Some basic information on using the Barkla HPC cluster

July 2021

Cliff Addison and Manhui Wang

Advanced Research Computing

Overview of the Liverpool Barkla HPC cluster

The Liverpool Barkla HPC cluster is available for demanding research computing tasks.

Barkla is a collection of many individual computer "nodes", each with multiple "cores" (computing elements which typically do one task each). Each of these may appear little or no faster than some high-end desktop systems, but there are over 5800 cores, they have a higher than usual amount of memory each, and various other hardware advantages over desktop PCs.

In particular, they are designed to work together in parallel, with fast communication between them and onto a large shared disk storage area (i.e. all the nodes see essentially the same storage). This might well allow demanding calculations or data manipulations, if written to work in parallel on many cores at once, to run 10 or more times faster, and typically significantly more efficiently in various respects — including your time once you are used to the system. The management of the system is done for you, and advice is available on how best to use it to your work.

Users normally access the system by logging in to a "login node" (sometimes called the "head node") over the network. The login node is for light use, such as editing input files, compiling your own programs and, most importantly, submitting "jobs" to run the relevant tasks on one or more of the many "compute nodes" connected to it. Programs which run for a long time or take a lot of memory should not be run on the login node, and there are some restrictions on what you can do on it to prevent other users being affected.

The cluster has management software to support shared use by many users simultaneously. This sort of "batch" environment may be unfamiliar, but is fairly straightforward. To do some work, it is necessary to compose a "script" (or often a command will do it for you) and type a command to submit the job to the batch queue. It will then run with whatever resources (such as number of cores) you've asked for. The job sits in a queue along with other users' jobs until the resources are available to run it (which might be immediately if the cluster is lightly loaded). It will then run unattended — perhaps while you're asleep — and eventually produce some output. You can either check on its progress or get email when it has finished.

The operating system used on the cluster is GNU/Linux — not Microsoft Windows. It is typically (but not necessarily) used by typing at a "command line" in a "terminal emulator" window, not with a GUI environment (although GUIs are available). The main tasks you are likely to want to do are:

- editing job scripts or input data for the jobs you want to run;
- submitting and monitoring the jobs;
- dealing with the output, in terms of just reading it, manipulating it, or transferring it somewhere else.

A brief overview of Linux with links off to related topics can be found at Opensource.com. Another introduction to Linux and its basic commands can be found at Surrey EE Unix Tutorial (at the starting level, Linux, Mac Darwin and UNIX commands are identical)

A short, command summary can be found at Beginner Linux Tutorial. There are other starting topic on this web-site as well.

There is also a 25-minute video posted at Linux Tutorial Series - Part 01 that claims to provide an introduction to Linux.

Scripts or input files can be transferred to the cluster from elsewhere, e.g. you could edit things on your PC. The general remote access mechanism is "ssh", variations of which can be used to copy files between systems. See the detailed documentation. ssh also allows you to send output from a graphical program running on the cluster for display on your desktop if the desktop supports the remote graphics protocols used. Again see the detailed documentation.

What you need to do in this environment are usually fairly simple, even if it looks intimidating initially, and people quickly get used to it, especially if — as for most users — it's a question of running one or two existing programs for all your work. It may also be possible to automate or abstract much of this by using scripts to glue together sub-tasks.

In the text below, references of the form *name*(*number*), such as "ssh(1)" refer to the page describing *name* in section *number* of the "man" system. Normally you can just type `man` *name* to read it.

Connecting to the system

Barkla HPC cluster at the University of Liverpool is a modern system with high capability nodes, including large memory nodes, Xeon Phi accelerator nodes, GPU nodes, and visualisation nodes.

Barkla consists mainly of 138 standard compute nodes, 2 large memory nodes, 4 Xeon Phi accelerator nodes, 2 visualisation nodes, 2 login nodes and 5 GPU nodes with a fast Intel Omnipath interconnect in between. Each standard compute nodes has 40 cores (two 20-core Intel Xeon Gold Skylake processors) and 384 GB of memory, so 9.6 GB of memory per core. In addition, there are two 40-core, 1 TB shared memory systems for large memory jobs along with 5 GPU nodes equipped with four Nvidia NVLink P100/V100 GPUs. These nodes were state of the art as of winter 2017. Users could log into login nodes and visualisation nodes directly, while are recommended not to log into compute nodes (i.e., 138 standard compute nodes, 2 large memory nodes, 4 Xeon Phi accelerator nodes, and 5 GPU nodes) directly. The compute nodes are manged by Slurm job scheduler, and users can submit jobs to these compute nodes.

The login nodes of Barkla HPC cluster at Liverpool are:

`barkla4.liv.ac.uk (login1)` and `barkla5.liv.ac.uk (login2)`

Users can also connect to Barkla via two visualisation nodes:

`barkla6.liv.ac.uk (viz01)` and `barkla7.liv.ac.uk (viz02)`

Users are recommended not to run jobs (including graphical applications) directly on login nodes (login1 & login2 of Barkla), as these login nodes are for light use, such as editing input files, compiling your own programs and, most importantly, submitting jobs. Users should submit jobs via Slurm job scheduler on Barkla. Users could test and debug lightweight applications (including GPU, Graphical User Interface applications) directly on the two visualisation nodes of Barkla (viz01 and viz02, or barkla6.liv.ac.uk and barkla7.liv.ac.uk), and on the visualisation nodes users can also submit jobs to the queue as they can act as head nodes as well. Users could log into the two visualisation nodes directly from local machines. If users need to carry out massive data transferring or time-consuming compilation/building, please do it on the visualisation nodes rather than login nodes. Each visualisation node has 40 CPU cores and 380GB memory, while each login node is a virtual machine with just two CPU cores and 32GB memory.

Access is only possible using the Secure Shell (via PuTTy on Windows systems or `ssh(1)` or other clients on Linux / UNIX / Cygwin systems). The MobaXterm installation available via Install University Apps under Utilities is an easy way to access a subset of Cygwin so you have ssh access and X-windows graphics windows on your Windows system. There is also a PuTTy installation available via Install University Apps under Internet that provides an ssh connection to remote systems and then puts you into a standard Linux Command environment.

When using ssh, it is possible to establish an X-windows connection to the system if you use the command:

`ssh -X` *myusername*`@barkla4.liv.ac.uk` (or `ssh -X` *myusername*`@barkla5.liv.ac.uk`)

Nearly all users will connect with the same username and password as used on the University web-mail or Managed Windows Service. You will not need to specify *myusername* if you are connecting from that same user account on a different machine. The visualisation nodes (barkla6 and barkla7) have been properly configured, and there are separate guides about how to use them. You may find it convenient to configure X11 forwarding by default to local machines (see `ssh_config(5)` for OpenSSH, and note it is a security risk to do this to untrusted machines), and to use password-less login by copying your public key with, say, OpenSSH's `ssh-copy-id(1)` and running an agent (e.g. `ssh-agent(1)/keychain(1)`) to avoid typing your passphrase every time.

Command line companions to ssh are

- `scp` (to copy over files, possibly in a hierarchy) e.g. `scp -rp mydir myusername@barkla4.liv.ac.uk:mydirectory`
- `sftp` (an alternative, but usually less convenient file-transfer program) e.g. `sftp myusername@barkla4.liv.ac.uk`

File transfers from Windows systems can be done with the above commands from MobaXterm or by using the `pscp` or `psftp` features of PuTTy, which are PuTTy variants of the standard `scp` and `sftp` commands. You can also launch a SFTP graphic interface session within MobaXterm to drag-and-drop files/folders between your computer and the remote HPC cluster.

Once you are logged on, you are presented with a standard GNU/Linux environment. Various editors are available, such as `vi` (console mode), `emacs`, `nano` and `gedit`.

Connecting to the system off campus (working from home)

The connection to Liverpool HPC cluster Barkla is limited to the University IP addresses. However, you could still get access to the cluster off campus. There are several approaches to achieve it.

1. You could connect to the system with the Liverpool VPN. For details about how to register VPN, please search "VPN" at the CSD Service Desk Knowledge website https://liverpool.service-now.com/ess/common_answers.do, or visit the knowledge article at https://liverpool.service-now.com/kb_view.do?sysparm_article=KB0010407). With VPN it works as if your computer is at the Liverpool University Campus. So you can gain full access to any available service (including access to Barkla with X Windows and downloading papers from the library) as you do on campus. We have tested the access to the HPC clusters with X Windows off campus via VPN regularly, and it works pretty well. Please note the following issues may affect the connection: (1) You may have to update Cisco AnyConnect to the latest version; (2) VPN connection requires stable network and reasonable speed (sometimes wireless connection may not meet such requirements); and (3) Any attempt of connection should start only after VPN is well connected.

2. You could use one of worldwide accessible University Unix/Linux servers (lxb.liv.ac.uk and lxc.liv.ac.uk) as the intermediate server. Please note that two-factor authentication with Duo may be required (please visit the knowledge article "Getting started with Linux and ssh" at https://liverpool.service-now.com/kb_view.do?sysparm_article=KB0012193). These servers can be accessed with the same username and password as the clusters. There is relatively large volatile space on lxb/lxc, which can be used as the transit when transferring large data. You need to log into the intermediate server first (with X Windows or command line ssh), and then log into the cluster using command line ssh on the intermediate server.

3. You could connect to the system using Apps Anywhere (https://www.liverpool.ac.uk/csd/apps-anywhere/). Once you log into Apps Anywhere, select PuTTY or MobaXterm from the Utilities group and login as you do locally. Apps Anywhere is available either through your web browser (such as Internet Explorer) or through a stand-alone application called Citrix Workspace. This approach could be very handy (via web browser) but it may be slow sometimes. You may connect briefly to check on the status of a job or to submit an already existing script with minor edits.

4. You could log into your office MWS PC off-campus via Remote Desktop (see https://www.liverpool.ac.uk/csd/working-from-home/faqs/, and search "Remote Desktop") when your VPN is connected, then log into the cluster with PuTTY or MobaXterm on your office MWS PC. In order to do this, please follow these two sets of instructions: (1) Preparing to use your MWS PC from home via Remote Desktop - *this needs to be done while you're still on campus*;

and (2) Using your MWS PC from home with Remote Desktop. This approach may take some efforts to set up, but it is most powerful as you work in the office.

Basic environment

When you log into Barkla, you will start in your home directory. The absolute path to this directory is `/users/`*`myusername`* (e.g. `/users/caddison`). In addition you have three other top-level directories from which you can work. These are volatile directory `/mnt/data1/users/`*`myusername`* (e.g. `/mnt/data1/users/caddison`) on NFS file system, scratch directory `/mnt/lustre/users/`*`myusername`* (e.g. `/mnt/lustre/users/caddison`) on Lustre file system, and temporary directory `/tmp/users/`*`myusername`* (e.g. `/tmp/users/caddison`) on local file system. From users' point of view, volatile and scratch directories are in shared file systems and can be accessible on all nodes (including login nodes login1/2, visualisation nodes viz01/02 and compute nodes), while temporary directory is in a local file system and can be accessible only on one specific node. That is to say, on all nodes, users can see the exactly same files/subdirectories in volatile and scratch directories, but the files/subdirectories in temporary directory could be different (these files are normally generated when running programs on different nodes).

Your home directory has a quota on its size, to enable backing up all of the user home directories. It is **not** intended to be used as the target directory for most of the jobs on the cluster. It is intended to hold source files, small data files and those important job files that are difficult to reproduce. Space and number of files in your home directory are limited (normally up to 50 GB space and 100k files per user). If you run out of quota, you may find it difficult to get anything done.

The command `quota -s` will give you information on your current quota and how much of that quota you have used.

Your volatile directory (`/mnt/data1/users/`*`myusername`*, which is linked by a symbolic link `volatile` in your home directory) is the one that should normally be the one from which jobs are submitted and to which output is written. This directory has no user quotas on it, but it does have a physical limit. Users are kindly asked to manage space on this system sensibly and to not allow old and irrelevant output files and the like to fill up space on it. The volatile space is **not** backed up, hence the name. Important files should be removed from your volatile area to someplace more secure as part of your normal workflow management.

Your scratch directory (`/mnt/lustre/users/`*`myusername`*, which is linked by a symbolic link `sharedscratch` in your home directory) is very similar to your volatile directory. It hangs off a file system that supports parallel IO, which is particularly relevant for some applications such as Fluent and for some application domains that make use of standard IO systems such as netcdf or hdf5, both of which support parallel IO. In addition, your scratch directory might be a good target for job output files if these job output files are large (say several hundred megabytes or larger). Please note: the name "sharedscratch" only means the directory is a scratch directory shared across the nodes, but it is actually invisible to other users in the cluster.

Your temporary directory (`/tmp/users/`*`myusername`*, which is linked by a symbolic link `localscratch` in your home directory) is on the local file system of the specific node. It has relatively fast read/write speed, and won't cause any burden to the network traffic between the cluster nodes. When a job is submitted to certain nodes, such temporary directory is assigned/created for each node. The files in the temporary directory are subject to automatic deletion after a certain time (currently 30 days).

Data sharing

All the home, and volatile, scratch directories should be set as read/write/execute access only for the owner when the account was created on Barkla. No other users can get access to these directories initially. Some users might change the default access permissions later in order to share data, but please be very careful when doing so. Incorrect access permission settings could put your data at high risk of being accidentally deleted by a random user. There are two approaches to share files/directories among users on Barkla.

1. If you need to share data with a group of users, we can create a specific group for you, and add new users to the group as required. So it is relatively easy to share directories/files in the specific group with the commands "chown" and "chmod". All users were set the primary groupship as "clusterusers" initially on Barkla when the accounts were created. You could see your groupship via command "groups" or "id <username> ". It is recommended not to share data within the general group "clusterusers" as this group includes all users on Barkla (around 400 currently). If you want to share a directory with your group members, you have to change the directory's groupship "clusterusers" to your additional groupship using the command "chown". Then you could use "chmod" command to grant access permissions (it is best to grant just read/execute, but not write permissions) for users in the same group. Please don't grant any permission for other users who are not in your specific group. Please note this will grant access permissions to all users in the specific group rather than a specific user, but you could change it back if you don't want to share it. You could find lots of tutorials about the commands "chown" and "chmod". For example, https://www.howtoforge.com/linux-chown-command/ https://www.howtoforge.com/tutorial/linux-chmod-command/

Here is an example how to share a directory within your specific group. If user `mhwang` wants to share directory `/mnt/data1/users/mhwang/shareddata` (read/execute permissions, NOT write permission) with group users in group `siteadmins`, the following commands could work:

```
chown mhwang:siteadmins /mnt/data1/users/mhwang

chown -R mhwang:siteadmins /mnt/data1/users/mhwang/shareddata

chmod u=rwx,g=rx,o= /mnt/data1/users/mhwang (or chmod 750 /mnt/data1/users/mhwang)

chmod -R u=rwx,g=rx,o= /mnt/data1/users/mhwang/shareddata
```

and prohibit the read/write/execute access for all other directories in `/mnt/data1/users/mhwang`:

```
chmod -R u=rwx,g=,o= /mnt/data1/users/mhwang/<all other directories>
```

If you don't want to mix your own data with your group data, we could also create a group directory to share data for a specific group. The group directory is initially owned by a user in the group, and the ownership can be changed later as requested.

Please note that write access for group and other users in any sub-directories in your shared directory could be very dangerous, and your data writable to group/other users are at high risk of being accidentally deleted by group/other users.

2. If you want to share data with a specific user, you could grant finer permissions for this specific user using "setfacl"on Barkla. This should work on all of the directories, but it does take a bit of getting used to. "man setfacl" can provide some details. The command "getfacl" can be used to examine the ACLs (access-control list) of a file or directory.

Example:

```
setfacl -R -m u:caddison:rx /mnt/data1/users/mhwang
```

This will enable user "caddison" read/execute access to all files and directories in `/mnt/data1/users/mhwang`.

If only a sub-directory `shareddata` is needed then a two-step process is needed:

(1) Provide rx access to just the top-level directory

```
setfacl -m u:caddison:rx /mnt/data1/users/mhwang
```

(2) Then provide rx access to the sub-directory

```
setfacl -R -m u:caddison:rx /mnt/data1/users/mhwang/shareddata
```

Module files

A standard difficulty when running an application or compiling a program on a computer is the need to specify the location of binary files, libraries and the like so things will run correctly. Module files are a set of files that allow you to modify the necessary environment variables for particular applications or library components.

Modules are available on the login/head nodes and compute nodes without needing to be set up in login scripts. Users could load the relevant modules in the session from which you submit a job, or load the relevant modules inside the job script. By default, all environment variables are propagated when you submit a job via Slurm job scheduler. Please note that (1) if you load the modules on head/login nodes (without loading modules inside the job script), you will need to load the relevant modules very time when you submit your job; (2) if you load the relevant modules inside the job script, please do add "`module purge`" before loading modules inside the job script. In practice, loading modules inside the job script could avoid any interference from the session where you submit a job.

You can see which modules are available using the command `module avail`.

```
module avail
---  /users/siteadmin/gridware/personal/el7/etc/modules  ---
  apps/hpl/2.1/gcc-4.8.5+openmpi-1.8.5+atlas-3.10.2
  apps/python/2.7.8/gcc-4.8.5
  apps/R/3.3.0/gcc-4.8.5+lapack-3.5.0+blas-3.6.0
  compilers/gcc/system *default*
  libs/gcc/system
  mpi/openmpi/1.10.2/gcc-4.8.5
  mpi/openmpi/1.8.5/gcc-4.8.5
  null
---  /opt/gridware/local/el7/etc/modules  ---
  apps/cmake/3.5.2/gcc-4.8.5
  apps/memtester/4.3.0/gcc-4.8.5
  compilers/gcc/5.1.0
  compilers/gcc/5.5.0
  compilers/gcc/system *default*
  compilers/intel/2018u1
  libs/gcc/5.1.0
  libs/gcc/5.5.0
  libs/gcc/system
  libs/intel/2018u1
  libs/intel-mkl/2018u1/bin
  libs/nvidia-cuda/8.0.61/bin
  mpi/intel-mpi/2018u1/bin
  mpi/openmpi/1.10.7/gcc-4.8.5
  null
---  /opt/clusterware/etc/modules  ---
  null
  services/clusterware
  services/git
  services/pdsh
  services/s3cmd
---  /opt/apps/etc/modules  ---
  apps/MATLAB/R2017b
  apps/singularity/2.4
```

To see loaded modules:

```
module list
```

To unload a module, for example,

`module unload mpi/openmpi/1.10.7/gcc-4.8.5` [or whatever the module name is.]

To load a module, for example,

```
module load mpi/openmpi/1.10.7/gcc-4.8.5
```

To display information about a module, for example,

```
module show compilers/gcc/5.5.0
```

This will list the full path of the modulefile and the environment variables set up by the module if loaded.

Using the Slurm job scheduling system

Users can only run jobs on the compute nodes of the cluster by going through the Slurm job scheduling system. Slurm is documented in man pages on the system. The detailed Slurm documentation can be found at https://slurm.schedmd.com/. A two-page command/option summary can be downloaded at https://slurm.schedmd.com/pdfs/summary.pdf. Some introduction can be also found at Alces-flight (who is the provider of Barkla together with Dell) website at http://docs.alces-flight.com/en/stable/slurm/slurm.html.

 Using the Slurm for job submission means that parallel jobs run on a set of dedicated nodes (no other job can run on the same set of nodes) while the number of serial (scalar) jobs on a node depends upon the number of cores available.

For those who are familiar with Sun Grid Engine (SGE), it is relatively easy to port the job script to Slurm as both job schedulers' concepts are quite similar. Tables 1, 2 and 3 in Appendices list some command user commands, environmental variables, and job specifications for both SGE and Slurm.

If a user wanted to run the test code "hello_world" with 1 core of a compute node, then a Slurm job script called sbatch_hello_world.sh might look something like:

```
#!/bin/bash -l
# Use the current working directory, which is the default setting.
#SBATCH -D ./
# Use the current environment for this job, which is the default setting.
#SBATCH --export=ALL

date
echo "This code is running on "
hostname
#echo "Starting running on host $HOSTNAME"
./hello_world
echo "Finished running - goodbye from $HOSTNAME"
```

Please note: the "-l" option on the first line of the Slurm job script is necessary. This will request a login session, and ensure that environment modules can be loaded as required as part of the script.

The command "sbatch" is used to submit the job to the Slurm scheduler:

```
[siteadmin@login1[barkla] test]$ sbatch sbatch_hello_world.sh
Submitted batch job 2122
```

The output file would be, for example, slurm-2122.out which looks like this:

```
[siteadmin@login1[barkla] test]$ cat slurm-2122.out
Tue  9 Jan 16:00:55 GMT 2018
This code is running on
node026.pri.barkla.alces.network
Hello, World!
Finished running - goodbye from node026.pri.barkla.alces.network
[siteadmin@login1[barkla] test]$
```

When a job is submitted to the Slurm, it is queued and a basic check is made that the resources required (e.g. number of cores, memory) are available. Once the job starts execution, the job script is executed on the first (possibly only) compute core. Information about special environment variables, where the executable and input files are located and where the output files should be located all need to be passed to this compute core. Without any additional information, the default environment is that specified via the user's .bashrc file and the default is that any relative directory path for executables/files etc. is relative to the user's home directory. By default both standard output and standard error are directed to a file of the name "slurm-%j.out", where the "%j" is replaced with the job allocation number.

A slightly more sophisticated example of a serial job script is:

```
#!/bin/bash -l

# Specify job name
#SBATCH -J PI_sequential
# Specify where standard error and output should be sent
#SBATCH -e sequential_errors
#SBATCH -o sequential_output
# Specify the current working directory as the location for executables/files
# This is the default setting.
#SBATCH -D ./
# Export the current environment to the compute node
# This is the default setting.
#SBATCH --export=ALL

# Actually do something!

sequential_pi < infile
```

A range of other options can either be specified in the job script file or in the sbatch command line itself. The most useful of these is specifying a maximum time limit on your job that is different from the queue default time (currently 8 hours). The maximum time at present is 3 days (i.e. 72 hours) for standard users, but only if this time is requested explicitly. For dedicated users who can get access to the dedicated partitions/nodes, the maximum time limit can be extended as per your request.

For instance, to specify a maximum time of 12 and a half hours, the job script could include the line:

```
#SBATCH -t 12:30:00
```

To specify this limit from the qsub command line, you could use:

```
sbatch -t 12:30:00 myjobscript
```

The standard compute nodes have about 380 GB of memory, or 9.5 GB per core. Slurm job scheduler on Barkla has been configured with default memory per core for all partitions. The default memory per core is normally the optimal setting for all nodes, and the total memory for the job is set automatically based on the number of cores. Therefore, users are recommended not to specify the memory in the job script or command line manually. Incorrect memory setting might cause job pending or oversubscribe the resource like CPU cores.

For parallel jobs, users need to request both the numbers of nodes and cores explicitly. For example,

```
# Request the number of nodes
#SBATCH -N 2
# Request the number of cores
#SBATCH -n 80
```

If the number of nodes is not requested explicitly, the default setting will spread the cores across any number of nodes. If the number of cores is not requested explicitly, the default will be just 1 core per node, even if you have requested the number of nodes. It is recommended to request just one compute node when you request no more than 40 cores, and to request any whole compute nodes when you request more than 40 cores.

Please note that only shared memory parallelisation is supported for some particular packages (such as Gaussian, Matlab, Abaqus, most Python code, OpenMP code, and so on). In these cases, users need to ensure the job runs on a single node by specifying 1 node explicitly. For example,

```
# Request the number of nodes
#SBATCH -N 1
# Request the number of cores
#SBATCH -n 40
```

Any attempt to use multiple nodes for such jobs might cause overloading on the first node, and waste on the remaining nodes (because all processes/threads will run on the first node and the remaining nodes will be actually left idle). Please also ensure the number of cores requested match with the number of processes/threads for your code to avoid overloading problem. For Gaussian jobs, please make sure the number of processes specified by %nproc or %nprocshared in the Gaussian input files is equal to the number of cores. For OpenMP-related (including Python, R) jobs, you could ensure the number of OpenMP threads is equal to the number of cores by adding the following command line in the beginning of Slurm job script (please refer to the Python Slurm job example /opt/apps/Slurm_Examples/python/sbatch_hello.sh):

```
export OMP_NUM_THREADS=$SLURM_NTASKS
```

As mentioned above, when a job script is submitted, it will be issued a job number. This number provides the user with a way to trace the job through the system and to terminate the job if something has gone wrong.

The simplest way to check the status of your jobs in the system is to use the squeue command without any arguments.

squeue

For example, a 40 core parallel job run as user siteadmin shows:

```
[siteadmin@login1[barkla] test]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              2128     nodes sbatch_s siteadmi PD       0:00      1 (Resources)
```

A list of jobs for any one user can be obtained using

squeue -u username

A list of jobs, queued or running, for all users can be obtained using:

squeue

This list can be long and your job information might get overlooked because of all of the other jobs present.

Unfortunately, this squeue information has lines that are something like 114 characters long, so they are wrapped and can be rather difficult to interpret on a normal window. However, you will always be able to identify things such as the job-id, the first part of the job script name, the user name and the state of the job.

Jobs typically pass through several states in the course of their execution. The typical states are PENDING (PD), CONFIGURING (CF), RUNNING (R), SUSPENDED (S), COMPLETING (CG), FAILED (F), and COMPLETED (CD). An explanation of each state follows. The state (ST) field indicates the progress of the job. Initially, the status will be 'PD' indicating that the job is queued and waiting for sufficient resources to become available. The status will then change to 'CF' indicating that the job is being transferred to the compute node and finally the status will change to 'R' when the program is running. When the job has completed, it will disappear from the queue list. An 'F' in the status field signals that an error has occurred. A few likely causes are:

- incorrect permissions on the job script
- incorrect path to the executable
- incorrect permissions on the executable
- input file(s) are not readable
- output file(s) are not writeable

Accidentally submitting a job from another user's directory (that contains the executable of interest) is a common reason for jobs failing.

The command:

squeue – j NNNN

where NNNN is a job number, provides a summary of that jobs status. For example,

squeue – j 2382

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
 2382     nodes  xhpl.sh siteadmi  R    1:56:18     58 node[001-042,081-096]
```

If something is wrong with a job, it can be removed (deleted) from the system with the command:

scancel NNNN

There are other options for squeue and scancel. “squeue –help” gives a complete set of options for squeue, “scancel –help” gives a complete set of options for scancel. Man pages are also available for either (man squeue, man scancel respectively).

One helpful command to list the cluster status is:

sinfo

This provides a snap shot of the current state of all of the job queues on the system and merits some explanation.

There are several partitions (nodes, himem, phi, and gpu, etc) which refer to the corresponding hardware configuration of compute nodes. The command

```
[siteadmin@login1[barkla] MPI_Examples]$ sinfo -Nl
```

(capital N, little l – not one), provides the status of each node on the cluster.

The PARTITION column shows all the partitions in the cluster. The AVAIL column shows the partition state (up or down). The TIMELIMIT column shows the maximum time limit for any user job, and infinite is used to identify partitions without a job time limit. The NODES column indicates the count of nodes with this particular configuration. The STATE column indicates the state of the nodes, and possible states include: allocated, completing, down, drained, draining, fail, failing, future, idle, maint, mixed, perfctrs, power_down, power_up, reserved, and unknown. The NODELIST column indicates the names of nodes associated with this configuration/partition.

Partitions within Slurm on Barkla

A partition is a set of compute nodes grouped logically, and can be considered a job queue, which has an assortment of constraints such as job size limit, job time limit, users permitted to use it, etc. Currently the partitions within Slurm on Barkla include dedicated partitions (`cooper`, `krabbenhoft`, `langfeld`, `rosseinsky`, `theofilis`, `troisi`, `soldini`, and `schaich`), common partitions (`nodes`, `long`, `himem`, `phi`, and `gpu/gpuc`), and low priority partition (`lowpriority`). The dedicated partitions are owned by several specific research groups who have funded these compute nodes (75 nodes in total). Only the users from these specific groups can have exclusive access to the corresponding dedicated partitions. All users can have access to the resource in common partitions. Partition "`nodes`" is the default partition, partition "`long`" is designed for longer jobs (up to 7 days currently), partition "`himem`" consists of two high memory nodes (1.1TB each), partition "`phi`" consists of four KNL Xeon Phi nodes, and there are three GPU nodes in partition "`gpu`" and two GPU nodes in partition "`gpuc`".

```
PARTITION     NODES S:C:T   MEMORY  TIMELIMIT   NODELIST
lowpriority  75    2:20:1  380000  1-00:00:00  node[001-059,104-112,130-136]
cooper        16    2:20:1  380000  10-00:00:00 node[001-016]
krabbenhoft  3     2:20:1  380000  15-00:00:00 node[017-019]
langfeld     8     2:20:1  380000  5-00:00:00  node[020-027]
rosseinsky   20    2:20:1  380000  5-00:00:00  node[028-039,105-112]
theofilis    10    2:20:1  380000  5-00:00:00  node[040-049]
troisi       10    2:20:1  380000  5-00:00:00  node[050-059]
soldini      1     2:20:1  380000  5-00:00:00  node104
schaich      7     2:20:1  380000  5-00:00:00  node[130-136]
nodes*       61    2:20:1  380000  3-00:00:00  node[060-103,113-129]
long         8     2:20:1  380000  7-00:00:00  node[122-129]
himem        2     2:20:1  1100000 6-00:00:00  himem[01-02]
phi          4     1:64:1  192000  3-00:00:00  phi[01-04]
gpu          3     2:12:1  380000+ 3-00:00:00  gpu[01-03]
```

The "`lowpriority`" partition coincides with all the dedicated nodes, and such partition is set up on the basis that the idle resource on the dedicated nodes can be used by other users. The idea is to ensure all the nodes are utilised fully but not at the inconvenience of those who bought the dedicated nodes for their group's use. All users can submit jobs to the "`lowpriority`" partition, but risk being kicked off by the owners at any time. The time limit of jobs in “lowpriority” partition is currently just 24 hours. Part of the reason for this is hinted in the "lowpriority" name. If a user from the dedicated group submits a job to the dedicated nodes and if these nodes are being used via the “lowpriority” partition, the relevant job in the “lowpriority” partition will be restarted on other available nodes, if possible, or requeued/killed otherwise. By default, jobs in “lowpriority” partition will be requeued after being kicked off. If you want to cancel your jobs after being pre-empted, you could add the following line in the Slurm job script:

```
#SBATCH --no-requeue
```

Quality of Service (QOS) within Slurm on Barkla

A QOS can define an associated priority and a set of resource limits. One can specify a QOS for each job submitted to Slurm. Several QOS’s (including `normal, low, dedicated, large`) have been defined within Slurm on Barkla.

`normal`: this is the default QOS, and is used for all partitions by default. MaxCPUsPerUser has been set for this QOS, which is used to limit the maximum CPU cores for all running jobs per user with this QOS (normally 400 cores per user, and it could be adjusted dynamically according to the overall load). User can still submit as many as jobs with this QOS, but jobs won’t get running if the user reaches the maximum limit of CPU cores for the running jobs. The reason for such pending jobs may be shown as “(QOSMaxCpuPerUserLimit)” with “`squeue -l`”.

`low`: this QOS has been attached to “lowpriority” partition as the Partition QOS. MaxCPUsPerUser has also been set for this QOS (normally 400 cores per user, and it could be adjusted dynamically according to the overall load). Users could submit jobs with both “normal” and “low” QOS’s to “lowpriority” partition, but “low” QOS will take effect as Partition QOS will override the job's QOS.

`dedicated`: this QOS has been attached to above dedicated partitions and “phi” partition. MaxCPUsPerUser has not been set for this QOS. Please note this QOS won’t work for other shared partitions (e.g. `nodes, lowpriority`).

`large`: this QOS can be used only for large jobs in “lowpriority” and “nodes” partitions. Users who do need to submit large jobs (normally more than around 400 cores per job) can be added into this QOS as per your request. Meanwhile, we will limit the maximum number of running jobs for the user for all QOS’s to just 1 for fair usage.

A job can request a QOS using the "`--qos=`" option to the sbatch, salloc, and srun commands:

```
sbatch -p lowpriority --qos=low job.sh
```

or add the following lines in the job script (then submit the job as before):

```
# Request the partition
#SBATCH -p lowpriority
# Request the QOS
#SBATCH --qos=low
```

Please note that (1) all jobs will be submitted with the default QOS (i.e. `normal`) if the QOS is not specified explicitly, and (2) the CPU cores for the running jobs with "`normal`" QOS will be counted in across all partitions if no Partition QOS is set up. For users who use the dedicated partitions (including "`phi`" partition) or "`lowpriority`" partition, they can specify QOS "`dedicated`" or "`low`" explicitly. So that the CPU cores with these QOS's won't be counted towards the total CPU cores for QOS "`normal`". For example, users who can use "`cooper`" partition could submit their jobs:

```
sbatch -p cooper --qos=dedicated job.sh
```

Users could submit their jobs to "`phi`" partition:

```
sbatch -p phi --qos=dedicated job.sh
```

Users could submit their jobs to "`lowpriority`" partition:

```
sbatch -p lowpriority --qos=low job.sh
```

Job limits within Slurm on Barkla

Along with the OOS limits (the maximum number of CPU cores across all running jobs by a user is limited), there are also limits for job arrays and maximum submitted jobs: (1) the maximum number of jobs in a job array is 1000, and (2) the maximum number of jobs which can be submitted to the system at any given time for a user is 1000. If any of these limits is reached, new submission request will be denied with an error message:

```
sbatch: error: Batch job submission failed: Job violates accounting/QOS
policy (job submit limit, user's size and/or time limits)
```

So, each standard user can submit up to 1000 jobs (please note sub-jobs in job arrays are counted towards the total number). It is not directly relevant to the number of jobs arrays. As long as the total number of jobs doesn't exceed the limit, it doesn't matter how many job arrays are submitted. For example, one can submit just one job array with 1000 jobs, or 10 job arrays with 100 jobs each, or 1000 ordinary jobs without any job array. The purpose for such limitation is to reduce the number of short-time jobs. If users submit a huge number of small jobs, the database of Slurm for recording all kinds of job information could burst very quickly. It is recommended to group some small short-time jobs together (for example, we could group 1000 one-minute jobs into 10 100-minute jobs). For some special tasks, we could increase the maximum number of submitted jobs to 2000-3000 during a given time period as per your request. Thus, you could submit two job arrays (with 1000 jobs each) and some ordinary jobs.

Slurm Example Scripts and Programs

In order to illustrate how sequential and parallel programs can be run under the Slurm scheduler, example `Slurm job` scripts, test programs and a Makefile can be found in `/opt/apps/Slurm_Examples`

Most of the examples are based on a Fortran 90 demonstration program for pi that comes with the ancient MPICH distribution. There are several sample codes to compile and then run via a batch script including `hello`, `pi3f90`, `cpi` and `pi_argument`.

`hello` is a basic "Hello World" parallel program. It shows that all of the processes requested are active and responding.

`pi3f90` (and `cpi`) takes its input from the file `infile` redirected as standard input.

`pi_argument` takes its input from the file `infile` whose name is passed as an argument. This file `infile` can be edited by the user if they so wish. It must terminate with a 0 or negative number.

It is probably simplest to copy this entire subdirectory to a subdirectory where you have write access. Full details (including some system dependent ones) are provided in the accompanying README file.

Each executable can be obtained by entering the command

`make` *executable*

where *executable* is the name of the desired code (please load a particular MPI module firstly, e.g. `module load mpi/openmpi/1.10.7/gcc-4.8.5`). You could generate all the executables only once by the command "`make all`".

I) **Executing a sequential job interactively on the login nodes**

The code `sequential_pi` can be formed and then run with the commands:

```
make sequential_pi
./sequential_pi
```

This takes input from the keyboard.

NOTE: *Running jobs on the login nodes is only acceptable for short test or debugging runs!*

II) **Executing a job interactively via Slurm scheduler**

You can start a new interactive job on the cluster by using the srun command; the scheduler will search for an available compute node, and provide you with an interactive login shell on the node if one is available.

```
[siteadmin@login1[barkla] test]$ srun --pty /bin/bash
[siteadmin@node081[barkla] test]$ . /sequential_pi
Enter the number of intervals: (0 quits)
20
      20  pi is approximately: 3.1418009868930943  Error is:
0.0002083333033012
0
[siteadmin@node081[barkla] test]$ exit
exit
[siteadmin@login1[barkla] test]$
```

In the above example, the srun command is used together with two options: --pty and /bin/bash. The --pty option executes the task in pseudo terminal mode, allowing the session to act like a standard terminal session. The /bin/bash option is the command that you wish to run - here the default Linux shell, BASH.

The Slurm scheduler does not set up your session to allow you to run graphical applications inside an interactive session.

If the job-scheduler could not satisfy the resource you've requested for your interactive job (e.g. all your available compute nodes are busy running other jobs), it will report back after a few seconds with an error:

```
srun: job 20 queued and waiting for resources
```

III) **Executing a sequential job by submitting it to Slurm scheduler**

The script file `sbatch_sequential_pi.sh` can be used to submit a job with `sequential_pi` to the batch queue, with input redirected from the file infile.

The necessary commands to perform this are:

```
make sequential_pi
sbatch sbatch_sequential_pi.sh
```

Two files will be generated after the job has run:

- `sequential_errors` — standard error output
- `sequential_output` — standard output (what you would see on the screen with an interactive job)

IV) **Submitting a parallel job using sbatch**

The syntax involved in submitting a parallel job via sbatch can easily be forgotten, so some examples that can provide a template for these jobs are provided.

- `sbatch_openmpi.sh` provides a basic script to run a pi program on several cores with input taken from a user provided file. Both standard output and standard error are directed to a particular file.
- `sbatch_pi_argument.sh` shows how the standard output and error files can be redirected to files of the user's choosing.
- `sbatch_NeedsPE_pi.sh` is similar to `qsub_pi_argument.sh` except that the number of nodes required in the job is specified on the command line.

NOTE: The unit for parallel environments on Barkla is the total number of cores.

V) **Submitting several jobs as one array job**

The Slurm scheduler provides *array jobs* which can be useful for parametric sweeps, or generally repeated similar runs, e.g. which can be written as a template with the data indexed in some way.

The script file used in this example (i.e. `sbatch_array_pi.sh`) contains:

```
#!/bin/bash -l
#SBATCH -D ./
#SBATCH --export=ALL
#SBATCH -t 00:30:00
# Define job array
#SBATCH -a 1-4

./sequential_pi < inpi$SLURM_ARRAY_TASK_ID > out_pi$SLURM_ARRAY_TASK_ID
```

It runs four instances (defined by `-t 1-4`) of the `sequential_pi` program, each with different input and output files. They may run at the same time, if enough cores are free, or be queued as appropriate, and the array job itself doesn't finish until each of its sub-tasks does. The `SLURM_ARRAY_TASK_ID` environment variable contains the sub-job index of the array job, ranging from 1–4 in this case. The tasks can either be serial, as in this case, or parallel, if multiple cores are specified via the *-n value*.

VI) **A detailed example**

The script `sbatch_pi_argument.sh` submits a 40 core job to the system. The error and output files are redirected to files specified by the user. Notice that the executable `pi_argument` (source file `pi_argument.f90`) accepts an argument that specifies the input file to use.

This submit script is reproduced here:

```
#!/bin/bash -l

# Use the current working directory and current environment for this job.
#SBATCH -D ./
#SBATCH --export=ALL
# Define an output file - will contain error messages too
#SBATCH -o pi%j.out
# Define job name
#SBATCH -J pi
# Request 40 cores on 1 node
#SBATCH -p nodes -N 1 -n 40
# Insert your own username to get e-mail notifications
#SBATCH --mail-user=<username>@liverpool.ac.uk
#SBATCH --mail-type=ALL

# Load the necessary modules
module purge
module load mpi/openmpi/1.10.7/gcc-4.8.5

# Edit to match your own executable and arguments

EXEC="./pi_argument infile"

#
# Should not need to edit below this line
#
echo ========================================================
echo SLURM job: submitted  date = `date`
date_start=`date +%s`

echo ========================================================
echo Job output begins
echo -----------------
echo

hostname

# $SLURM_NTASKS is defined automatically as the number of processes in the
# parallel environment.

echo Running with $SLURM_NTASKS cores
mpirun -np $SLURM_NTASKS $EXEC

echo
echo ---------------
echo Job output ends
date_end=`date +%s`
seconds=$((date_end-date_start))
minutes=$((seconds/60))
seconds=$((seconds-60*minutes))
hours=$((minutes/60))
minutes=$((minutes-60*hours))
echo =========================================================
echo SLURM job: finished   date = `date`
echo Total run time : $hours Hours $minutes Minutes $seconds Seconds
echo =========================================================
```

Some things to note

1. A parallel environment is normally specified via the -n c command where c is the number of cores to be used,
2. The option
   `#SBATCH -D ./`
   specifies that the current working directory should be used as the location for executables and file, which is the default setting.
3. The option
   `#SBATCH --export=ALL`
   exports the current environment variables to the sbatch environment, which is the default setting.
4. The option
   `#SBATCH -o pi%j.out`
   specifies that the output and error files are joined with name pi%j.out, where the "%j" is replaced with the job allocation number. By default both standard output and standard error are directed to a file of the name "slurm-%j.out".
5. Two common mail options are:

   `#SBATCH --mail-user=<user>`
   Specify user to receive email notification of state changes as defined by --mail-type. The default value is the submitting user.
   `#SBATCH --mail-type=<type>`
   Notify user by email when certain event types occur. Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL etc. Multiple type values may be specified in a comma separated list.

VII) **An example with more options and environmental variables**

The script `sbatch_hello_openmpi_full.sh` includes most of the frequently used options and environmental variables. Users could modify this script slightly for their own cases.

Refinements on access to compute nodes for parallel jobs

With the simple mpi parallel environment, the scheduler finds the first block of cores that is sufficiently large and that meets secondary requirements, such as memory per core. This may mean a job with 40 or fewer cores runs on more than one node.

There are several common options to specify the particular parallel environment:

```
#SBATCH -N, --nodes=<nnodes> (or <minnodes-maxnodes>)
```
This requests *nnodes* number of nodes (or, minimum *minnodes* nodes and maximum *maxnodes* nodes) to be allocated to this job.

```
#SBATCH -n, --ntasks=<number>
```
This requests the *number* of CPU cores to be allocated to this job.

```
#SBATCH -c, --cpus-per-task=<ncpus>
```
This requests *ncpus* number of CPU cores per MPI process.

```
#SBATCH --ntasks-per-node=<ntasks>
```
Request that *ntasks* tasks to be invoked on each node. If used with the `--ntasks` option, the `--ntasks` option will take precedence and the `--ntasks-per-node` will be treated as a maximum count of tasks per node.

```
#SBATCH -p gpu,gpuc
#SBATCH --gres=gpu:1
```
This requests “gpu” and “gpuc” partitions and 1 GPU on the GPU node to be used.

```
#SBATCH --exclusive[=user|mcs]
```
The job allocation can not share nodes with other running jobs (or just other users with the "`=user`" option or with the "`=mcs`" option).

GPU resource on Barkla

Currently the GPU resource on Barkla has been properly configured, and some GPU examples with Slurm have been tested and provided.

1. There are 5 GPU nodes (i.e., gpu01 powered with 4 Nvidia Tesla P100 GPUs and gpu[02-05] powered with 4 Nvidia Tesla V100 GPUs) in “gpu” and “gpuc” partition, which can used with the Slurm job scheduler.
You could submit a GPU job via Slurm using sbatch, and it is crucial to include the following options in your Slurm job script:
```
# Request GPU partitions
#SBATCH -p gpu,gpuc
# Request the number of GPUs to be used (if more than 1 GPU is required,
change 1 into Ngpu, where Ngpu=2,3,4)
#SBATCH --gres=gpu:1
# Request the number of nodes
#SBATCH -N 1
# Request the number of CPU cores (There are 24 cores on the GPU node, so 6
cores for 1 GPU)
#SBATCH -n 6
```

Alternatively, you could run the GPU job interactively using srun via the command line. For example,

```
srun -p gpu -N 1 -n 6 --gres=gpu:1  --pty /bin/bash
```

Please don't request multiple partitions in the command line. If you want to use more than 1 GPU at a time, then you can change `--gres=gpu:1` into `--gres=gpu:Ngpu (Ngpu=2,3,4).`

2. There are two visualisation nodes (viz01 and viz02, or barkla6.liv.ac.uk and barkla7.liv.ac.uk) which are powered with two Nvidia Quadro P4000 GPUs each. Users can run, test, and debug lightweight GPU applications directly on these two visualisation nodes (i.e. without Slurm job scheduler).

3. There are four KNL Xeon Phi nodes in "phi" partition, which can used with the Slurm job scheduler. They don't have GPUs, but some code (e.g., TensorFlow) might get good performance on them.

Some Slurm job scripts/examples for TensorFlow and Theano with GPU are at /opt/apps/Slurm_Examples/GPU/ on Barkla. You could play these examples, and modify these scripts (e.g., sbatch_tensorflow_gpu.sh and sbatch_theano_gpu.sh) slightly for your own cases. We have tested these TensorFlow and Theano examples with GPUs, and they all work fine.

Compiling and linking codes for maximum performance

The Skylake processors on Barkla tend to produce good performance with minimal compiler optimisation because these processors have better memory bandwidth characteristics than their predecessors and good floating point performance. However, performance can often be improved further by taking full advantage of particular compiler optimisations and linking against tuned performance libraries. More information on getting good performance out of your code can be found in the directory `/opt/apps/Barkla_Docs/Barkla_Launch_20171206`.

Compilers

Barkla users have a choice of two different compiler platforms - the free and generally good gcc/gfortran family and the licensed Intel compiler family. Each of these has its advantages.

The first recommendation is normally to try the gcc/gfortran compilers. With the versions available on the cluster (version 4.8 by default, and other versions available by loading a module), good performance and fairly solid code is available. OpenMP version 3.1 is supported by these compilers. GCC version 6.3 and higher have support for the Skylake hardware families, which is a strong point in its favour.

Whilst the Intel compiler has been known to cause difficulties in the past, current versions are generally reliable and produce fast code, although its default optimisation may be too aggressive. For many applications, near optimal performance is only possible using a precompiled library. The Intel Math Kernel Library (MKL) provides a broad range of tuned support for libraries such as the BLAS, LAPACK, ScaLAPACK and fftw3.

Applications and libraries

In addition to those system applications available by default, most users applications are available by loading the appropriate module files. MPI-based applications (e.g. cp2k, dl_poly) and libraries (e.g. libnetcdf.so) become accessible from the default path once the relevant modules are loaded. A up-to-date list of installed applications can be obtained by command "`module avail`".

How to install Python packages locally?

Different Python packages may require totally different dependency packages. It is unrealistic to install various Python packages globally. Along with many installations of Python and Python-relevant packages (apps/python/2.7.8/gcc-5.5.0, apps/python3/3.8.5/gcc-5.5.0, libs/scipy/0.17.0/gcc-5.5.0+atlas-3.10.3+python-2.7.8+numpy-1.10.4, etc), we have provided various versions of Anaconda (which is a distribution of the Python and R programming languages):

```
  apps/anaconda/2.5.0/bin
  apps/anaconda2/2019.10
  apps/anaconda2/5.2.0
  apps/anaconda3/2019.03
  apps/anaconda3/2019.10-bioconda
  apps/anaconda3/2019.10-general
  apps/anaconda3/2019.10-pytorch
  apps/anaconda3/2020.02
  apps/anaconda3/2020.02-pytorch
  apps/anaconda3/2020.07
  apps/anaconda3/4.4.0
  apps/anaconda3/5.2.0
  apps/anaconda3/5.2.0-python37
  apps/anaconda3/5.2.0-revised
```

Most standard Python packages are included in these Anaconda installations. You might find it more effective to use one of them. There is a Slurm job script with Python in /opt/apps/Slurm_Examples/python/. If your required Python packages are still not included in these implementations, you could install your own Python packages in your user space without root privileges. As the installation process may be time-consuming and resource-intensive, please do it on one of the visualisation nodes rather than login nodes. There are several commonly used approaches to install Python packages locally.

1. You could install Python packages in your own conda virtual environment. A conda environment is a directory that contains a specific collection of conda packages that you have installed.

(1) First, you need to load one of Anaconda modules. For example,

```
module load apps/anaconda3/2020.02
```

(2) You could create your own virtual environment. For example,

```
conda create -n myenv pystan
```

For new versions of Anaconda, you may need to specify your own location for your virtual environment to avoid writing in the default directory which is not yours. For example,

```
conda create --prefix ~/.conda/envs/myenv pystan
```

Please note that you could specify environment directories (envs_dirs) in your conda configuration file ~/.condarc to avoid specifying environment location every time. For more details please see https://docs.conda.io/projects/conda/en/latest/user-guide/configuration/use-condarc.html#specify-env-directories.

(3) After a virtual environment is created, you could activate it. For example,

```
source activate myenv
```

For newer versions of anaconda:

```
conda activate myenv
```

(4) You could install any required packages in your current virtual environment. For example,

```
conda install -c conda-forge pymc3
```

(5) You could check the installed packages in current virtual environment:

```
conda list
```

(6) You could deactivate an active virtual environment:

```
source deactivate
```

For newer versions of conda:

```
conda deactivate
```

(7) After the installation, if you want to use installed packages in a virtual environment, you need to load the module and activate the virtual environment. For example,

```
module load apps/anaconda3/2020.02
source activate myenv
```

Please see the document at https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html for more details about how to use conda. Please be careful when running command "conda init" as this will write conda initialization settings in your ~/.bashrc, which often causes problems. Such conda initialization settings in your ~/.bashrc are not recommended as they attempt to always load a particular conda virtual environment, which prevents you from using other conda environments. We have seen many strange problems when setting conda initialization in ~/.bashrc.

2. You could install your own Anaconda from scratch without loading any system's Anaconda module, but it might take a while and need very large space. With your own Anaconda, you could install your Python packages in the base environment, or in your virtual environment similarly as above. Please note your home directory (/users/$USER) is under quota both for size and the number of files present. Conda installs involve lots of files, so you need to make certain that you are using your volatile area (not your home area /users/$USER) to hold these

files. For more details about how to install conda, please see the document at https://conda.io/projects/conda/en/latest/user-guide/install/index.html.

3. You could also install Python packages using pip together with --user flag, after loading one of Anaconda modules. This will ensure the installed packages are isolated to the current user. Please note that pip installation should be a last resort only if conda installation is not available. For example,

```
module load apps/anaconda3/2020.02
pip install pyepal --user
```

If your Python packages are installed successfully via "pip --user" installation, they could be in the directory (for Python 3.7):

```
~/.local/lib/python3.7/site-packages/
```

You will be able to use them if you have set up the following environment variables:

```
export PYTHONPATH=~/.local/lib/python3.7/site-packages:$PYTHONPATH
export PATH=~/.local/bin:$PATH
```

Please see the document at https://packaging.python.org/tutorials/installing-packages/#use-pip-for-installing for more details.

How to install R packages locally?

R normally comes with some standard and recommended packages. This is usually in a system location. Users could use the R packages after loading relevant R module. There is a Slurm job script with R in /opt/apps/Slurm_Examples/R/. In some scenarios, users may install their own R packages if some extra R packages are needed. As the installation process may be time-consuming and resource-intensive, please do it on one of the visualisation nodes rather than login nodes. The brief procedure is shown as follows:

(1) Load one of R modules, or a module which includes R (e.g., apps/anaconda3/2019.10-bioconda). For example,

```
module load apps/R/3.6.3/gcc-5.5.0+lapack-3.5.0+blas-3.6.0
```

(2) Set up environment variable R_LIBS_USER

Environment variable R_LIBS_USER is used to specify the location of R packages to be installed locally. Once an R module is loaded, R_LIBS_USER will be set as `$HOME/gridware/share/R/${version}` by default (e.g., `$HOME/gridware/share/R/3.6.3`), which is located in your top-level $HOME directory and subject to quota. We recommend to set R_LIBS_USER as a subdirectory in your top-level `volatile` directory. This can be done by (a)setting up R_LIBS_USER in your `~/.bashrc` file:

```
export R_LIBS_USER=~/volatile/R_libs/3.6.3
```

or (b) creating a user-controllable R environment file called `.Renviron` in your top-level $HOME directory and including the following line in `.Renviron`:

```
R_LIBS_USER=~/volatile/R_libs/3.6.3
```

(3) Install your own R packages in the interactive R session

Fire up an R session:

```
R
```

To install a package (for example, lattice), use this command inside R:

```
> install.packages("lattice", repos="http://cran.r-project.org")
```

If the directory `~/volatile/R_libs/3.6.3/` doesn't exist (this happens when you attempt to install a package for the first time), R will say it cannot install into the main area because it is not writable and then will ask if you want a personal library instead (say yes) and it will prompt to create the directory `~/volatile/R_libs/3.6.3/` as a personal library (say yes):

```
Warning in install.packages("lattice", repos = "http://cran.r-project.org") :
  'lib      =      "/opt/gridware/depots/e2b91392/el7/pkg/apps/R/3.6.3/gcc-
5.5.0+lapack-3.5.0+blas-3.6.0/lib64/R/library"' is not writable
Would you like to use a personal library instead? (yes/No/cancel) yes
Would you like to create a personal library
'~/volatile/R_libs/3.6.3/'
to install packages into? (yes/No/cancel) yes
```

Then follow the interactive instructions, and your R packages will be installed locally.

Acknowledgement of the HPC Service in Research Outputs

All users of the ARC HPC facilities hosted at Liverpool should use the following text to name and acknowledge their use of the facilities in published works (including but not limited to papers, conference proceedings, presentations, posters, blog and other social media posts):

This work was undertaken on Barkla, part of the High Performance Computing facilities at the University of Liverpool, UK.

Appendices

Table 1. Common User Commands for SGE and Slurm

User Commands	SGE	SLURM
Job submission	qsub [script_file]	sbatch [script_file]
Job deletion	qdel [job_id]	scancel [job_id]
Job status by job	qstat -u * [-j job_id]	squeue [job_id]
Job status by user	qstat [-u user_name]	squeue -u [user_name]
Job hold	qhold [job_id]	scontrol hold [job_id]

Job release	qrls [job_id]	scontrol release [job_id]
Queue list	qconf -sql	squeue
List nodes	qhost	sinfo -N OR scontrol show nodes
Cluster status	qhost -q	sinfo
Interactive login	qlogin	srun
GUI	qmon	sview

Table 2. Common Environmental Variables for SGE and Slurm

Environmental Variables	**SGE**	**SLURM**
Job ID	$JOB_ID	$SLURM_JOBID
Job name	$JOB_NAME	$SLURM_JOB_NAME
Number of processes/cores	$NSLOTS	$SLURM_NTASKS
Submit directory	$SGE_O_WORKDIR	$SLURM_SUBMIT_DIR
Submit host	$SGE_O_HOST	$SLURM_SUBMIT_HOST
Node list	$PE_HOSTFILE	$SLURM_JOB_NODELIST
Job Array Index	$SGE_TASK_ID	$SLURM_ARRAY_TASK_ID
Queue name	$QUEUE	$SLURM_JOB_PARTITION
Temporary directory	$TMPDIR	$TMPDIR

Table 3. Common Job Specifications for SGE and Slurm

Job Specification	**SGE**	**SLURM**
Script directive	#$	#SBATCH
Queue	-q [queue]	-p [partition]
Count of nodes	N/A	-N [min[-max]]
CPU count	-pe [PE] [count]	-n [count]
Wall clock limit	-l h_rt=[seconds]	-t [min] OR -t [days-hh:mm:ss]
Standard out file	-o [file_name]	-o [file_name]
Standard error file	-e [file_name]	e [file_name]
Combine STDOUT & STDERR files	-j yes	(use -o without -e)
Copy environment	-V	--export=[ALL|NONE|variables]
Event notification	-m abe	--mail-type=[events]
Send notification email	-M [address]	--mail-user=[address]
Job name	-N [name]	--job-name=[name]
Restart job	-r [yes|no]	--requeue OR --no-requeue (NOTE: configurable default)
Set working directory	-wd [directory]	--workdir=[dir_name]

Resource sharing	-l exclusive	--exclusive OR--shared
Memory size	-l mem_free=[memory][K|M|G]	--mem=[mem][M|G|T] OR --mem-per-cpu=[mem][M|G|T]
Charge to an account	-A [account]	--account=[account]
Tasks per node	(Fixed allocation_rule in PE)	--tasks-per-node=[count]
		--cpus-per-task=[count]
Job dependancy	-hold_jid [job_id | job_name]	--depend=[state:job_id]
Job project	-P [name]	--wckey=[name]
Job host preference	-q [queue]@[node] OR -q [queue]@@[hostgroup]	--nodelist=[nodes] AND/OR --exclude=[nodes]
Quality of service		--qos=[name]
Job arrays	-t [array_spec]	--array=[array_spec] (Slurm version 2.6+)
Generic Resources	-l [resource]=[value]	--gres=[resource_spec]
Licenses	-l [license]=[count]	--licenses=[license_spec]
Begin Time	-a [YYMMDDhhmm]	--begin=YYYY-MM-DD[THH:MM[:SS]]