HELIOS cluster documentation

Pavel Strachota

April 25, 2024

Download PDF version for offline use

Contents

1 Introduction

This document is a brief documentation for new users of the HELIOS cluster at the Department of Mathematics, Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague. It is intended to provide the very basic information required to get started. Manuals for using the more advanced features of the software environment can be found in the references.

The impatient users may skip directly to **Section 3**.

2 Design and Specifications

HELIOS is a high performance computing (HPC) cluster, i.e. a system of interconnected compute servers (a.k.a. *compute nodes*, *execution hosts*) where computing tasks are scheduled and executed. Apart from the compute nodes, the system features a high capacity, high throughput storage subsystem and a server for interactive access - the *login node*. Users work on the login node to prepare their computing tasks for execution (compile the source code, set parameters etc.) and then use the job scheduler to submit them for execution. It is **forbidden** to execute demanding interactive computations directly on the login node!

2.1 Hardware Specs

- **login node**: 2x16-core Intel XEON Gold 6130@2.1GHz CPU (hyper-threading enabled), 384 GB RAM
- **additional login node**: 2x16-core Intel XEON Gold 6130@2.1GHz CPU (hyper-threading enabled), 384 GB RAM - currently configured as type-A compute node
- **24x type-A** node: 2x16-core AMD EPYC 7281@2.1GHz CPU (SMT mode disabled), 128 GB RAM, 360 GB local SSD storage

- **2x type-B** node: 2x8-core Intel XEON Gold 6134@3.2GHz CPU (hyper-threading disabled), 384 GB RAM, 360 GB local SSD storage
- **2x type-C** node: 4x NVIDIA Tesla V100 SXM2 w. 16GB HBM2 RAM, 2x16-core Intel XEON Gold 6130@2.1GHz CPU (hyper threading disabled), 384 GB RAM, 360 GB local SSD storage
- **1x type-C4** node: 4x NVIDIA A100 SXM4 w. 80GB HBM2 RAM, 1x32-core AMD EPYC 7543@2.8GHz CPU (SMT mode disabled), 1024 GB RAM, 550 GB local SSD storage
- **1x type-C1** node: 1x NVIDIA A100 PCIe w. 80GB HBM2 RAM, 1x16-core AMD EPYC 7313@3.0GHz CPU (SMT mode disabled), 512 GB RAM, 700 GB local SSD storage
- 200 TB raw storage space: QSAN XS3216 dual controller SAN in active-active mode + QSAN XD5216 dual controller JBOD expansion enclosure
  - RAID6, Lustre High Availability storage cluster using two front-end servers
- Intel Omni-Path 100Gbit/s interconnect

2.2 Software Environment

- **CentOS** 7.8 Linux
- **PBS Pro** 18.1 job scheduler
- **environment modules** for individual software package loading and multi-versioning

3 Logging in to HELIOS

There are two ways of logging in to the login node:

1. Using **SSH command line** interface:

   ```
   ssh USERNAME@helios.fjfi.cvut.cz
   ```

   Note that Windows users may use **PuTTY**. See Section 3.1 below for what to substitute for `USERNAME`.

2. **Remote desktop** (over SSH) using the **X2Go** client. Please select ”MATE” as the desktop environment in the session configuration dialog.

File transfer to and from the login node can be done using the standard tools: **sftp, scp, sshfs, WinSCP**.
SSH server runs on the standard port 22.

- **Password authentication** is allowed from within the CTU network (IP range 147.32.0.0/16) only.
- For access from outside the CTU network, **public-key authentication** has to be used instead. For help on creating and using SSH keys, consult the documentation for `ssh-keygen` under Linux and **PuTTYGen** under Windows. Alternatively, one can use the **VPN** available at CTU.

3.1 User Accounts

User accounts are created upon request (see Section 8). **User names and passwords are the same** as for most other IT resources at the university (USERMAP, SSO, etc.). Accounts for external users can also be created.

The primary group for all users is ”users”. In addition, users are members of one or more supplementary groups (students, employees, project team members etc.). Access to job submission queues (see Section 5.1) is controlled by group membership.

3.2 Command Shell

The default command shell for all users is `/bin/bash` and this setting cannot be changed by `chsh` (it will be overwritten upon next user database update). The other installed shells are `tcsh` and `zsh`.

4 Installed Software

4.1 Environment Modules

Most useful software components except for system utilities are managed by the **`module`** command which basically sets up the environment (the contents of the `PATH`, `MANPATH`, `LD_LIBRARY_PATH` etc. variables) **for the current shell session** in an appropriate way. The software modules (e.g. the compiler and the MPI library module) must be loaded **during compile time** on the login node as well as **during execution time** on the compute nodes. See also Section 5.4.

- To list the available software modules (the layout is changed here for better readability):

```
[stracpav@login1 ~]$ module avail

---------------- /usr/share/Modules/modulefiles ----------------
null           module-info      modules         dot

gcc/6.5        gcc/8.2          gcc/8.5         gcc/11.3
intel/icc13    intel/icc19      intel/oneapi-2023

cmake/3.20.1   cmake/3.24.2     cmake/3.27.7

cuda/10.0      cuda/10.1        cuda/10.2       cuda/11.7

ANSYS/19.1     ANSYS/21R1       openfoam/6      openfoam/v2012

MATLAB/R2012a MATLAB/R2017a     MATLAB/R2019b   MATLAB/R2020a
MATLAB/R2023b

gnuplot/5.2.8
visit/2.12     Qt/5.13.0        vtk/8.2.0
```

```
paraview/5.5.1
paraview/5.7.0  paraview/5.7.0-headless
paraview/5.8.1  paraview/5.8.1-headless

python/3.10.9
julia/1.4.2   julia/1.5.2      julia/1.6.0    julia/1.10.2

R/3.5.2       RStudio/1.2.1206

Mathematica/11.3.0    Mathematica/12.0.0

openmpi/2.1.5-gcc_4.8.5
openmpi/2.1.5-gcc_4.8.5-psm2
openmpi/2.1.5-gcc_4.8.5-psm2-cuda10.1
openmpi/3.1.3-gcc_4.8.5
openmpi/3.1.3-gcc_4.8.5-psm2
openmpi/4.1.0-gcc_4.8.5-psm2-cuda10.1
openmpi/4.1.0-gcc_4.8.5-psm2-cuda11.7
```

- To load a particular module:

  ```
  [stracpav@login1 ~]$ module load openmpi/2.1.5-gcc_4.8.5-psm2
  ```

  Note that the TAB key can be used to auto-complete the module names.

- To list the currently loaded modules

  ```
  [stracpav@login1 ~]$ module list
  Currently Loaded Modulefiles:
    1) openmpi/2.1.5-gcc_4.8.5-psm2
  ```

- To unload a particular module:

  ```
  [stracpav@login1 ~]$ module unload openmpi
  ```

  Note that the module name needn't be complete.

- To unload all loaded modules

  ```
  [stracpav@login1 ~]$ module purge
  ```

4.1.1 Module-Specific Notes

- The **psm2** versions of **OpenMPI** use the Omni-Path fabric for communication (ultra-fast and preferred). The other versions use 1GbE only and are provided mostly for debugging purposes.
- **MATLAB** installations are not complete, some more exotic features are missing. **Simulink** is also not installed. If you find you need them, please contact support.
- **MATLAB Parallel Server** license is currently not available (the general CTU license is used). Therefore, cluster profiles cannot be used and parallel jobs / parallel pools can only run in "local" mode within a single compute node (see Section 5.4.5). Please do not start MATLAB parallel pools on the login node!
  - Apart from not having the necessary license available, MATLAB versions 9.7 (R2019b) and onward are fully compatible with the current configuration of the PBS Pro scheduler.
- **ANSYS** installations are not complete and are mostly oriented toward **Fluent** and **CFX** CFD.
  - They are limited to 16 CPUs by the CTU academic license.
  - The number of HPC licenses is also limited and submitting a larger number of ANSYS jobs may result in "`Not enough ___ - HPC licenses`" errors where "`___`" stands for the used ANSYS product, e.g. Fluent.
  - Only single-node MPI computations (Section 5.4.7) and single-node interactive sessions (Section 5.5) are supported.
- **R** contains the basic installation only. Custom packages can be installed to the users' home directories, as needed.
- **ParaView** needs to be run with the MESA OpenGL rendering backend as 3D acceleration is not available over X11 forwarding or in X2Go
  - for `paraview/5.5.1`, use

    ```
    [stracpav@login1 ~]$ paraview --mesa
    [stracpav@login1 ~]$ pvserver --mesa
    ```

  - for `paraview/5.7.0` and `paraview/5.8.1`, use

    ```
    [stracpav@login1 ~]$ paraview-mesa paraview
    [stracpav@login1 ~]$ paraview-mesa pvserver
    ```

  - for `paraview/5.7.0-headless` and `paraview/5.8.1-headless`, use

    ```
    [stracpav@login1 ~]$ pvserver
    ```

4.2 System Packages

There is a number of internal CentOS (RPM) packages installed on the login node and on the compute nodes. All compute nodes have the same set of packages installed. If you happen to need a piece of software in the form of an additional RPM package, please contact support.

5 Running Compute Jobs

The compute jobs are scheduled to be run on the compute noes by submitting them to one of the available job queues. Whether or not a user is allowed to submit a job to a queue depends on their membership in the supplementary groups.

5.1 Job Queues

The queues are currently configured as follows:

- To list all queues:

```
[stracpav@login1 ~]$ qstat -q

server: login1

Queue            Memory CPU Time Walltime Node   Run   Que   Lm  State
---------------- ------ -------- -------- ---- ----- ----- ----  -----
cpu_a              --       --    72:00:00  --     25     0   --   E R
cpu_b              --       --    24:00:00  --      0     0   --   E R
gpu                --       --    24:00:00  --      0     0   --   E R
gpuA               --       --    24:00:00  --      0     0   --   E R
express            --       --    00:10:00  --      0     0   --   E R
express2           --       --    00:10:00  --      0     0   --   E R
expressX           --       --    00:10:00  --      0    50   --   E R
cpu_a2             --       --       --     --     18    10   --   E R
gpu2               --       --       --     --      1     0   --   E R
cpu_aX             --       --    72:00:00  --      0     0   --   E R
gpuX               --       --    24:00:00  --      0     0   --   E R
student            --       --    01:00:00  --      0     0   --   E R
gpu_student        --       --    00:30:00  --      0     0   --   E R
cpu_aL             --       --    240:00:0  --      0     0   --   E R
gpuL               --       --    96:00:00  --      0     0   --   E R
                                                ----- -----
                                                   44    60
```

  Notes:

  - Note the **maximum wall time** (real execution time) of the jobs in each queue. Apart from that, queues also specify a much shorter default wall time for each job if one is not required. The default wall time settings is not written here (it is possible to find out using the qmgr command). Instead of relying on the default wall time, **always specify a wall time requirement upon job submission** (see Section 5.4 below).

5.2 Queue Properties

- **student** and **cpu_a** accept jobs to be executed on type A nodes (see Section 2.1).
- **cpu_b** accepts jobs to be executed on type B nodes.
- **gpu** and **gpu_student** accept jobs to be executed on type C nodes (NVIDIA Tesla V100 GPUs)
- **gpuA** accepts jobs to be executed on type C4 and C1 node (NVIDIA A100 GPUs)
- **express** queue accepts very short jobs that preempt (temporarily suspend) the other running jobs.
- All users have access to the **student** and **gpu_student** queues
  - These queues are restricted to a maximum of 16 CPU cores and 1 GPU accelerator, respectively.
  - These queues are served by a subset of type A nodes only.
- Upon supervisor's request, students working on their bachelor's and master's degree projects can gain access to the **cpu_a**, **cpu_b**, and **gpu** queues.
- Employees, PhD students and project team members are eligible to submit jobs to **cpu_a**, **cpu_b**, **gpu**, **gpuA**, and **express**.
- The **cpu_aL** and **gpuL** queues are identical to **cpu_a** and **gpu**, respectively, except for the substantially longer walltime limit (see the queue listing above). Access to these queues is temporarily granted to individual users upon request. If you need to submit one-time computations exceeding the limits of the standard queues, please contact support and describe your requirements.
- The other queues are reserved for BioCCS/U research team members and submission to them will fail for all other users.

5.3 Requesting Resources

Upon submitting jobs to queues (see Section 5.4 below for examples how to do that), the user specifies the resources required for the job. Besides the execution time (wall time) mentioned above, **it is necessary** to specify the requirement for

- the **number of CPU cores** (given per compute node or *chunk* - see man `pbs_resources` for details),
- **the amount of memory (RAM)** per compute node**,**
- **and the number of GPU accelerators** (if applicable).

Once the job is scheduled for execution, the required resources are fully granted for the job. On the other hand, the resource limits are also strictly enforced (by means of the *cgroups* feature of the Linux kernel).

The above resources have very restrictive default values: **1 CPU core, 256 MB of memory and no GPU accelerator**. This means that:

- Failing to ask for an appropriate number of CPU cores and trying to run a multi-threaded program will bind all threads to a single CPU core and the program will perform poorly.
- Failing to ask for an appropriate amount of memory and running virtually any serious computation will result in an out-of-memory error and the program being killed.
- Failing to ask for a GPU accelerator and trying to run a CUDA application will fail.

5.3.1 Advice for Memory Requirements

The maximum amount of memory available for a job is approx. **124 GB** on type A nodes (`cpu_a` queue) and approx. **375 GB** on type B/C nodes (`cpu_b`, `gpu` queues). A tighter estimate of the maximum can be found by investigating the available resources using e.g. "`pbsnodes node01`" and subtracting a couple of megabytes from the obtained value.

- If your job uses all CPU cores on one or more compute nodes, it can simply ask for all available memory as well.
- However, if you only require a single CPU core (or a small number of them) for your job, please try to estimate the real memory requirements so as not to prevent other jobs from running on the same compute node.

5.4 Running Batch (Non-interactive) Jobs

Running a batch (non-interactive) job consists in creating a job submission script and using the **`qsub`** command to enqueue it. Job submission scripts are BASH/Python scripts with special ”comment-like” directives for the PBS Pro job scheduler. The directives can also be replaced by the respective command-line arguments to `qsub`. Below, one can find examples of submission scripts for different purposes.

5.4.1 MPI Jobs

The job submission script for an **MPI job** may look like this:

```
#!/bin/bash
### Job Name
#PBS -N intertrack_job
### required runtime
#PBS -l walltime=01:00:00
### queue for submission
#PBS -q cpu_a

### Merge output and error files
#PBS -j oe

### request 4 chunks with 32 CPUs each
### running 32 MPI processes per chunk (total 128 MPI ranks)
### (i.e. 1 chunk = 1 complete node with 124 GB of memory available for the job),
#PBS -l select=4:mem=124G:ncpus=32:mpiprocs=32

### start job in the directory it was submitted from
cd $PBS_O_WORKDIR

### load the necessary software modules
module load openmpi/2.1.5-gcc_4.8.5-psm2
```

```
### run the application and provide its command line arguments
mpirun ./intertrack Params

### Note that more applications/shell commands may be added here
### (e.g. for post-processing of the results)
```

Suppose we save the above script under the name ”RunQ” in the application’s executable directory. Then we submit the job by

```
[stracpav@login1 ~/WORK/Progs-backport/apps/intertrack]$ qsub RunQ
1576.login1
```

The job’s ID is printed.

5.4.2 Hybrid OpenMP / MPI Jobs

The job submission script for a **hybrid OpenMP** / **MPI** job may look like this:

```
#!/bin/bash
### Job Name
#PBS -N intertrack-hybrid-job
### required runtime
#PBS -l walltime=01:00:00
### queue for submission
#PBS -q cpu_b

### Merge output and error files
#PBS -j oe

### Request 2 chunks with 16 CPUs each
### and spawn 1 MPI process with 16 threads on each node
### (1 chunk = 1 complete node in the cpu_b queue with ~375 GB of memory available)
#PBS -l select=2:mem=375G:ncpus=16:mpiprocs=1:ompthreads=16

### start job in the directory it was submitted from
cd $PBS_O_WORKDIR

### load the necessary software modules
module load openmpi/2.1.5-gcc_4.8.5-psm2

### run the application (don't forget to disable OpenMPI's default binding policy
### when multithreading is used)
mpirun --bind-to none ./intertrack Params
```

5.4.3 CUDA Jobs

The job submission script for a **CUDA** job using 10 GB of memory, 4 CPU cores, and a single NVIDIA Tesla V100 accelerator may look like this:

```
#!/bin/bash
### Job Name
#PBS -N LBM_CUDA
### required runtime
#PBS -l walltime=01:00:00
### queue for submission
#PBS -q gpu

### Merge output and error files
#PBS -j oe

#PBS -l select=1:mem=10G:ncpus=4:ngpus=1

### start job in the directory it was submitted from
cd $PBS_O_WORKDIR

### load the necessary software modules
module load cuda/10.0

### run the application
./lbm_cuda_simulation
```

Note that the `CUDA_VISIBLE_DEVICES` environment variable is not set at all, which is completely fine for device isolation by means of *cgroups* and CUDA versions > 7.0 .

5.4.4 Mixed CUDA + MPI (+OpenMP) Jobs

Combining CUDA and MPI allows executing parallel code on multiple GPU accelerators on a single node and even across multiple compute nodes. Currently, multi-node GPU computation is only possible for the BioCCS/U research team members using the **gpuX** queue. OpenMPI supports the *GPUDirect* technology which enables direct transfers of data between GPUs over NVLink or the OmniPath fabric without intermediate use of the main system memory. Distributed GPU computation is a rather delicate activity requiring interoperation of CUDA-aware OmniPath kernel drivers, the CUDA-aware PSM2 library, OpenMPI build with CUDA enabled, and the CUDA framework itself. From the programmer's perspective, mixing multi-GPU CUDA code with MPI has some specific caveats, as described e.g. here.

On HELIOS, the supported combinations of modules are as follows:

- `cuda/10.1` + `openmpi/2.1.5-gcc_4.8.5-psm2-cuda10.1` in the **gpu**, **gpuL**, **gpu2**, **gpuX**, **gpuA** queues
- `cuda/10.1` + `openmpi/4.1.0-gcc_4.8.5-psm2-cuda10.1` in the **gpu**, **gpuL**, **gpu2**, **gpuX**, **gpuA** queues
- `cuda/11.7` + `openmpi/4.1.0-gcc_4.8.5-psm2-cuda11.7` in the **gpuA** queue

These MPI modules with CUDA support must NOT be used for regular MPI computations on the CPU-only nodes.

The job submission script for a **CUDA+MPI+OpenMP job** may look like this:

```
#!/bin/bash
### Job Name
#PBS -N lbm3d-CUDA-MPI
### required runtime
#PBS -l walltime=01:00:00
### queue for submission
#PBS -q gpuX

### Merge output and error files
#PBS -j oe

### request 2 GPU nodes each with 32 CPUs, 4 GPUs, 4 MPI ranks & 8 OpenMP threads
#PBS -l select=2:mem=375G:ncpus=32:ngpus=4:mpiprocs=4:ompthreads=8

### start job in the directory it was submitted from
cd $PBS_O_WORKDIR

### load the necessary software module
module load gcc/6.5
module load cuda/10.1
module load openmpi/2.1.5-gcc_4.8.5-psm2-cuda10.1
# module load openmpi/4.1.0-gcc_4.8.5-psm2-cuda10.1

export PSM2_CUDA=1
export PSM2_GPUDIRECT=1

### run the application and provide its command line arguments
mpirun ./sim_1

### Note that more applications/shell commands may be added here
### (e.g. for post-processing of the results)
```

5.4.5 MATLAB (Mathematica, R, Julia, ...) Jobs

Assume that we have a **MATLAB script** named `myscript.m` stored in the current directory. The job submission script for a (single-threaded) non-interactive MATLAB job may look like this:

```
#!/bin/bash
```

```
### Job Name
#PBS -N MATLAB_test
### required runtime
#PBS -l walltime=01:00:00
### queue for submission
#PBS -q cpu_a

### Merge output and error files
#PBS -j oe

### Request 16 GB of memory and 1 CPU core on 1 compute node
#PBS -l select=1:mem=16G:ncpus=1

### start job in the directory it was submitted from
cd $PBS_O_WORKDIR

### load the necessary software modules
module load MATLAB/R2020a
# module load Mathematica/12.0.0
# module load R/3.5.2
# module load julia/1.5.2

### run the script:
### ---------------

### ... for MATLAB R2018b or older
#matlab -nodisplay -r "myscript; quit;"
### ... for MATLAB R2019a or newer
matlab -batch "myscript"

### ... for recent Mathematica versions
#wolframscript -script myscript.wls

### .. for R
#R CMD BATCH myscript.r

### ... for Julia
#julia myscript.jl
```

The script also indicates (in the commented-out lines) that an analogous approach can be adopted for running **Mathematica**, **R**, **Julia**, and other console-based jobs limited to a single compute node and using one or more threads (set the `ncpus` resource accordingly).

5.4.6 Python Jobs

Python versions 2.7 and 3.6 are readily installed on all nodes (no `module load` command is necessary). In addition, more recent Python version(s) are gradually being added as modules. It is recommended that one creates a Python virtual environment for their work so that custom Python packages can be installed via `pip` as necessary.

```
[stracpav@login1 ~]$ python3.6 -m venv my_virtual_env
```

or

```
[stracpav@login1 ~]$ module load python/3.10.9
[stracpav@login1 ~]$ python3 -m venv my_virtual_env
```

Provided that the job is submitted from within the `my_virtual_env` directory containing the Python script `myscript.py`, it can look e.g. like this:

```
#!/bin/bash
### Job Name
#PBS -N python_script
### required runtime
#PBS -l walltime=01:00:00
### queue for submission
#PBS -q cpu_a

### Merge output and error files
#PBS -j oe

### Request 16 GB of memory and 1 CPU core on 1 compute node
#PBS -l select=1:mem=16G:ncpus=1

### start job in the directory it was submitted from
cd $PBS_O_WORKDIR

# activate the Python virtual environment:
# Note that once activated, the Python version used when creating
# the virtual environment is readily available and the respective module
# need not be loaded my means of "module load python/X.X.X".
source bin/activate

### run the application
python myscript.py
```

5.4.7 ANSYS Jobs

The case setup is usually prepared using the ANSYS GUI, see Section 5.5. The actual simulations using ANSYS products (e.g. Fluent) can then be run as batch jobs provided that

- each job runs on a single compute node only,
- each job uses no more than 16 CPU cores (academic license limitation),
- there is an available license in the license pool at the moment of execution.

The job submission script for an MPI-parallel non-interactive ANSYS (Fluent) job may look like this:

```
#!/bin/bash
### Job Name
#PBS -N ANSYS_Fluent_case
### required runtime
#PBS -l walltime=01:00:00
### queue for submission
#PBS -q cpu_a

### Merge output and error files
#PBS -j oe

### Request 60 GB of memory and 16 CPU cores on 1 compute node
#PBS -l select=1:mem=60G:ncpus=16:mpiprocs=16

### start job in the directory it was submitted from
cd $PBS_O_WORKDIR

module load ANSYS/19.1
### Use the
export PATH=$ANSYS_BASE/fluent/bin:$PATH

### direct PBS Pro support is not set up in ANSYS,
### so we obtain the MPI parameters as follows:
NPROCS=`wc -l < $PBS_NODEFILE`
### run ANSYS Fluent
fluent 3ddp -t${NPROCS} -p -cnf=$PBS_NODEFILE -mpi=openmpi -g -i My_Case.jou > log.txt
```

Notice how the `ANSYS_BASE` environment variable can be used to set relative path to the individual ANSYS product (e.g. Fluent) executable.

5.4.8 OpenFOAM Jobs

Integrating PBS Pro job submission into the OpenFOAM traditional

$$\texttt{Preprocess} - \texttt{decomposePar} - \texttt{runParallel} - \texttt{Postprocess}$$

chain is a little bit more complicated in comparison to the other procedures described in this manual. Please contact support if you intend to run non-interactive parallel OpenFOAM simulations.

5.5 Running Interactive Jobs

Sometimes it is useful or required to work on a compute node interactively.

- To start a minimal interactive session that you believe will be completed in 10 minutes, issue the command

  ```
  [stracpav@login1 ~]$ qsub -I -q cpu_a -l walltime=0:10:00
  ```

  Note that omitting the wall-time setting (i.e. leaving the queue's default wall-time) may postpone the scheduling of your job as the scheduler will wait until a long enough time frame is available. Omitting the host-level resources (mem, ncpus) will reserve the poor defaults (1 CPU core and 256 MB of RAM).

- To start an interactive session where parallel processing on 8 CPU cores and 20 GB of memory is required, issue the command

  ```
  [stracpav@login1 ~]$ qsub -I -q cpu_a -l walltime=0:10:00 -l select=1:mem=20G:ncpus=8
  ```

Interactive jobs may be used to run **GUI-based** applications that require substantial computing power directly on the compute nodes.

- For example, to start ANSYS on a type-B node, issue the commands

  ```
  [stracpav@login1 ~]$ qsub -IX -q cpu_b -l walltime=10:00:00 -l select=1:mem=375G:ncpus=16
  qsub: waiting for job 1584.login1 to start
  qsub: job 1584.login1 ready

  [stracpav@node19 ~]$ module load ANSYS/19.1
  [stracpav@node19 ~]$ runwb2
  ```

  Note: The above code will start the ANSYS Workbench on one of the type-B compute nodes and display it on your local screen using X11 forwarding. The same can be done from an X2Go remote desktop session which is particularly convenient if you are not in the local CTU network and X11 forwarding is very slow. **MATLAB**, **Mathematica**, **R-Studio**, **ParaView** etc. can be run in the same fashion.

5.5.1 Jupyter Notebook

It is possible to connect to a running Jupyter Notebook server from the user's local machine provided that an SSH tunnel is established to the compute node. Starting from scratch, one can follow the procedure below:

1. As in Section 5.4.6, create a virtual environment and install Jupyter Notebook:

```
[stracpav@login1 ~]$ python3.6 -m venv jupyter_virtual_env
[stracpav@login1 ~]$ cd jupyter_virtual_env
[stracpav@login1 ~/jupyter_virtual_env]$ source bin/activate
(jupyter_virtual_env.... ]$ pip install --upgrade pip
(jupyter_virtual_env.... ]$ pip install jupyter
```

Optionally, one can set up Jupyter to listen on all interfaces by default, which requires generating a configuration file

```
(jupyter_virtual_env.... ]$ jupyter notebook --generate-config
```

and then editing the file `~/.jupyter/jupyter_notebook_config.py` so that the following lines are uncommented and changed accordingly:

```
c.NotebookApp.allow_origin = '*'
c.NotebookApp.allow_remote_access = True
c.NotebookApp.ip = '*'
```

2. Once the setup is completed, create an interactive job and start Jupyter Notebook on the compute node:

```
[stracpav@login1 ~]$ qsub -I -q cpu_a -l walltime=1:00:00 -l select=1:mem=8G:ncpus=1
qsub: waiting for job 139604.login1 to start
qsub: job 139604.login1 ready

[stracpav@node11 ~]$ cd jupyter_virtual_env
[stracpav@node11 ~/jupyter_virtual_env]$ source bin/activate
(jupyter_virtual_env.... ]$ jupyter notebook --no-browser --ip=node11
[I 12:30:53.266 NotebookApp] Serving notebooks from local directory: ....
[I 12:30:53.266 NotebookApp] The Jupyter Notebook is running at:
[I 12:30:53.266 NotebookApp] http://node11:8888/?token= ...
[I 12:30:53.267 NotebookApp] Use Control-C to stop this server ...
[C 12:30:53.286 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///mnt/lustre/helios-home/stracpav/.local/share/jupyter/ ...
    Or copy and paste one of these URLs:
        http://node11:8888/?token=82b67a9f3e29b318bd306 ...
```

If the optional part of step 1 has been performed, the "`--ip`" flag is not required.

3. Now open a new SSH connection to Helios, using port forwarding to forward e.g. local port 8888 to the remote port 8888 on node11:

```
ssh -L 8888:node11:8888 stracpav@helios.fjfi.cvut.cz
```

4. Finally, open a local browser and paste the suggested URL (with the correct token) to the address line, replacing the node name by ”localhost”:
   http://localhost:8888/?token=82b67a9f3e29b318bd306cdc773faa8d3d8eb575320ef4ad
5. After shutting down Jupyter, close the second SSH connection to Helios and also leave the interactive job.

5.6 Job Management

- To list all jobs

```
[stracpav@login1 ~]$ qstat
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
270.login1        STDIN            eichler           1920:34: R cpu_a
275.login1        STDIN            eichler           707:11:1 R gpu
1527.login1       Xh-Exp03         zakalexa          429:04:5 R cpu_a
1537.login1       Xh-Exp13         zakalexa          429:11:5 R cpu_a


[stracpav@login1 ~]$ qstat -a

login1:
                                                        Req'd  Req'd   Elap
Job ID      Username Queue  Jobname  SessID NDS TSK Memory Time  S Time
----------- -------- ------ -------- ------ --- --- ------ ----- - -----
270.login1  eichler  cpu_a  STDIN    103164   1  16    --    --  R 791:4
275.login1  eichler  gpu    STDIN     76047   1   1    --    --  R 772:3
1527.login1 zakalexa cpu_a  Xh-Exp03 127224   1  16    --  100:0 R 26:49
1537.login1 zakalexa cpu_a  Xh-Exp13 123656   1  16    --  100:0 R 26:50
```

- To print a list of job IDs fulfilling the given criteria

```
[stracpav@login1 ~]$ qselect -u zakalexa
1527.login1
1537.login1
```

- To delete a job with the given ID (the full ID needn’t be given as long as the job identification is unique)

```
[stracpav@login1 ~]$ qdel 1527
```

- To delete all my jobs

```
[stracpav@login1 ~]$ qselect -u $USER | xargs qdel
```

- To change job parameters (required resources etc.): Use the `qalter` command. Note that changing parameters of already running jobs is barely possible for obvious reasons.

5.7 Direct SSH Access to Compute Nodes

Users having a running job (either batch or interactive) can connect to the respective compute nodes directly via SSH. The nodes are on the internal network and are accessible from the login node only unless an SSH tunnel is created. The SSH session is automatically terminated by PBS once the job finishes.

- To find out which nodes your job no. `154524` runs on, use the following:

```
[stracpav@login1 ~]$ qstat -f 154524 | grep exec_host
    exec_host = node01/0*32+node02/0*32
```

- In this example, the job is an MPI job which uses 32 cores on `node01` and another 32 cores on `node02`. To log in to `node01`, type

```
[stracpav@login1 ~]$ ssh node01
```

6 Storage Space

6.1 Home Directories

Each user has their home directory available on the login node as well as on all compute nodes. The home directory is mounted under

`/mnt/lustre/helios-home/USERNAME`

Quotas are currently not applied on the home directories. The whole file system has roughly 180 TB of usable space available for both the regular users and the BioCCS/U team members. Quotas may be introduced later if excess capacity use by some individual users becomes an issue.

6.2 Scratch Space

On each compute node and for each user, there is a directory (a scratch space)

`/scratch/USERNAME`

which is available for writing intermediate results as the job runs. The local SSD storage is much faster than the Lustre file system if thousands of small files need to be handled. After the job finishes, any files that it left in the scratch directory remain there for **7 days**. After that, they are deleted automatically.

It is not so easy to access the scratch space on an individual node once the job finishes. In order to collect all relevant data from the scratch space before the job finishes, write a submission script in the following fashion:

```
### PBS Job submission setup
.
.
```

```
.
### create a subdirectory in the user's scratch area for each job, based on
### a unique job ID
### (so that more jobs don't interfere if they meet on a single compute node)
OUTPUT_SCRATCH=/scratch/$USER/$PBS_JOBID
mkdir $OUTPUT_SCRATCH
.
.
.
### run the application which writes to the scratch directory
./my_app.exe -output $OUTPUT_SCRATCH
.
.
.
### do some post-processing and store the results in the user's home directory
.
.
.
### be nice to others and clean up what you left in the scratch space
rm -rf $OUTPUT_SCRATCH
```

7 Remote Visualization

For visualization of large datasets, **ParaView** and **VisIt** software tools are installed and available as modules (see Section 4.1). It is particularly easy to **start a parallel ParaView remote visualization server** on the login node (with direct high speed access to the user's home directory on the Lustre filesystem) and connect to it from the user's workstation using **an identical version** of ParaView installed locally. The procedure is as follows:

1. log in to Helios, creating an SSH tunnel to forward the port 11111 of **login1** to your local machine:

   ```
   ssh -L 11111:localhost:11111 USERNAME@helios.fjfi.cvut.cz
   ```

   On Windows, use the port forwarding features of **PuTTY**.

2. On Helios login node (using the SSH shell created in step 1), launch the **ParaView server**. The MPI implementation readily shipped with ParaView can be used to start multiple processes of the server (see Section 4.1.1 for details on running different versions of ParaView):

   ```
   [stracpav@login1 ~]$ module load paraview/5.8.1-headless
   [stracpav@login1 ~]$ mpiexec -np 4 pvserver
   ```

3. On the local machine (either Linux or Windows), start ParaView and establish a new server connection (from the "**File ▸ Connect**" menu entry), using "**localhost**" as the host name and "**11111**" as the port. In the next step, choose that the server is started manually (as it is already started by you).

4. Connect to the server. From now on, the remote file system is accessible through the ”Open” dialog and all rendering is performed on the remote server. Only the ParaView user interface displaying the rendering results runs on the user's machine.
5. When finished, choose ”File ▹ Disconnect” or simply close ParaView. This will terminate the ParaView server as well.

8 Support

For support, please contact Pavel Strachota.

References

[1] PBS Professional 18.2 User's Guide https://www.pbsworks.com/pdfs/PBSUserGuide18.2.pdf

[2] PBS Professional 18.2 Reference Guide https://www.pbsworks.com/pdfs/PBSRefGuide18.2.pdf

[3] Environment Modules http://modules.sourceforge.net/