Hopper User Guide *(version 2025.09.08)*

[Written/edited by Leah Isseroff Bendavid, based on and including excerpts from the PSB Bridgers User Guide (https://www.psc.edu/bridges/user-guide) and the Weizmann Institute Chemfarm User Guide (http://www.weizmann.ac.il/chemistry/chemfarm/home).]

Contact Information

Asprey CCAS director: Joe Tanski, jtanski@vassar.edu

Hopper System Administrator: Jesse Hayward, jhayward@vassar.edu

What is a computer cluster?

A computer cluster is a cluster of identical, commodity-grade computers networked into a small local area network with installed libraries and programs, allowing the processing to be shared among them. The result is a high-performance parallel computing cluster comprising inexpensive personal computer hardware.

There are two factors that convert a collection of compute nodes into a cluster. One is shared storage and the second is the workflow manager. Shared storage allows any compute node to access the same files and the same programs. Additionally, shared storage creates for the user an identical environment on any compute node in a cluster. The workload manager allocated specific parts of hardware or other resources to run a program. It ensures that every user will be allocated the requested resources without disturbing other users. The user's request for the resources and to run the program is identified by the workload manager as a "job". Since the cluster is a public resource and may have usage policies, all jobs are organized in queues of different priority and with different amount of resources. The workload manager sorts and runs jobs from different queues.

Connecting to Hopper

When you connect to Hopper, you are connecting to its login node or head node The login node is used for managing files, submitting batch jobs, and launching interactive sessions. *It is not intended for production/research computing*. Compute nodes are where your real work are done. From a login node, you can use the SLURM workload manager to connect to one or more compute nodes in an interactive session or submit a batch script to be run on one or more compute nodes. You *should not* log in to the compute nodes directly. Doing so would have a detrimental effect on the performance of those compute nodes. See the running jobs section of the user guide for information on production computing on Hopper.

You need to use an SSH client from your local machine to connect to Hopper. SSH is client-server software, which means that both the user's local computer and remote computer must have it installed. You must install SSH client software on your local machine. Free SSH clients for Macs, Windows Machines, and many versions of UNIX are available. Popular SSH clients (GUI) include PuTTY for Windows and Cyberduck for Macs. A command line version of SSH is installed on Macs by default; if you prefer that, you can use it in the Terminal application. Windows also offers a command line version of SSH via the command window.

Using your SSH client, connect to hostname **`jr.vassar.edu`** using port **`22.`** Enter your Hopper username and password when prompted. To connect to Hopper using a command line version of SSH (e.g., Terminal or Cygwin), you would use the command **`ssh username@jr.vassar.edu`** where "username" should be replaced by your own login name. Enter the password when prompted. You cannot log in to Hopper from off-campus unless you are connected to Vassar's VPN (Virtual Private Network). Consult with Vassar's CIS Department for help in accessing the VPN.

A UNIX shell is a command-line interpreter that provides a traditional user interface for UNIX and UNIX-like systems. Hopper runs a UNIX-like operating system (Linux), so you will be using a shell wherever you are logged into Hopper or in batch scripts. The default shell on Hopper is bash, a Bourne-type shell. Other shells are available for you if you prefer. For a practical introduction to using the UNIX operating system with a focus on Linux command line skills, consider reviewing the Practical Unix OpenClassroom or the CodeAcademy course "Learn the Command Line". A comprehensive introduction to UNIX shell usage and text editing is beyond the scope of this User guide, and the reader is directed to review other resources.

System Configuration

Hopper contains 8 computational nodes, of two different types. Each type of node has a different number of sockets (a multi-core package), CPUs cores per socket, and threads per core, resulting in a different total number of CPUs per node. Additionally, the types of nodes differ in their available Random Access Memory (RAM), or simply "memory", which is where the data is located when a program runs on a CPU or a collection of CPUs. The specifications for the computational nodes on Hopper are provided in Table 1 below.

Table 1. Specifications of the Computational Nodes on Hopper

Node type/brand	Node names	Sockets per node	CPU cores per socket	Threads per CPU core	Total CPUs per node	RAM per node
SuperMicro	node1, node2	4	12	1	48	64 GB
Dell EMC	node3, node5, node6, node7, node8	2	32	2	128	512 GB
Lambda (GPU)	lambda	2	12	2	48	128 GB

When a user submits a job, this is interpreted as a request for computational resources. One may request a specific number of CPUs/nodes/RAM in their job request. If the request for resources is physically impossible to fulfill, the job will never start. Note that even though one may request a number of CPUs for a job, the program running that job will only use all of the CPUs if it is a parallelized program (i.e., it uses Message Passing Interfaces - MPI, multithreading, or OpenMP). Also note that even with parallelized

programs, programs do not speed up linearly with increasing numbers of cores - processes are often slowed down by memory latency and communication overhead.

File Spaces

There are several distinct file spaces on Hopper, each serving a different function.

Home ($HOME)

This is your Hopper home directory. It is the usual location for your batch scripts, source code, and parameter files; it **should not be used a workspace for running jobs**. Its path is */home/username,* where *username* is your personal username. You can refer to your home directory with the environment variable `$HOME`.Your home directory is visible to all of the Hopper nodes. Your home directory has an 8 GB quota (students) or 16 GB (faculty) and is backed up regularly. `$HOME` is not intended to be a location for persistent storage of large amounts of data.

Work ($WORK)

This is your Hopper work directory, located at */work/username*, where *username* is your personal username. The work directory is for all I/O for running calculations and more persistent storage of data, although this directory is **not backed up** - you are responsible for backing up your data elsewhere. Users have a quota of 16 GB quota (students) or 1026 GB quote (faculty) of storage in the work directory.

To check your quota and your usage, run the **check-my-quota** script.

Work ($SCRATCH)

This is your fast but volatile workspace, located at /scratch/username, where *username* is your personal username. Scratch is designed to be fast, but with the possibility of data loss.

Transferring Files

scp

To use `scp` for file transfer, you must specify a source and destination of your transfer. The format for either source or destination is

```
username@machine-name:/path/filename
```

To transfer a file from a local machine to Hopper, one would type the following in a command line interface on the local machine (e.g., Terminal or Cygwin);

```
scp /path/to/myfile username@jr.vassar.edu:/path/to/file-destination
```

(Note that you can only do this if your local machine is on the Vassar network or connected to the Vassar VPN).

```
scp username@jr.vassar.edu:/path/to/myfile /path/to/file-destination
```

Enter your password when prompted.

Graphical User Clients

You can also use a graphical user client to transfer files from Hopper to a local machine (and vice versa). Some Windows options include WinSCP and FileZilla. CyberDuck can be used on Macs.

Programming Environment

C, C++, and Fortran

The GNU compilers for C, C++, and Fortran are available on Hopper, and are loaded automatically. The compilers are:

	C	**C++**	**Fortran**
GNU compiler	`gcc`	`g++`	`gfortran`

MPI Programming

Two types of MPI are supported on Hopper: MPICH and OpenMPI. There are two steps to compile an MPI program:

1. Load the correct module for the MPI type you want to use, using the command `module load modulename`, where the appropriate *modulename* is given in the table below.
   [See MODULE SOFTWARE for more information on modules.]
2. Issue the appropriate MPI wrapper command (provided in the table below) to compile your program.

The two MPI types may perform differently on different problems or in different programming environments. If you are having trouble with one type of MPI, please try using another type.

MPI Type	**Module Name**	**C wrapper command**	**C++ wrapper command**	**Fortran wrapper command**
OpenMPI MPICH	openmpi-3.1 mpich-3.3	mpicc	mpicxx	mpifort

Running Jobs

All production computing must be done on Hopper's compute nodes, **NOT** on Hopper's login nodes. The SLURM scheduler (Simple Linux Utility for Resource Management) manages and allocates all of Hopper's compute nodes. Several *partitions*, or job queues, have been set up in SLURM to allocate resources efficiently.

To run a job on Hopper, you need to decide *how* you want to run: interactively or in batch; and *where* to run- that is, which partitions you are allowed to use.

What are the different ways to run a job?

You can run jobs on Hopper in several ways:

Batch mode - where you first create a batch (or job) script which contains the commands to be run, then submit the job to be run as soon as resources are available

Interactive mode - where you type the commands to receive output back to your screen as the commands complete

Regardless of which way you choose to run your jobs, you will always need to choose a partition to run them in.

Which partitions can I use?

Different partitions control different types of Hopper's resources; they are configured by the type of node they control along with other job requirements like how many nodes or how much time or memory is needed. For more information, see Hopper Partitions.

Batch Jobs

To run a batch job, you must first create a batch (or job) script, and then submit the script using the `sbatch` command. A batch script is a file that consists of `SBATCH` directives, executable commands, and comments.

`SBATCH` directives specify your resource requests and other job options in your batch script. You can also specify resource requests and options on the `sbatch` command line. Any options on the command line take precedence over those given in the batch script. The `SBATCH` directives must start with "#SBATCH" as the first text on a line, with no leading spaces. Comments begin with a '#' character.

The first line of any batch script must indicate the shell to use for your batch job.

Sample Batch Scripts

A sample script for an MPI job is provided below.

Note that:

- The script uses the bash shell, indicated by the first line "`#!/bin/bash`". If you use a different shell some UNIX commands will be different.

- For *username, jobname, myprogram, and path-to-work-directory,* you must substitute your appropriate names, programs, and paths.
- Request the resources that are appropriate for your calculation. This includes reasonable numbers of nodes and cores, as well as the estimated run time. More than one node should only be requested in instances where there are not enough cores on one node alone. If your code is not parallelized, *do not* request more than that one core (there is no computational advantage to requesting more than one core).

```
#!/bin/bash
#SBATCH -J jobname                          # job name
#SBATCH -o jobname.out                      # output and error file name
#SBATCH –nodes=1                            # total number of nodes to run job on
#SBATCH –ntasks-per-node=48                 # number of CPUs per node to run job on
#SBATCH -p general                          # queue (partition) - general, emc, lambda
#SBATCH -t 01:30:00                         # run time (hh:mm:ss) - 1.5 hours
#SBATCH –mail-user=username@vassar.edu      # user Email address for errors, reports
#SBATCH –mail-type=begin                    # email me when the job starts
#SBATCH –mail-type=end                      # email me when the job finishes

#commands to execute the job appear below the header

module load mpich-3.3
cd /home/username/path-to-work-directory
srun –mpi=pmi2 -n $totalcpus /home/username/bin/myprogram
```

A sample script for a single core job might look like something like the example given below:

```
#!/bin/bash
#SBATCH -J jobname                          # job name
#SBATCH -o jobname.out                      # output and error file name
#SBATCH –nodes=1                            # total number of nodes to run job on
#SBATCH –ntasks-per-node=1                  # number of CPUs per node to run job on
#SBATCH -p general                          # queue (partition) - general, emc, lambda
#SBATCH -t 01:30:00                         # run time (hh:mm:ss) - 1.5 hours
#SBATCH –mail-user=username@vassar.edu      # user Email address for errors, reports
#SBATCH –mail-type=begin                    # email me when the job starts
#SBATCH –mail-type=end                      # email me when the job finishes

#commands to execute the job appear below the header

cd /home/username/path-to-work-directory
/home/username/bin/myprogram
```

To submit a batch job, use the `sbatch` command. The format is

```
sbatch -options batch-script
```

The options to `sbatch` can either be in the header of your batch script or on the `sbatch` command line. Options in the command line override those in the batch script.

Some basic options to the `sbatch` command

Option	Description	Default
`-p` *`partition`*	Partition requested	general
`-t` *`HH:MM:SS`*	Walltime requested in HH:MM:SS	infinite
-N *n*	Number of nodes requested.	1
`–mem=`*`n`*`GB` Note the "--" for this option	Memory in GB.	None
`–ntasks-per-node=`*`n`*	Request *n* cores be allocated per node.	1
`–mail-type=`*`type`* Note the "--" for this option	Send Email when job events occur, where *type* can be BEGIN, END, FAIL, or ALL	None
`–mail-user=`*`user`* Note the "--" for this option	User to send Email to as specified by –mail-type. Default is the user who submits the job	None
`-h`	Help, lists all the available command options	

Interactive Sessions

The `srun` command can be used to request nodes for interactive use. If you use the syntax:

```
srun -N 2 –ntasks-per-node=8 –pty bash
```

this requests 2 nodes (-N 2) with 8 tasks per node (--ntasks-per-node=8), and it launches a bash shell on the compute nodes. The option –pty is necessary to start a session that looks like a normal interactive session but is on one of the compute nodes.

Note: you can add any "normal" options to the `srun` line, like `-p` for partition or `-t` for runtime.

After you enter the `srun` command you will be put into the queue waiting for nodes to become available. When they do, you will get an interactive session on a compute node, which will start in the directory from which you launched the session. You can then run any programs/computations interactively. Note: the compute nodes may have a different environment or commands available in comparison with the head node. The environment you get on a compute node is determined by a combination of three things:

- The environment as set in your session from which you launched the `srun` command.
- Extra variables set by SLURM
- Settings from your .bashrc file

Hopper Partitions

There are three partitions available for use.

General

Jobs in the general partition run on node1, node2, and node3. This is the default partition. There are currently no queue limits for this partition, although nodes/cores cannot exceed those available in the partition.

EMC

Jobs in the emc partition run on node5, node6, node7, and node8. Usage of the emc partition is reserved/prioritized for the Bendavid Research Group. There are currently no queue limits for this partition, although nodes/cores requested cannot exceed those available in the partition.

Lambda

Jobs in the lambda partition are reserved for GPU-based projects. There is one node with multiple GPU cards installed, with adequate memory and processing capabilities to support GPU-directed workflows/research.

Node, partition, and job status information

sinfo

The `sinfo` command displays information about the state of Hopper's nodes. The nodes can have several states:

alloc	Allocated to a job
down	Down
drain	Not available for scheduling
idle	Free
resv	Reserved

See also: sinfo man page

squeue

The `squeue` command displays information about the jobs in the partitions. Some useful options are:

`-j` *`jobid`*	Displays the information for the specified jobid
`-u` *`username`*	Restricts information to jobs belonging to the specified username
`-p partition`	Restricts information to the specified partition
`-l`	(long) Displays information including; time requested. time used, number of requested nodes, the nodes on which a job is running, job state, and the reason why a job is waiting to run.

See also: squeue man page for a discussion of the codes for a job state, for why a job is waiting to run, and more options.

scancel

The `scancel` command is used to kill a job in a partition, whether it is running or still waiting to run. Specify the jobid for the job you want to kill. For example

```
scancel 12345
```

kills job # 12345

See also: scancel man page

sacct

The `sacct` command can be used to display detailed information about jobs. It is especially useful in investigating why one of your jobs failed. The general format of the command is:

```
sacct -X -j nnnnnn -S MMDDYY –format parameter,parameter2, …
```

For '*nnnnnn*' substitute the jobid of the job you are investigating. The date given for the -S option is the date at which `sacct` begins searching for information about your job. The commas between the parameters in the –format option cannot be followed by spaces. The –format option determines what information to display about a job. Useful parameters are:

JobID	
Partition	
Account	the account charged
ExitCode	useful in determining why a job failed
State	useful in determining why a job failed
Start, End, Elapsed	start, end, and elapsed time of the job
NodeList	list of nodes used in the job
NNodes	how many nodes the job was allocated
MaxRSS	how much the job was used
AllocCPUs	how many cores the job was allocated

See also: sacct man page

Monitoring Memory Usage

It can be useful to find the memory usage of your jobs. For example, you may want to find out if memory usage was a reason a job failed. You can determine a job's memory usage whether it is still running or

has finished. To determine if your job is still running, use the `squeue` command

```
squeue -j nnnnnn -O state
```

where *nnnnnn* is the jobid.

For running jobs: srun and top or sstat

You can use the `srun` and `top` commands to determine the amount of memory being used.

```
srun –jobid=nnnnnn top -b -n l | grep userid
```

For *nnnnnn* substitute the jobid of your job. For *'userid'* substitute your userid. The RES field in the output from `top` shows the actual amount of memory being used by a process. The top man page can be used to identify the fields in the output of the `top` command.

See the man pages for `srun` and `top` for more information.

You can also use the `sstat` command to determine the amount of memory being used in a running job

```
sstat -j nnnnnn.batch –format=JobID,MaxRss
```

where *nnnnnn* is your jobid.

See the man page for `sstat` for more information.

For jobs that are finished: sacct or job_info

If you are checking within a day or two after your job has finished you can issue the command

```
sacct -j nnnnn –format=JobID,MaxRss
```

If this command no longer shows a value for MaxRSS, use the `job_info` command

```
job_info nnnnn | grep max_rss
```

Substitute your jobid for *nnnnnn* in both of these commands.

See the man page for `sacct` for more information.

See also: documentation for SLURM, including man pages for all of the SLURM commands

---

Module Software

The **Module** package provides for the dynamic modification of a user's environment via *module files.* Module files manage necessary changes to the environment, such as adding to the default path or defining environment variables, so that you don't have to manage those definitions and paths manually.

Modules are also used to manage multiple versions of applications, tools, and libraries, and where name conflicts between multiple packages would cause problems. Type `module available` to see a

complete list of the modules available to the system.

Basic use

To see what modules are available for a software package, type

`module avail` *`package-name`*

To set up the environment for a software package, load that environment variable with the `module load` command.

`module load` *`package-name`*

Module commands

`module avail`	lists all the available modules
`module avail` *`foo`*	lists all the available modules for package *foo*
`module help` *`foo`*	displays help on module *foo*
`module display` *`foo`*	indicates what changes would be made to the environment by loading module *foo* without actually loading it
`module load` *`foo`*	loads module *foo*
`module list`	displays currently loaded modules
`module swap` *`foo1 foo2`*	switches loaded module *foo1* with module *foo2*
`module unload` *`foo`*	reverses all changes to the environment made by previously loading module *foo*