

- Minus
  - -11 JOB_EXEC_RERUN: Job was rerun
  - -10 JOB_EXEC_FAILUID: Invalid UID/GID for job
  - -4 JOB_EXEC_INITABT : Job aborted on MOM initialization
  - -3 JOB_EXEC_RETRY: job execution failed, do retry
  - -2 JOB_EXEC_FAIL2 : Job exec failed, after files, no retry
  - -1 JOB_EXEC_FAIL1 : Job exec failed, before files, no retry

- 0-127 indicate exit code given by last command in job script. Examples:
  - 0 Job Success!
  - 1 General error

- 128-173 indicate process ended due to receiving signal. Examples:
  - 128 Invalid argument to exit()
  - 131 SIGQUIT: ctrl-\, core dumped
  - 132 SIGILL: Malformed, unknown, or priviledged instruction
  - 133 SIGTRAP: Debugger breakpoint
  - 134 SIGABRT: Process itself called abort
  - 135 SIGBUS: Bus error (on Guillimin: often a file system issue)
  - 136 SIGFPE: Bad arithmetic operation (e.g. division by zero)
  - 137 SIGKILL (e.g. kill -9 command)
  - 139 SIGSEGV: Segmentation Fault
  - 143 SIGTERM (probably not canceljob or oom)
  - 151 SIGURG: Urgent condition on socket

- 174-253 indicate a "Fatal error signal".

- Larger than 253
  - 254 Command invoked cannot execute
  - 255 Command not found, possible path problem
  - 265 SIGKILL (e.g. kill -9 command), possible out-of-memory error
  - 271 SIGTERM (e.g. canceljob or oom), possible memory error

---