# HPC Cluster Tips and Tricks

# General problems

- I have a question
  - Please first check all documentation to see if the answer is already in there (and especially this document, which is created out of the answers to common questions.).
  - Questions to HPC support can be asked via the Self-Service Portal, via the '*I have a different question*' form under the '*Research support*' tile. Please give a relevant '*Brief Description*', and in your question include the line '*This question is for operator group ICT-SYSTEMS-HPC.*' (Note: the linux bastions and the central storage are general IT facilities, not dedicated parts of the HPC cluster; direct your questions about those to their respective support teams.)
  - Do not send email directly to the personal email address of a support person, as that person may not always be able to respond immediately (i.e. outside office hours or when away or ill) and the other HPC support team members will not be able to see and respond to your question.

- SSH: "`The authenticity of host 'login`*X*`' can't be established.`"
  - When connecting to a login node for the first time, you must **not continue** when the key fingerprint reported by your ssh connection does **not match** one of the fingerprints shown here:
    - `login1.hpc.tudelft.nl` and `login2.hpc.tudelft.nl`
      - `2iPjH/j/Tf5JZU4OJyLpASA/GZ40eCqvcQnSSa++3nQ (ECDSA)`
      - `MURg8IQL8oG5o2KsUwx1nXXgCJmDwHbttCJ9ljC9bFM (ED25519)`
      - `mKgxUQvmOVM74XvFNhWt0ODsRvfnmwIgZWcw8uPJ68o (RSA)`
      - `05:24:a0:b4:83:27:05:32:4b:83:78:2a:20:99:f8:5c (ECDSA)`
      - `C5:21:46:cb:73:cd:72:e6:18:04:d6:67:2a:67:90:75 (ED25519)`
      - `05:17:84:7f:9f:18:e3:71:b4:df:5e:c0:12:db:e8:fc (RSA)`
    - `login3.hpc.tudelft.nl`
      - `IaBwyYiZi1Etj7yBDtdv7sByHzH+hedW69QA8UxGUqk (ECDSA)`
      - `O3AjQQjCfcrwJQ4Ix4dyGaUoYiIv/U+isMT5+sfeA5Q (ED25519)`
      - `fslv0RnC9zkVBf34i3g1BPKaYBcsTgKqu8+PMKLTEvw (RSA)`
      - `5e:9a:69:30:75:d3:b5:75:29:b3:32:fc:48:ab:b2:f9 (ECDSA)`
      - `31:eb:cd:95:8f:d1:78:29:e1:70:f9:8b:b0:cd:56:5c (ED25519)`
      - `ba:b9:92:4b:1a:00:8c:f1:aa:49:09:53:fa:b6:79:5f (RSA)`
  - When the key fingerprint matches, you can safely continue.
  - When in the future your ssh connection tells you that the key has changed, and it doesn't match one of the fingerprints above, contact the HPC support team.

- SSH: "`Permission denied, please try again.`"
  - The HPC cluster is not freely accessible. It is facilitated by several departments and groups within the university for their research/education.
  - If you have access, either your access expired (in case an end date was set), or your account was (temporarily) disabled due to problems with your use of the login nodes, or there is a problem with your NetID account/password.

- "`-bash: cd: /home/nfs/<NetID>: Key has expired`"
  - Your Kerberos ticket has expired. Your need to renew it, either by running '`kinit`', or by logging out then logging in again (using your password!).

    - Log out when you're not using the cluster (so you don't hit this problem, and so you don't block resources on the login node).

- "Disk quota exceeded"
    - The size of the data in this storage has reached the maximum allowed size (also known as quota limit). For home folders the maximum allowed amount of data is 8 GB, for project storage the quota limit can be up to 5TB of data.
    - To see how much space your files are using, run '`du -h ~ | sort -h`'. (When you have many files or folders, this can take a long time!)
    - To make space available, you'll either need to clean up some files (like installation archives and caches), or move some of your files somewhere else. Note: your home folder is for storing settings and installing small software packages, not for storing data or large software installations. You need to store those in project storage. Your project leader/supervisor can request project storage for you via the Self-Service portal (TOPdesk).

- "The system load on login*X* is too high! Please use another node if you can."
    - This message is mainly a warning for the person that is causing the high load. If that is you, you should either do the work as a cluster job, or limit the number of threads or memory that you use. If you're not the one, you can choose to ignore it.

- staff-umbrella: "`Operation not permitted`"
    - The network filesystem for the bulk, groups and project storage (staff-bulk, staff-groups, staff-umbrella) does not support `chmod` (changing permissions) or `chown` (changing owner or group) operations. When you run these operations, you will receive an "Operation not permitted" error. This has nothing to do with your personal rights, it's just not supported.
    - However, it's not necessary to change these, since the *default permissions are correct for normal use*. So you can **safely** skip these operations or **ignore** these errors.
    - For `rsync`, use '`rsync -a --no-perms`'.
    - When the error causes problems, a workaround is to (temporarily!) use the `/tmp` folder: move your folder that gives the error to `/tmp`, create a symbolic link from the folder in /tmp to the original location, rerun the commands that gave the error as before, then move your folder back from `/tmp` to the original location. For example, when you get an error in folder *<foldername>*, do:
      ```
      mkdir /tmp/${USER}
      mv <foldername> /tmp/${USER}/<foldername>
      ln -s /tmp/${USER}/<foldername> <foldername>
      <rerun command(s) that gave the error>
      rm <foldername>
      mv /tmp/${USER}/<foldername> <foldername>
      rmdir /tmp/${USER}
      ```

## Software

- My program requires a newer version of CMake
    - Use '`cmake3`'.

- How can I run a Docker container?
    - Using '`singularity`'. See the SURFsara Singularity documentation for information on using containers.

- My program requires a newer version of GCC
  - Newer versions of GCC are available through the devtoolset modules. See the slides for information on using modules.

- I want to use R
  - You can install R using Conda. Conda is available via the '`miniconda`' module.
  - You can use R from a container. Containers can be run using '`singularity`'.

- How to use TensorBoard on the HPC cluster?
  - TensorBoard is very insecure: anybody can connect to it, without authentication (i.e. when you run TensorBoard on the HPC cluster, any TU Delft user can connect to it). And this is actually on purpose, because making it secure and being able to guarantee that would require too much effort. **So you can't run TensorBoard directly on the HPC cluster!**
  - The most secure way to run TensorBoard is to run it on your personal computer (with a proper firewall). When you put your TensorFlow log files on a network folder, you can access them directly on your personal computer so you can use TensorBoard in the same way as you do in the HPC cluster. (You can also download the log files if you find that easier.)

## Job resources

- What are the limits?
  - The hard limits are in place only to protect the cluster from extreme overloads. The guiding principles of the cluster are fair-share and fair-use: all users should be able to use the cluster at the same time, and nobody should cause problems for the cluster or other users. So you need to make sure that your jobs do not unfairly hinder other jobs.

- What is the minimum runtime for a job?
  - Submitting, scheduling and starting jobs bring along a certain overhead. To reduce the effects of the overhead, jobs need to run for at least 1 minute, but preferably 5 minutes to 1 hour. So limit the number of CPUs to at most 2 to 4 (to make the jobs easier to schedule and not finish *too quickly*) and where possible combine multiple jobs into one job (to reduce the number of jobs to at most 50 and add together the runtime of short jobs).

- How to determine the CPU load, memory use and number of active threads of your program?
  - Log in on a login node (login1 or login3) and run your program ('python …').
  - Log in a second time on the same login node and run '`top -H -o TIME -u $USER`' to see all your threads and their %CPU and %MEM use. A %CPU of 100 corresponds to 1 CPU, 200 to 2 CPUs, and less than 75 is an indication that your program is not able to fully (optimally) use a CPU. %MEM is a percentage of 16 GB (login1) or 512 GB (login3); you'll need to convert the %MEM to MB or GB for your job script (round up to the next GB for a little extra headroom). To determine the active threads of the program, count the threads (belonging to the program) with '`%CPU`' > 5 and steadily increasing '`TIME+`'. (Quit top by pressing 'q'.)
  - For running jobs, the 'CPU Load' statistics on the website is one indicator of problems (when the CPU load exceeds 100%).
  - Background: the system restricts a job so that it can only use the allocated CPU cores, and a CPU core can only run one thread at a time. So the actual CPU use cannot exceed the allocated number of CPU cores. When your program uses more active threads than the

number of allocated CPU cores, only some threads can run, and the rest of the threads have to wait. So the CPU load, i.e. the number of active (running and waiting) threads per core, goes above 100%. Having threads waiting reduces the performance (and adds overhead).

**Example**
Let's assume you want to know the sum of 50 numbers. The simplest way to calculate this is to add the first two numbers, then add the third number to that, and so on. You will need to perform 49 additions, one at a time (one after the other), in one sequence, thus **1 thread**. To run this single thread, **a single CPU** is needed (to perform one addition at a time). The "time" needed to compute the sum is **49 CPU cycles** (1 addition per cycle). The **CPU load is 100%** (1 active thread per CPU) and the **CPU efficiency is 100%** (the one CPU will be active the whole time).

A simple way to speed this up would be to divide the 50 numbers in two groups of 25 numbers, and use **2 separate threads** to calculate the sums of the two groups at the same time (in parallel). Then add the sums of the two groups to get the total sum. Each thread needs its own CPU to perform the addition, so you need **2 CPUs** to be able to run the **2 threads** in parallel. Afterwards you will need to add the two sums together to get the total sum. So the time needed when using 2 threads is **25 cycles** (24 additions to sum each group, and 1 addition for the total). The average CPU load will be **98%** (1 active thread per CPU for 24 cycles, plus one final addition on only 1 CPU; the other CPU will be idle). The average CPU efficiency is also **98%** (49 additions in 50 available CPU cycles).

The **2 threads** could also have shared **1 CPU** (by first doing an addition in the first thread, temporarily storing that result, then doing an addition in the second thread, temporarily storing that result, and so on). However, because of the additional overhead, it would take *longer* than by using *one* thread. The average CPU load would be ***196%*** (24 additions with 2 active threads on 1 CPU, plus the final addition). The CPU efficiency would be **100%** (the one CPU will be active the whole time), but for a longer time. *In general, more than 1 active thread per CPU is less efficient.*

You might have jumped to the conclusion that more threads would be even better, right? What happens when you divide the numbers in 8 groups (6 groups of 6 numbers and 2 groups of 7 numbers), and use **8 threads** (on **8 CPUs**) in parallel to sum those groups? The 6 threads that sum 6 numbers will be finished in **5** cycles, but the 2 threads that sum 7 numbers need **6** cycles. So those 6 CPUs will have to wait **idle** for one CPU cycle for the other **2 threads** to finish. Then you still have to add the sums of the 8 groups to get the total sum. If you use **1 thread** for this, you will need 7 CPU cycles. During this time, the other 7 CPUs will all be waiting **idle**!
The time needed would be **13 cycles** (*excluding any additional overhead*): 6 cycles to sum the groups and 7 cycles to calculate the total. The first 5 cycles 8 active threads run (on the 8 CPUs), the next cycle only 2 active threads remain and the final 7 cycles only 1 active thread runs. So the average CPU load is only **47%** (= (5 * (8 / 8) + (2 / 8) + 7 * (1 / 8)) / 13). In total 104 CPU cycles (13 cycles * 8 CPUs) were available to perform the 49 additions, so the CPU efficiency is only **47%** (= 49 / 104). *In general, the more threads you use, the lower the average CPU efficiency!*

Conclusion: in this example, you can run **4** jobs with **2** threads each ***in less time*** than it takes to run **2** jobs with **8** threads each (especially considering that larger jobs often have longer waiting times). *So choose your number of threads optimally.*

- How to determine the GPU use of your program?
  - Log in on a login node with a GPU (`login1` or `login3`).
  - Run '`nvidia-smi -l`'. Make sure no processes are using the GPU.
  - Log in a second time *on the same login node* and run your program ('`python …`').
  - The current GPU utilisation of your program is reported by `nvidia-smi` under '`GPU-Util`'.

- Quit your program and nvidia-smi by pressing '`Ctrl+c`', and log out from the login node.

- How do I request CPUs for a multithreaded program?
  - If you can specify the number of threads your program will use:
    - determine a "smart" number of threads ( for example the number of currently available CPUs, or the number of threads needed to finish within 4 hours), **always a multiple of 2**, but **preferably no more than 8**,
    - request a cpu for each thread (`--cpus-per-task=<#threads>`), and
    - tell your program to use `"$SLURM_CPUS_PER_TASK"` threads.

      ```
      #/bin/sh
      #SBATCH --ntasks=1 --cpus-per-task=2
      srun ggsearch36 -t "$SLURM_CPUS_PER_TASK"
      ```
  - If your program uses a fixed number (2, 4,..) of threads:
    - request a cpu for each thread (`#SBATCH --cpus-per-task=<#threads>`).

      ```
      #/bin/sh
      #SBATCH --ntasks=1 --cpus-per-task=2
      srun my_program
      ```
  - If your program automatically uses as many threads as there are CPUs on a node (for example **java** programs):
    - do **not** use functions that detect the total number of CPUs of a computer (for example, `os.cpu_count()` for Python), use functions that detect the **CPU affinity** (for example, `len(os.sched_getaffinity(0))` for Python).
    - for some code you can explicitly specify the correct number of threads using an environment variable (export <VARIABLE>=<value>):

      ```
      #/bin/sh
      #SBATCH --ntasks=1 --cpus-per-task=2
      export NUMBA_NUM_THREADS="$SLURM_CPUS_PER_TASK"
      srun python script.py
      ```
    - If that is not possible, request a complete node for one task (`#SBATCH --ntasks=1 --exclusive`), and tell srun to use `"$SLURM_CPUS_ON_NODE"` threads.

      ```
      #/bin/sh
      #SBATCH --ntasks=1 --exclusive
      srun --cpus-per-task="$SLURM_CPUS_ON_NODE" java_program
      ```

- How can I use a GPU?
  - Use this example GPU job script.
  - Request a GPU:
    - `#SBATCH --gres=gpu` (first available GPU of any type)
    - `#SBATCH --gres=gpu:pascal:1` (one GPU, only Pascal type)
    - `#SBATCH --gres=gpu:2` (two GPUs for the same job)
  - And load a CUDA module (and also a cuDNN module when needed).

- How can I let multiple programs use the same GPU?
  - If you want to let multiple instances of your program share a GPU, you'll have to start them in parallel from the same job using `srun`.

    ```
    #!/bin/sh
    #SBATCH --gres=gpu:1
    #SBATCH --ntasks=2
    srun program
    ```
  - If you want to use different programs, use the `--multi-prog` option:

job.conf:

```
0 <program 1>
1 <program 2>
```

job.sh:

```
#!/bin/sh
#SBATCH --gres=gpu
#SBATCH --ntasks=2
srun --multi-prog job.conf
```

- How much memory can I use?
  - A task can run on a single node only. Since most jobs consist of a single task, those jobs are therefore limited to the total amount of memory in a node. However, since that would leave no memory for other jobs on the node, do not request more memory than your jobs need! (Also see How can I see the CPU or RAM usage of a job?)
  - There is also a per-user and per-QoS memory limit for the combined requested memory of all running jobs (of a user) in a certain QoS (see the slides). So, to run the most jobs at the same time, don't request more memory than your jobs need.
  - The average amount of memory that you can request when you want to run a lot of jobs is **less than 8Gb** per job (**less than 4Gb** per CPU). This will give you the most running jobs on the cluster.

- How can I see the CPU or RAM usage of a job?
  - `sstat` shows specific usage information for (running) job *steps* (i.e. when you start your program using `srun program`), using something like this (all on one line):

    ```
    sstat --allsteps
    --format=JobID,NTasks,MinCPU,MaxRSS,MaxDiskRead,MaxDiskWrite
    <jobid>
    ```

    The MaxRSS field, for example, shows the maximum amount of memory used until now.
  - `seff <jobid>` shows basic CPU and memory usage statistics of the whole job, including easy to understand percentages, but only for *finished* jobs.

- How can I see the GPU usage of a job?
  - Use this example GPU job script to obtain the GPU usage of your job.

- How can I limit the number of jobs per node?
  - When a job risks overloading a node (for example because it creates a large load on the network storage, or because it uses a lot of space in `/tmp`), you need to explicitly limit the number of jobs that can be run simultaneously on a node.
  - One way to do this is to use the GRES (Generic consumable RESource) "jobspernode". This allows you to limit the number of jobs per node to one, two, three or four. You do this by requesting **one** of the following GRES in your jobs (pick the one you need):

    ```
    #SBATCH --gres=jobspernode:one:1
    #SBATCH --gres=jobspernode:two:1
    #SBATCH --gres=jobspernode:three:1
    #SBATCH --gres=jobspernode:four:1
    ```

    When you specify one of these in your job, for example the one for four jobs per node, the scheduler will start (up to) four jobs on one node, and will then wait until a job finishes before starting another job on that node.

- Should I feel guilty about running a lot of jobs with GPU/CPU usage?
    - Actually, the more jobs running and waiting in the queue, the more efficient and fair the scheduler can plan and divide the resources over the waiting jobs, so the higher the throughput and the sooner all work is finished. The scheduler will make sure that the available resources are divided fairly between the jobs of all users. This is done based on previous usage: the amount of allocated CPUs, GPUs and memory multiplied by a job's real runtime. The higher a user's previous usage, the lower the priority of that user's jobs waiting in the queue. So when one user has a high previous usage, the waiting jobs of other users will be started before the jobs of that user. (The previous usage continuously decays, so when the usage stops, that user's priority automatically goes back up again in a few days.)
- How do I clean up `/tmp` (when a job fails)
    - When your job stores temporary data locally on a node, your job needs to clean up this data before it exits. So include an `rm` command at the end of your job script. (The system does not clean up this data automatically.)
    - When the job fails (or is canceled or hits a timeout), the clean up requires some special code in the job script:

```
#!/bin/sh
#SBATCH ...

# Create local temporary folder
tmpdir="/tmp/${USER}/${SLURM_JOBID}"
mkdir --parents "$tmpdir"

# You may want to uncomment this to know what to clean up when the
clean up fails:
#echo "Temporary folder: $(hostname --short):${tmpdir}"

# Cleanup temporary folder
function clean_up {
  rm --recursive --force "$tmpdir" && echo "Clean up of $tmpdir
completed successfully."
  exit
}

# Setup clean_up to run on exit
trap 'clean_up' EXIT

# Make sure your program uses this temporary folder
# (Tell your program to use the "$tmpdir" location!)
srun …
```

    - If all else fails, login in interactively to the node and manually clean up the files:
      `sinteractive --nodelist=<node>`
    - Request a node with enough space in `/tmp` (at least twice what your script needs, because other jobs need to use `/tmp` too):
      `#SBATCH --tmp=<size>G`

# Scheduler problems

- Interactive sessions hang when left for some time without input
  - This seems to be a bug. For now, use `sattach` or one of the login nodes.
  - Of course, if a session is really idle (i.e. nothing running), just close it so another job can run.

- Job pending with reason "`QOSGrpCpuLimit`"
  - Each QoS (Quality of Service) sets limits on the total number of CPUs in use for that QoS. If the total number of CPUs in use by running jobs (of the whole group of users combined) in a QoS hits the set limit, no new jobs in that QoS can be started, and jobs will stay pending with reason "QOSGrpCpuLimit". This is not an error; when running jobs finish, the highest priority pending job will be started automatically.

- Job pending with reason "ReqNodeNotAvail"
  - Sometimes nodes are not available to start new jobs (for example when they are reserved for maintenance). When jobs can only run on those nodes, they will remain pending with the reason "ReqNodeNotAvail" until the node become available again.
  - The requested runtime of a job is an important factor here. When a reservation starts in 1 day, a job with a requested runtime of 7 days will not be able to start (since it would not be finished before the start of the reservation), but a job with a requested runtime of 4 hours can still be run normally. So when possible reduce the requested runtime.
  - The requested resources (number of CPUs, amount of memory, number or type of GPUs or specific constraints) limit the number of nodes that are suitable to run a job. So when possible reduce the requested resources and constraints, and do not request specific GPU types.

- Why does my job run on some nodes but fail on other nodes?
  - The nodes have different configurations (processor types, number of cores, memory size, GPU support, and so on). If you need a specific feature, make sure to request it in your sbatch script. Examples:
    ```
    #SBATCH --constraint=avx
    #SBATCH --constraint=avx2
    #SBATCH --gres=gpu
    ```
  - If your program uses specific processor instruction extensions (AVX/SSE/AES/…), it will crash (with an 'Illegal instruction' error) on processors that do not support those extensions. Either compile your program to use only standard instructions (be generic), or request nodes that have support. The login node login3 has the least advanced CPUs so if you compile your programs there they should run on all other nodes.

- Why does my job fail and is there no slurm-XXXXX.out output (or error)?
  - When your job doesn't have a valid Kerberos ticket it can't read or write files (such as the `slurm-XXXXX.out` output).
    It's best to do a fresh login to a login node when you want to submit a new job. This way you're sure your job's Kerberos ticket is valid and will remain valid for the next 7 days. If needed, you can update the Kerberos ticket for a running job by executing '`auks -a`' (also from a fresh login).

- "sbatch: error: Batch job submission failed: Access/permission denied"
  - You can only submit jobs (using `sbatch`) from the login nodes (`login1, login2, login3`). When you try to submit a job from within another job (including an interactive job) you will receive this error.
  - To submit multiple jobs from a script, create a file with the submit commands:
    ```
    sbatch job1.sh
    sbatch job2.sh
    sbatch job3.sh
    …
    ```
    Then login to one of the login nodes and source the file:
    ```
    source script
    ```
- "sbatch: error: Batch job submission failed: Invalid account or account/partition combination specified"
  - Either you're trying to use a (special) partition that you don't have access to.
  - Or your account has been (temporarily) disabled because of problems with your jobs. The usual problems are (a combination of) of these:
    i. Your jobs are not using all of the requested resources (CPUs, GPUs, memory, runtime).
    ii. Your jobs try to use more resources than requested (more active threads than the requested number of CPUs, out of memory failures, timeout failures).
    iii. Too many jobs are failing or being cancelled.
    iv. Failing to follow the cluster workflow as described on page 5 of the slides.

    You need to figure out the problem(s), fix them, and then send an email to the cluster administrators to explain the problem(s) and the way you fixed them.
- "srun: error: Unable to allocate resources: Invalid qos specification"
  - You can't directly execute a jobscript, you need to submit the jobscript using `sbatch`:
    ```
    sbatch jobscript.sh
    ```
- "slurmstepd: error: Exceeded step memory limit at some point."
  - Your program wants to use more memory than you requested. You'll either need to limit the memory use of your program or request more memory. (Also see How much memory can I use? and How can I see the CPU or RAM usage of a job?)
- "Auks API request failed : auks cred : credential lifetime is too short" / "Auks API request failed : krb5 cred : unable to renew credential"
  - Your authentication (Kerberos ticket) expired. You get a Kerberos ticket (using your password) when you log in to the bastion server or login node. Jobs that run on a compute node also require authentication. Therefore, when you submit a job, your Kerberos ticket is stored in the Auks system so that your job can use it. However, the maximum lifetime of a Kerberos ticket is 7 days. So 7 days after you last logged in and submitted a job, the Kerberos ticket in the system expires and the job fails.
  - Therefore, for infinite jobs or long jobs that have had to wait in the queue for a couple of days, the Kerberos ticket needs to be renewed before it expires. The simple way to do this is to log in to a cluster login node and run '`auks -a`'.
  - When you frequently need to do this, you can (on a cluster login node) run '`install_keytab`' to install an encrypted version of your password (called Kerberos keytab) for automatic Kerberos ticket renewal. *Important:* when you change your NetID password, your Kerberos keytab becomes invalid, so you will need to rerun this command.

- What can be done about some jobs using all CPUs or memory (making it impossible for me to use other unused resources like GPUs)?
    - Resources can be used by one job at a time only, so when some resources are completely used, other jobs will have to wait until their required resources become available. The waiting is unavoidable.
    - See the answer to Should I feel guilty about running a lot of jobs with GPU/CPU usage? (The scheduler is already dividing the available resources fairly over all jobs based on the policies, using priorities based on previous usage. As soon as a running job finishes and it's resources become available again, the highest priority waiting job is automatically started.)
    - The policies for the scheduler are set by the cluster users (in the cluster board meeting). To change the policies, all users must agree that the changes are necessary and fair to all users. (So the cluster administrators aren't able to change the policies on request!)
    - You can always contact a user (nicely!😉) about the possibility to (for example) exclude a certain node. That user can decide for him-/herself if he/she wants to cooperate or not (for example in case of deadlines).
      It's usually possible to determine a person's initial (or first name) and last name from his/her username; if not, run: `finger <username>`
    - If you experience a real problem because of this (not being able to efficiently develop code because of the waiting, or not going to make an upcoming deadline), you should contact the cluster administrators to see if they can provide support for that specific problem.
    - It's not desirable to reserve resources for certain types of jobs only. Since other types of jobs wouldn't be able to use those resources even when they would be idle, this would reduce the overall cluster throughput (and recreate the exact same problem that you experience for those other jobs).
    - No type of research can make special claims regarding resources. All research that uses the cluster is equally dependent on the cluster resources: 50+ CPU jobs or jobs requiring 50GB memory can't be run on a laptop any more than a GPU job requiring 5GB of GPU memory. When one type of resource is completely used, that is because it is required to do the same kind of research that you need the cluster for.