DMOG docs

Release 1.0

Jose M Menendez

May 08, 2023

USER GUIDE

Welcome to the documentation page for DMOG, the new high-performance computing (HPC) facility at Heriot-Watt University. DMOG is a state-of-the-art cluster that aligns with the University Strategy 2025, which aims to support and enable world-leading research.

The cluster is named after Dame Maria Ogilvie Gordona pioneering geologist who began her scientific career at Heriot-Watt College in 1888. She was the first woman to receive a Doctor of Science from UCL and a PhD from Munich University. Maria combined academic research with married life and work for the Liberal Party and women's action groups. She won the Geological Society Lyell medal for research on plate tectonics and became a Dame of the British Empire for her work for women's rights.

This documentation page serves as a comprehensive guide to using the DMOG cluster, including information on accessing the system, submitting jobs, and utilizing the various software packages and tools available. We hope you find this resource helpful in your research endeavors.

Note: This project is under active development.

CHAPTER

ONE

CONTENTS

1.1 Introduction

Welcome to the DMOG High-Performance Computing (HPC) facility!

This guide is designed to help users access and utilize the system effectively. Whether you're a researcher, student, or staff member, the DMOG HPC facility can provide significant computing resources to support your work. This guide covers essential topics for using the facility, including system access, data storage, job submission, program compilation, and environment management. By following these guidelines, you can make the most of the DMOG HPC facility and achieve optimal performance for your computing needs.

1.2 General Use Policy

The DOMG HPC cluster is a shared resource that has been set up to support research. It is a powerful and valuable resource that can be used to undertake computationally intensive tasks that would otherwise not be possible. It is important to remember that this resource is shared by all users and must, therefore, be used responsibly. Each user has a responsibility to ensure that their use of the resource does not negatively impact other users, and that they do not misuse the resource in any way.

The facility is to be used only for research purposes. This means that it should not be used for any other purposes, such as commercial or personal activities. Any misuse of the facility will not be tolerated, and users who violate the usage policy will be subject to disciplinary action, which may include suspension or termination of their access to the facility.

As a user of DMOG, you will be associated with a group that will have a designated line manager. The line manager is responsible for managing the group's accounts and ensuring that the group's usage of the HPC facility follows our guidelines. This includes deciding on the data stored in the user accounts, as well as managing access to the accounts. If a user leaves the group, their account will remain under the management of the line manager, who will ultimately decide what to do with the account. This ensures that access to our HPC facility is managed appropriately and that users are held accountable for their usage.

It is essential that all users of the facility adhere to the guidelines outlined in the documentation. The facility staff is available to provide support and assistance to users, and they can be contacted if any issues or concerns arise. By working together and using the facility responsibly, we can ensure that the facility remains a valuable resource for the research community.

1.3 Access and Priorization

Access to the HPC cluster is granted to research staff and research students by default, and is free at the point of use. To gain access, users are required to fill out a TopDesk form providing basic information about their research and the expected resource requirements. The HPC team will review each application and grant access in a timely manner.

For those with specific and/or time-sensitive research requirements, a paid priority access service is available. This provides access to higher priority in the job queues. Please contact us for more information on this service.

1.4 Job Quotas

To ensure fair share of resources among the HPC user community, we have implemented job quotas. These quotas are subject to review and may be adjusted as required. The current job quotas are as follows:

1. maximum of 10 jobs running per user
2. maximum of 60 jobs submitted per user (allowing up to 50 jobs to be held in a queued/pending state)
3. Maximum of 128 cores
4. Maximum of 1006G RAM
5. Maximum of 2 GPUs per user (minimum of 1 GPU per job)
6. Maximum runtime 7 days

These quotas are designed to ensure that all users have a fair opportunity to access the cluster resources. Users who require additional resources are encouraged to contact the HPC team to discuss their requirements.

1.5 Connecting to DMOG

Before following the process below, this guide assumes you have an account on the cluster. If that is not the case, you can request one by contacting ISHelp@hw.ac.uk

Interaction with DMOG is done remotely via SSH (Shecure Shell). To connect to another machine using SSH you need to have a SSH client program installed on your machine. macOS, Linux, and newer versions of Windows come with a command-line (text-only) SSH client pre-installed. On older Windows versions there are various graphical SSH clients you can use like PuTTY or MobaXterm.

Using your favorite ssh-client, just type:

```
ssh <username>@dmog.hw.ac.uk
```

Note: the cluster is accessible within the University network. For remote access you need to log on to the University VPN first.

1.6 Data Management

There are several data storage types available to users. Each of which has different characteristics and policies, and it is suitable for different types of use. Please note that the cluster is intended to store data for as long as it is being processed and so none of the storage options are backed up.

The following table shows the different storage options that are currently available to users:

Storage	Use	Exported to nodes	Total Capacity	Notes
`/users/`	Home dir	Yes	12TB	Quota: 150GB, No backup
`/mnt/data/`	Local data	Yes	30TB	No backup
`/mnt/scratch/`	Compute	Yes	93TB	No backup
`/tmp/users/`	Compute	No	up to 1TB	No backup

1.7 File Transfer

The easiest way to transfer data from/to the cluster is to use one of the standard programs based on the SSH protocol such us `scp` or `rsync`.

1.7.1 The scp command

The `scp` command creates a copy of a file, or a directory (if the -r flag is called) on a remote machine.

To copy data to the cluster:

```
scp [options] /source/path/to/object <user>@domg.hw.ac.uk:/path/to/destination
```

To copy data from the cluster:

```
scp [options] <user>@domg.hw.ac.uk:/source/path/to/object /path/to/destination
```

For a complete list of options available: `man scp`

1.7.2 The rsync command

`rsync` uses the same underlying protocol as `scp`, but it employs a special delta transfer algorithm. It compares if there is any differences in the files and only transfer those differences. Also, while copying if the connection drops, it can pick up the transfer where it was left off.

The syntax to copy files to/from the cluster:

```
rsync [options] /source/path/to/object <user>@domg.hw.ac.uk:/path/to/destination
rsync [options] <user>@domg.hw.ac.uk: /source/path/to/object /path/to/destination
```

For a list of options: `man rsync`

1.8 A word on OpenFlightHPC

We have installed `OpenFlightHPC` on our cluster to provide a robust and scalable HPC management solution for our users. With `OpenFlightHPC`, users have access to a suite of tools that can help them manage their jobs and resources more effectively, improving the overall efficiency and productivity of the cluster. Additionally, the modular nature of `OpenFlightHPC` allows us to easily add new functionality and adapt the system to meet evolving needs.

1.8.1 HPC User environments (flight env)

User environments (i.e flight environments) are software environments that have been pre-configured to support specific applications or workflows on an HPC cluster. They typically include a specific combination of software libraries, compilers, and tools that are required to run the application or workflow effectively.

On DMOG, there are several flight environments available to users, including `Gridware`, `Conda`, and `Singularity`. `Gridware` is a package manager that provides access to a range of pre-built software packages that have been compiled for use on the cluster, including the `Modules` package that allows you to load and manage software modules on the cluster. `Conda` is an open-source package management system and environment management system that makes it easy to install and manage multiple software packages on the cluster. `Singularity` is a containerization tool that allows you to create and run containers that encapsulate specific software environments and dependencies.

By providing these flight environments on the cluster, we aim to make it easier for users to access the software and tools they need to complete their work. Each flight environment has its own unique features and benefits. You can easily explore and manage the flight environments available on the cluster by using these commands:

- `flight env avail`: This command will display a list of all the available flight environments on the cluster. It will show you the name of each environment, as well as a brief description of its purpose and contents.
- `flight env activate <env_name>`: This command is used to activate a specific flight environment on the cluster. Replace <env_name> with the name of the environment you want to activate. Once activated, any commands you run will be executed within the selected environment.
- `flight env info`: This command will display detailed information about the currently active flight environment, including the environment's name, description, and location on the cluster.
- `flight env deactivate`: This command is used to deactivate the currently active flight environment. Once deactivated, any commands you run will be executed outside of the selected environment.

1.8.2 Remote desktop (flight desktop)

When working on interactive applications on an HPC cluster, it can often be difficult to test and fine-tune your workflows, especially when you're working remotely. That's where Flight desktop comes in. By providing a desktop environment that you can access remotely, Flight desktop gives you the flexibility to work on your applications from your desktop or laptop computer as if they were running locally.

Now, let's look at the pre-requisites for Windows, Mac, and Linux:

- Windows:
- VNC viewer: You will need to install a VNC viewer, such as TightVNC or RealVNC, to connect to your Flight Desktop session.
- SSH client: You will need to install an SSH client, such as PuTTY, to connect to your HPC cluster account and launch Flight Desktop.
- **Mac:**
  - VNC viewer: You can use the built-in Screen Sharing app on Mac to connect to your Flight Desktop session, or you can install a third-party VNC viewer such as RealVNC.

  - SSH client: The built-in Terminal app on Mac can be used as an SSH client to connect to your HPC cluster account and launch Flight Desktop.

- **Linux:**
  - VNC viewer: You can install a VNC viewer, such as Remmina or TigerVNC, to connect to your Flight Desktop session.
  - SSH client: Most Linux distributions come with an SSH client pre-installed, so you should be able to use your system's built-in SSH client to connect to your HPC cluster account and launch Flight Desktop.

To launch a flight desktop environment, follow these steps:

1. Log in to your HPC cluster account.
2. Open a terminal window and type `flight desktop start <desktop-name>` to launch `Flight Desktop`. If you're unsure what name to use, you can find the list of available desktop sessions by typing `flight desktop avail`.
3. Wait for the desktop environment to start up. Be patient, this may take a few minutes.
4. Once the desktop environment has started, you can connect to it using a `VNC viewer`. The specific details for connecting to Flight Desktop will be shown on the screen once the desktop session is created and can be retrieved by using this command: `flight desktop show <sessionid>`, where sessionid is the ID for the specific session.
5. Note that the desktop sessions are created on the login node, and they must be terminated manually by the user when they are no longer needed. To do so, you can use the `flight desktop kill <sessionid>` command.

For further information on using Flight Desktop, you can consult the online manual available through `flight help desktop`.

1.8.3 Disabling OpenFlightHPC

`OpenFlightHPC` is a powerful tool for HPC users, but there may be situations where users want to disable it. Some reasons for disabling `OpenFlightHPC` could include avoiding potential conflicts with other software installed on the cluster, or simply not needing any of the features provided by `OpenFlightHPC`.

To disable `OpenFlightHPC`, users can issue the `flight stop` command, which will stop any currently running Flight services. Additionally, users can use the `flight set always off` command to disable Flight services from starting up automatically in the future. It's important to note that these commands only affect the user who issues them and not the entire cluster.

1.9 Job Management

DMOG uses the `Slurm` job scheduler to manage resources and ensure fair access for all users. To submit jobs to the scheduler, users write a submission script specifying the required resources (such as the number of CPUs, amount of memory, etc.) and input/output files. After submitting a job, `Slurm` will try to allocate the requested resources and start the job. If the requested resources are not immediately available, the job will be queued until enough resources become available.

In addition to batch mode jobs, `Slurm` also supports interactive jobs which can be useful for testing code or troubleshooting problems. To start an interactive session, users can request an allocation of resources and then start a shell session on a compute node.

`Slurm` is not only used to submit and monitor jobs, but also to describe the resources required for the jobs. To ensure that your job runs efficiently, it is important to specify the correct resources in your submission script.

1.9.1 Basic Slurm commands

- `sinfo`: This command is used to display information about the nodes available on the cluster, such as their state (idle, down, etc.), partition, number of CPUs, and amount of available memory. It can help users choose appropriate resources for their jobs.
- `squeue`: This command is used to display the current queue of jobs on the cluster, including their status, user, and estimated start time. It can help users track the progress of their jobs and estimate when they will start running.
- `sbatch`: This command is used to submit a batch job to the Slurm scheduler. Users specify the required resources and the commands to run in a script, and then submit the script using the sbatch command. The job is then scheduled by the Slurm scheduler and run on the requested resources.
- `scancel`: This command is used to cancel a running or pending job on the cluster. Users can specify the job ID or job name to be cancelled.
- `scontrol`: This command is used to view and modify Slurm configuration and status information. It can be used to monitor the status of Slurm components.
- `squeue`: use this command to get a high-level overview of all active (running and pending) jobs in the cluster.

Further information about these commands is available in the online manual: `man <command>`

1.9.2 Job queue states and reasons

The `squeue` command allows users to view information about the state of a job. The default output format of the command is as follows:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
JOBID		Row 1, column 3	pp	00	jlkj	jhkh	hjhkj

Where

JOBID	Job or step ID. For array jobs, the job ID format will be of the form <job_id>_<index>
PARTITION	Partition of the job/step
NAME	Name of the job/step
USER	Owner of the job/step
ST	State of the job/step. See below for a description of the most common states
TIME	Time used by the job/step. Format is days-hours:minutes:seconds (days,hours only printed as needed)
NODES	Number of nodes allocated to the job or the minimum number of nodes required by a pending job
NODELIST(REASON)	For pending jobs: Reason why pending. For failed jobs: Reason why failed. For all other job states: List of allocated nodes. See below for a list of the most common reason codes.

During its lifetime, a job passes through several states. The most common states are PENDING, RUNNING, SUSPENDED, COMPLETING, and COMPLETED. Some other states are shown in the table below:

PD	Pending. Job is waiting for resource allocation
R	Running. Job has an allocation and is running
S	Suspended. Execution has been suspended and resources have been released for other jobs
CA	Cancelled. Job was explicitly cancelled by the user or the system administrator
CG	Completing. Job is in the process of completing. Some processes on some nodes may still be active
CD	Completed. Job has terminated all processes on all nodes with an exit code of zero
F	Failed. Job has terminated with non-zero exit code or other failure condition

If the job has failed or it is pending, a reason for its current state is given in the last column of the `squeue` output. Some of the most common reasons are:

(Resources)	The job is waiting for resources to become available so that the jobs resource request can be fulfilled
(Priority)	The job is not allowed to run because at least one higher prioritized job is waiting for resources
(Dependency)	The job is waiting for another job to finish first (–dependency=… option)
(TimeLimit)	The job exhausted its time limit.
(ReqNodeNotAvail)	Some node required by the job is currently not available. The node may currently be in use, reserved for another job, in an advanced reservation, DOWN, DRAINED, or not responding
JobLaunchFailure	The job could not be launched. This may be due to a file system problem, invalid program name, etc.

For a complete list of job states codes and reasons, see Job State Codes and Job Reasons

1.10 Job Submission

1.10.1 Batch Jobs

To submit a batch job to the cluster, you need a submission script that contains two parts. The first part, located at the top of the script, describes the resources that the job requires using `#SBATCH` directives. These directives are treated as `Slurm` options and specify the necessary resources for the job. The second part of the submission script contains the job-specific shell commands.

`Slurm` options can also be added to the sbatch command directly as command line arguments, which will override the ones embedded in the job script.

Here are some examples of job submission scripts.

Serial job

```
#!/bin/bash -l

################# Part-1 Slurm directives ####################
## Working dir
#SBATCH -D /users/username
## Environment variables
#SBATCH --export=ALL
## Output and Error Files
```

(continues on next page)

(continued from previous page)

```
#SBATCH -o job-%j.output
#SBATCH -e job-%j.error
## Job name
#SBATCH -J serial-test
## Run time: "hours:minutes:seconds", "days-hours"
#SBATCH --time=00:05:00
## Memory limit (in megabytes)
#SBATCH --mem=1024
## Specify partition
#SBATCH -p nodes

################# Part-2 Shell script ####################
#================================
#  Activate Flight Environment
#--------------------------------
source "${flight_ROOT:-/opt/flight}"/etc/setup.sh

#============================
#  Create results directory
#----------------------------
RESULTS_DIR="$(pwd)/${SLURM_JOB_NAME}-outputs/${SLURM_JOB_ID}"
echo "Your results will be stored in: $RESULTS_DIR"
mkdir -p "$RESULTS_DIR"

#================================
#  Application launch commands
#--------------------------------
# Customize this section to suit your needs.

echo "Executing job commands, current working directory is $(pwd)"

# REPLACE THE FOLLOWING WITH YOUR APPLICATION COMMANDS

echo "Hello, dmog" > $RESULTS_DIR/test.output
echo "This is an example job. It ran on `hostname -s` (as `whoami`)." >> $RESULTS_DIR/
↪test.output
echo "Output file has been generated, please check $RESULTS_DIR/test.output"
```

Shared memory job

```
#!/bin/bash -l

################# Part-1 Slurm directives ####################
## Working dir
#SBATCH -D /users/username
## Environment variables
#SBATCH --export=ALL
## Output and Error Files
#SBATCH -o job-%j.output
#SBATCH -e job-%j.error
```

(continues on next page)

(continued from previous page)

```
## Job name
#SBATCH -J shared-mem-test
## Run time: "hours:minutes:seconds", "days-hours"
#SBATCH --time=00:05:00
## Memory limit (in megabytes). Total --mem or amount per cpu --mem-per-cpu
#SBATCH --mem-per-cpu=1024
## Processing slots
#SBATCH --nodes=1
#SBATCH --ntasks=32
## Specify partition
#SBATCH -p nodes

################## Part-2 Shell script ####################
#================================
#  Activate Flight Environment
#--------------------------------
source "${flight_ROOT:-/opt/flight}"/etc/setup.sh

#============================
#  Create results directory
#----------------------------
RESULTS_DIR="$(pwd)/${SLURM_JOB_NAME}-outputs/${SLURM_JOB_ID}"
echo "Your results will be stored in: $RESULTS_DIR"
mkdir -p "$RESULTS_DIR"

#================================
#  Application launch commands
#--------------------------------
# Customize this section to suit your needs.

echo "Executing job commands, current working directory is $(pwd)"

# REPLACE THE FOLLOWING WITH YOUR APPLICATION COMMANDS

echo "Hello, dmog" > $RESULTS_DIR/test.output
echo "This is an example job. It ran on `hostname -s` (as `whoami`)." >> $RESULTS_DIR/
↪test.output
echo "Output file has been generated, please check $RESULTS_DIR/test.output"
```

Parallel (mpi) job

```
#!/bin/bash -l

################## Part-1 Slurm directives ####################
## Working dir
#SBATCH -D /users/username
## Environment variables
#SBATCH --export=ALL
## Output and Error Files
#SBATCH -o job-%j.output
```

(continues on next page)

(continued from previous page)

```
#SBATCH -e job-%j.error
## Job name
#SBATCH -J parallel-mpi-test
## Run time: "hours:minutes:seconds", "days-hours"
#SBATCH --time=00:05:00
## Memory limit (in megabytes). Total --mem or amount per cpu --mem-per-cpu
#SBATCH --mem-per-cpu=1024
## Processing slots
#SBATCH --nodes=2
#SBATCH --ntasks=32
## Specify partition
#SBATCH -p nodes

################# Part-2 Shell script ####################
#================================
#  Activate Flight Environment
#--------------------------------
source "${flight_ROOT:-/opt/flight}"/etc/setup.sh

#================================
#  Activate Package Ecosystem
#--------------------------------
# e.g.:
# Load the OpenMPI module for access to `mpirun` command
flight env activate gridware
module load mpi/openmpi

if ! command -v mpirun &>/dev/null; then
    echo "No mpirun command found, ensure that a version of MPI is installed and␣
↪available in PATH" >&2
    exit 1
fi

#============================
#  Create results directory
#----------------------------
RESULTS_DIR="$(pwd)/${SLURM_JOB_NAME}-outputs/${SLURM_JOB_ID}"
echo "Your results will be stored in: $RESULTS_DIR"
mkdir -p "$RESULTS_DIR"

#================================
#  Application launch commands
#--------------------------------
# Customize this section to suit your needs.

echo "Executing job commands, current working directory is $(pwd)"

# REPLACE THE FOLLOWING WITH YOUR APPLICATION COMMANDS

echo "Hello, dmog" > $RESULTS_DIR/test.output
echo "This is an example job. It was allocated $SLURM_NTASKS slot(s) across $SLURM_JOB_
↪NUM_NODES node(s). The master process ran on `hostname -s` (as `whoami`)." >>
↪$RESULTS_DIR/test.output
```

(continues on next page)

(continued from previous page)

```
mpirun -np $SLURM_NTASKS \
    /bin/bash -c \
    'echo "This process was executed on `hostname -s` with rank $OMPI_COMM_WORLD_RANK."'␣
↪\
    >> $RESULTS_DIR/test.output

echo "Output file has been generated, please check $RESULTS_DIR/test.output"
```

GPU job

```
#!/bin/bash -l

################# Part-1 Slurm directives ####################
## Working dir
#SBATCH -D /users/username
## Environment variables
#SBATCH --export=ALL
## Output and Error Files
#SBATCH -o job-%j.output
#SBATCH -e job-%j.error
## Job name
#SBATCH -J gpu-test
## Run time: "hours:minutes:seconds", "days-hours"
#SBATCH --time=00:05:00
## Memory limit (in megabytes). Total --mem or amount per cpu --mem-per-cpu
#SBATCH --mem-per-cpu=1024
## GPU requirements
#SBATCH --gres gpu:1
## Specify partition
#SBATCH -p gpu

################# Part-2 Shell script ####################
#================================
#  Activate Flight Environment
#--------------------------------
source "${flight_ROOT:-/opt/flight}"/etc/setup.sh

#================================
#  Activate Package Ecosystem
#--------------------------------
# e.g.:
# Load the OpenMPI module for access to `mpirun` command
flight env activate gridware
module load mpi/openmpi

if ! command -v mpirun &>/dev/null; then
    echo "No mpirun command found, ensure that a version of MPI is installed and␣
↪available in PATH" >&2
    exit 1
fi
```

(continues on next page)

(continued from previous page)

```
#============================
#  Create results directory
#----------------------------
RESULTS_DIR="$(pwd)/${SLURM_JOB_NAME}-outputs/${SLURM_JOB_ID}"
echo "Your results will be stored in: $RESULTS_DIR"
mkdir -p "$RESULTS_DIR"

#==============================
#  Application launch commands
#------------------------------
# Customize this section to suit your needs.

echo "Executing job commands, current working directory is $(pwd)"

# REPLACE THE FOLLOWING WITH YOUR APPLICATION COMMANDS

echo "Hello, dmog" > $RESULTS_DIR/test.output
echo "This is an example job. It ran on `hostname -s` (as `whoami`)." >> $RESULTS_DIR/
↪test.output
echo "I was allocated the following GPU devices: $CUDA_VISIBLE_DEVICES" >> $RESULTS_DIR/
↪test.output
echo "Output file has been generated, please check $RESULTS_DIR/test.output"
```

1.11 Interactive Jobs

When debugging or developing code, interactive testing is often necessary. However, running interactive jobs directly on the login node can cause overloading. It is recommended to run interactive jobs on the compute nodes instead. This allows you to debug your code in the same environment that it will run in.

Resource allocation for interactive jobs is done through the command line.

To start an interactive session on CPU node:

```
srun --nodes=1 --ntasks-per-node=32 --mem=1024 --time=00:05:00 --partition=nodes --pty /
↪usr/bin/bash
```

On a GPU enabled node, the command is very similar:

```
srun --nodes=1 --ntasks-per-node=32 --mem=1024 --time=00:05:00 --partition=gpu --gres:1 -
↪-pty /usr/bin/bash
```

1.12 Making dynamic jobs scripts

`Slurm` provides several environment variables at runtime that can be used to create more dynamic submission scripts. These variables can be used to specify the job name, set the number of nodes or tasks, and much more. Here are some of the main environment variables that Slurm creates at runtime:

- `SLURM_JOB_NAME`: The name of the job
- `SLURM_JOB_ID`: The job ID number
- `SLURM_JOB_CPUS_PER_NODE`: The number of CPUs per node
- `SLURM_JOB_NODELIST`: The list of nodes allocated to the job
- `SLURM_ARRAY_TASK_ID`: The index of the job within an array job

In addition, `Slurm` also supports using format characters in submission scripts to define directives. These format characters can be used to dynamically specify options such as the output file name or the number of nodes to use. Here are some of the most common format characters:

- `%j`: Job ID
- `%N`: Node name
- `%u`: User name
- `%a`: Array job ID
- `%A`: Array job ID range

By using these environment variables and format characters, you can create more dynamic and flexible submission scripts that can adapt to different job requirements. For example, you can use the `%j` format character to dynamically specify the output file name based on the job ID, or use the `SLURM_JOB_CPUS_PER_NODE` variable to dynamically set the number of CPUs to use.

1.13 Software Management

1.13.1 System software

HPC clusters typically have many users with varying software requirements and installing software for each user can be time-consuming and may lead to conflicts between different versions of the same software.

`Modules` provide a way to manage software installations and make different versions of the same software available to users on demand. With modules, users can load and unload different versions of software into their environment without interfering with other users or the system's default settings. This makes it easy for users to work with different versions of the same software or to use different software packages for different projects or workflows.

Note: Please note that the easiest way to use modules is by activating the gridware user HPC environment (`flight env activate gridware`).

Some of the most useful commands for interacting with modules include:

- `module avail`: This command lists all the available modules on the system, showing the name, version, and any dependencies.
- `module load`: This command loads a specific module into the user's environment, making its executable and library files available for use.

- `module unload`: This command unloads a previously loaded module, removing its executable and library files from the user's environment.
- `module list`: This command lists all the currently loaded modules, showing their name and version.
- `module show`: This command displays detailed information about a specific module, including its version, dependencies, and any available module options.

1.13.2 Installing new software

In general, installing software on DMOG may require root access, as it often involves making changes to the system's configuration and installing packages in system directories that are only accessible to the root user.

However, there are alternative ways to install and manage software that do not require root access. For example, users may be able to install software in their home directories or in a designated software directory that's writable by users.

If you're not sure whether you need root access to install software on your cluster, it's a good idea to contact the cluster's system administrators for guidance.

1.13.3 Requesting new software

If you require software that is not currently available on the cluster and could benefit the broader user community, please contact the cluster support team at ISHelp@hw.ac.uk. They can assist you in installing the software or evaluating its suitability for the cluster. This will also help other users who may require the same software. Please note that the support team will evaluate the software for security and compatibility issues before installing it on the cluster.

1.14 AMBER

AMBER (Assisted Model Building with Energy Refinement) is a widely used software package for molecular dynamics simulations of biomolecules. It allows users to simulate the motion and behaviour of molecules in a variety of conditions, such as in different solvents or under different temperatures.

On DMOG, users can access AMBER by activating the gridware flight environment using the command `flight env activate gridware`. Once activated, the AMBER package module can be loaded using the command `module load apps/amber/22/gcc-12.2.0`. This will make the AMBER tools and executables available for use on the cluster.

AMBER includes several executables that are commonly used for molecular dynamics simulations. Some of the most frequently used executables include:

- `sander`: The main molecular dynamics engine used in AMBER, which can perform energy minimization, equilibrium simulations, and production runs.
- `pmemd`: An optimized version of sander that can run on GPUs, which can significantly accelerate molecular dynamics simulations.
- `cpptraj`: A utility that can perform a variety of tasks related to analyzing and processing the output from AMBER simulations, such as calculating RMSDs, extracting specific frames from a trajectory, and generating input files for other programs.

1.14.1 Running AMBER jobs

An example job script for AMBER would be:

```
#!/bin/bash
#SBATCH --job-name=amber
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --cpus-per-task=1
#SBATCH --time=1:00:00
#SBATCH --mem=128G
#SBATCH --partition=nodes

# Load required modules
flight env activate gridware
module load apps/amber/22/gcc-12.2.0

# Change to the directory where input files are located
cd /path/to/input/files

# Run the Amber simulation
srun --mpi=pmi2 $AMBERHOME/bin/pmemd.MPI -O -i input.in -o output.out -p prmtop -c␣
↪inpcrd -r output.rst
```

In this script, the job is named "amber" and will run on a single node with 32 tasks (processes) per node, using 1 CPU per task. The requested memory is 128GB and the job has a maximum runtime of 1 hour. The "nodes" partition is specified.

The required modules for Amber and Gridware are loaded, and the script changes to the directory where the input files for the simulation are located. Finally, the Amber simulation is run using the *srun* command with the appropriate arguments.

1.15 MATLAB

MATLAB is a high-level programming language and interactive environment widely used for numerical computation, visualization, and programming. It provides an extensive collection of mathematical functions, algorithms, and toolboxes for various areas of engineering, science, and finance. MATLAB is designed to make numerical analysis and data processing easier, faster, and more accurate. It is widely used in academia, research, and industry for a range of applications, from data analysis and visualization to control systems design and image processing.

MATLAB is available for use under the university's licenses for teaching and academic research purposes. If you need to use MATLAB for other purposes, please contact ISHelp@hw.ac.uk for further guidance.

1.15.1 Running MATLAB jobs

To use MATLAB on the cluster, you need to load the MATLAB module first. You can do this by running the following commands in your terminal:

```
flight env activate gridware
module load apps/matlab/r2022b
```

To run MATLAB jobs on the cluster, you need to create a job script that contains the MATLAB commands that you want to run. Here is an example job script (`my_matlab_job.sh`):

```
#!/bin/bash
#SBATCH --job-name=my-matlab-job
#SBATCH --output=my-matlab-job.out
#SBATCH --error=my-matlab-job.err
#SBATCH --nodes=1
#SBATCH --cpus-per-task=4
#SBATCH --time=00:30:00

flight env activate gridware
module load apps/matlab/r2022b

matlab -nodesktop -nosplash -r "my_matlab_script"
```

This job script requests one node with four CPUs and a maximum run time of 30 minutes. It loads the MATLAB module and runs a MATLAB script called `my_matlab_script`. The output and error messages are written to `my-matlab-job.out` and `my-matlab-job.err` respectively.

Here is an example MATLAB script that calculates the sum of two numbers (`my_matlab_script.m`):

```
a = 5;
b = 10;
c = a + b;
disp(['The sum of a and b is ', num2str(c)]);
```

Save this script as `my_matlab_script.m` and put it in the same directory as your job script. To submit the job script to the cluster, run the following command in the terminal:

```
sbatch my_matlab_job.sh
```

This will submit the job script to the cluster, and you can monitor the job status using the `squeue` command.

Here is an example job script that requests a GPU node and uses the "gpu" queue (`my_matlab_gpu_job.sh`):

```
#!/bin/bash
#SBATCH --job-name=my-matlab-gpu-job
#SBATCH --output=my-matlab-gpu-job.out
#SBATCH --error=my-matlab-gpu-job.err
#SBATCH --nodes=1
#SBATCH --cpus-per-task=4
#SBATCH --gres=1
#SBATCH --partition=gpu
#SBATCH --time=00:30:00

flight env activate gridware
```

(continues on next page)

(continued from previous page)

```
module load apps/matlab/r2022b
module load apps/nvidia-cuda/11.2.2

matlab -nodesktop -nosplash -r "my_matlab_gpu_script"
```

This job script is like the previous one, but it requests a GPU node with one GPU and uses the "gpu" queue. It also loads the CUDA toolkit module to enable GPU acceleration in MATLAB.

Here is an example MATLAB script that uses the GPU to perform matrix multiplication (`my_matlab_gpu_script.m`):

```
a = gpuArray.rand(1000);
b = gpuArray.rand(1000);
c = a * b;
d = gather(c);
disp(d);
```

Save this script as `my_matlab_gpu_script.m` and put it in the same directory as your job script.

To submit the job script to the cluster, run the following command in the terminal:

```
sbatch my_matlab_gpu_job.sh
```

This will submit the job script to the "gpu" queue, and the cluster will allocate a GPU node for your job to run.

1.16 R

R is a programming language and software environment used for statistical computing and graphics. It provides a wide variety of statistical and graphical techniques, and is highly extensible. R is popular among researchers and data analysts in many fields, and is widely used in the academic and business communities.

1.16.1 Running R jobs

To run R on the HPC cluster, you need to load the appropriate software module:

```
flight env activate gridware
module load apps/R/4.2.3
```

To run R on the CPU queue, you need to create a job script that specifies the resources required for your job. Here is an example job script:

```
#!/bin/bash
#SBATCH --job-name=my_R_job
#SBATCH --output=output.log
#SBATCH --error=error.log
#SBATCH --partition=nodes
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8
#SBATCH --time=01:00:00

# Load the R module
```

(continues on next page)

(continued from previous page)

```
flight env activate gridware
module load R/4.2.3

# Run the R script
Rscript my_R_script.R
```

In this example, the job is submitted to the nodes partition, which is the default partition for CPU jobs. The job requests one node with 8 CPU cores (`--nodes=1` and `--cpus-per-task=8`). The job also requests a runtime of 1 hour (`--time=01:00:00`). The `my_R_script.R` file is the R script that you want to run.

To run R on the GPU queue, you need to use a different partition and request the appropriate resources.

```
#!/bin/bash
#SBATCH --job-name=my_R_job
#SBATCH --output=output.log
#SBATCH --error=error.log
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=4
#SBATCH --time=01:00:00

# Load the R module
flight env activate gridware
module load R/4.2.3

# Run the R script
Rscript my_R_script.R
```

In this example, the job is submitted to the gpu partition, which is the partition for GPU jobs. The job requests one node with 1 GPU (`--gres=gpu:1`), 4 CPU cores (`--cpus-per-task=4`), and a runtime of 1 hour (`--time=01:00:00`). The `my_R_script.R` file is the R script that you want to run.

Here are a couple of simple R scripts that you can use to test your setup:

Example 1: Hello World

```
# Print "Hello, world!" to the console
cat("Hello, world!\n")
```

Example 2: Basic Data Analysis

```
# Load the built-in iris dataset
data(iris)

# Print the dimensions of the dataset
cat("Number of rows: ", nrow(iris), "\n")
cat("Number of columns: ", ncol(iris), "\n")

# Compute the mean of the Sepal.Length variable
mean_sepal_length <- mean(iris$Sepal.Length)
cat("Mean Sepal.Length: ", mean_sepal_length, "\n
```