ORC NEW USER TUTORIAL

Sign In: https://forms.office.com/r/XHkbPLAus1

GEORGE MASON UNIVERSITY

ORC NEW USER TUTORIAL

OFFICE OF RESEARCH COMPUTING

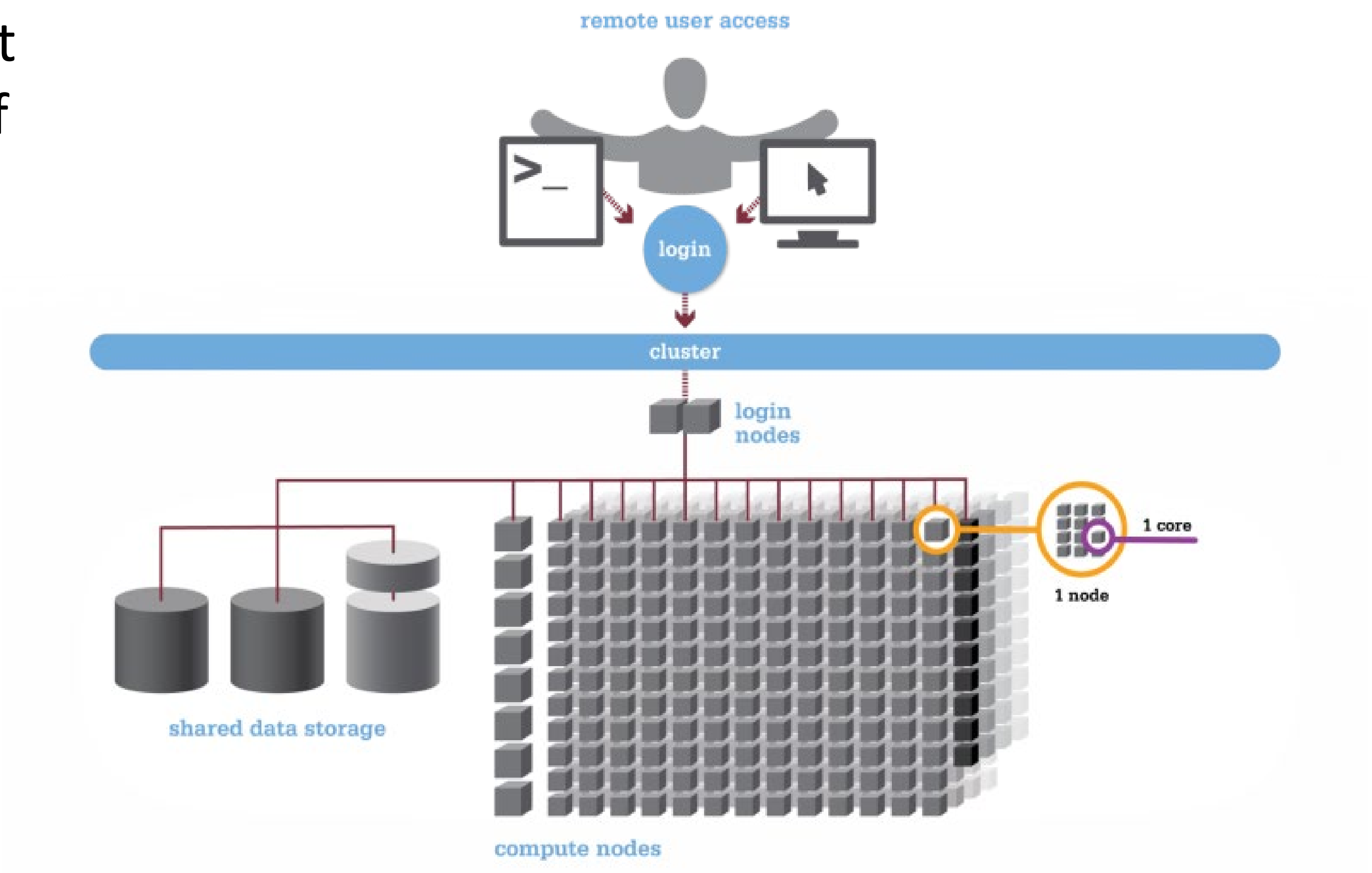
[orc.gmu.edu, wiki.orc.gmu.edu]

GEORGE MASON UNIVERSITY

ABOUT THE ORC CLUSTERS

WHAT IS A HIGH PERFORMANCE COMPUTING (HPC) CLUSTER

- Core (processor) - a single unit that executes a single chain of instructions.
- Node - a single computer or server
- Cluster - many **computers** (nodes) for computation connected on a
- Fast and low latency interconnection **network** hosting a
- Collection of **software**



https://www.osc.edu/book/export/html/2782

ORC HPC CLUSTERS

- **Two clusters at ORC:**
  - **ARGO**
  - **HOPPER**
- Shared-investment cluster
  - Free to use by anyone at Mason
  - Investors have: Higher priority, exclusive access to nodes on request, longer run times

ARGO HARDWARE SPECS

Domain	# Nodes	CPUs	# Cores	Memory	GPUs
General	78	E5-2670 @ 2.60GHz AMD Opt 6276 @ 2.3GHz E5-2660 v3 @ 2.60GHz E5-2670 v3 @ 2.30GHz E5-2660 v4 @ 2.00GHz E5-2680 v4 @ 2.40GHz Gold 5120 @ 2.20GHz	16/20/24/28/64	64 GB - 512 GB	
Large Memory	5	AMD Opt 6276 @ 2.3GH E5-2650 v4 @ 2.20GHz Gold 5120 @ 2.20GHz	24/28/64	512 GB (4x) 1.5 TB (1x)	
GPU	10	E5-2670 v3 @ 2.30GHz E5-2650 v4 @ 2.20GHz Gold 5120 @ 2.20GHz	24/28	128 GB (2x) 256 GB (2x) 768 GB (5x) 1.5 TB (1x)	8x K80 (1x) 4x K80 (2x) 4xV100-PCIE (2x) 4xV100-SXM2 (2x)

HOPPER HARDWARE SPECS

Domain	# Nodes	CPUs	# Cores	Memory	GPUs
Login Nodes	2	Dell PowerEdge R640 Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz	48 / node	384 GB DDR4 RAM	
Login Nodes	2	AMD	64 / node	256 GB RAM	1 NVIDIA T4
Compute Nodes	74	Dell PowerEdge R640 Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz	48 / node	192 GB DDR4 RAM 960 GB SSD Storage	

HOPPER HARDWARE SPECS - CONT

Domain	# Nodes	CPUs	# Cores	Memory	GPUs
Compute Nodes	48	AMD MILAN - 3072 Cores	64 / node	256 GB RAM	
Compute Nodes	20	AMD MILAN - 1280 cores	64 / node	512 GB RAM	
Compute Nodes (High Memory)	20	AMD MILAN 2048 cores	64|128/ node	1|2|4 TB RAM	
GPU	24	AMD MILAN	24 /node	512GB DDR4 RAM	- 4X A100-SXM4-80GB
GPU	21	AMD MILAN	24 /node	512GB DDR4 RAM	- 21X A40
GPU	2	AMD EPYC Rome 7742 CPUs @ 2.60 GHz	128 /node	1TB DDR4 RAM	8X NVIDIA A100-SXM4-40GB GPUs

SOFTWARE

Software available on the cluster includes:

- Low-level development tools:
  - C/C++, Python, MATLAB, R
- Specialized libraries:
  - Communication: MPI, multiprocessing
  - Computation: OpenBLAS, FFTW, LAPACK
  - Machine Learning: TensorFlow, Keras, PyTorch
- Scientific simulation and analysis
  - Amber, Gromacs, Cloudy, QuantumEspresso, Gaussian, LAMMPS

The complete list of software available on the cluster can be found here:

- https://orc.gmu.edu/software-available-software-modules/

ACCESSING THE CLUSTER

1. FROM the SHELL (CLI)
2. FROM OPEN ONDEMAND (WEB DASHBOARD)

ACCESSING THE CLUSTER

Before starting:

- To use the cluster efficiently, a basic level of proficiency with linux commands and philosophy is necessary. An online tutorial is available to help build linux skills at http://www.ee.surrey.ac.uk/Teaching/Unix/

- The ORC also organizes Carpentry Workshops (in collaboration with Library Digital Services) on topics such as Introduction to the Bash Shell and Python. Look for upcoming workshops on our events pages - https://orc.gmu.edu/events-calendar/

ACCESSING THE CLUSTER - CLI

ACCESSING THE CLUSTER - SSH

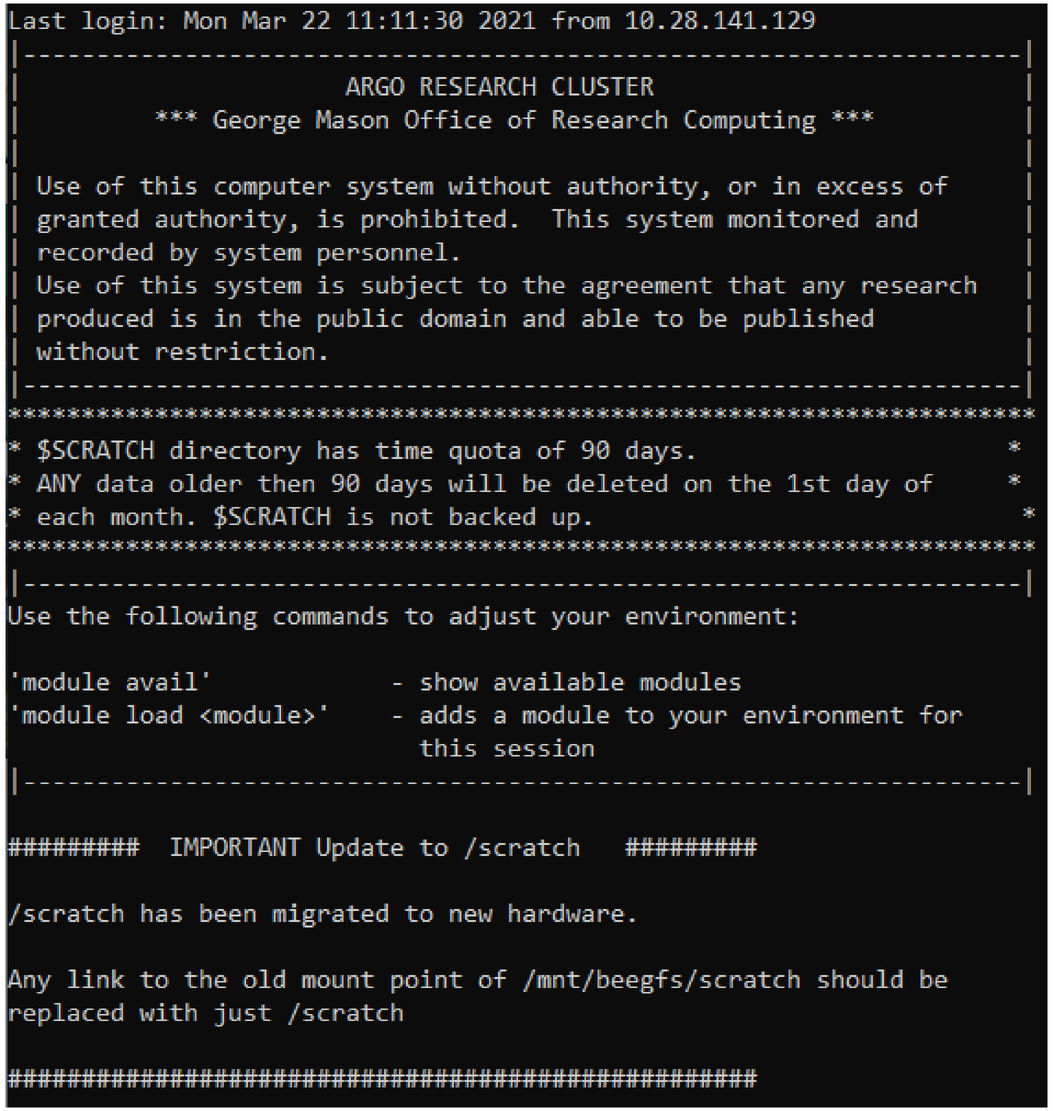
Log into the head nodes using Secure Shell ”ssh”:

$ ssh <userID>@hopper.orc.gmu.edu

Use a terminal/shell and the command above on a Linux/Mac or Windows (10) Machine.

Log in credentials are your Mason Net-ID and Password.

For convenience you can consider setting up passwordless SSH from your laptop or desktop; this will allow you to login securely without entering your password. The steps for this are described at http://wiki.orc.gmu.edu/index.php/Uploading_Data



```
Last login: Mon Mar 22 11:11:30 2021 from 10.28.141.129
|---------------------------------------------------------------------|
|                       ARGO RESEARCH CLUSTER                         |
|          *** George Mason Office of Research Computing ***          |
|                                                                     |
| Use of this computer system without authority, or in excess of      |
| granted authority, is prohibited.  This system monitored and        |
| recorded by system personnel.                                       |
| Use of this system is subject to the agreement that any research    |
| produced is in the public domain and able to be published           |
| without restriction.                                                |
|---------------------------------------------------------------------|
**********************************************************************
* $SCRATCH directory has time quota of 90 days.                      *
* ANY data older then 90 days will be deleted on the 1st day of      *
* each month. $SCRATCH is not backed up.                              *
**********************************************************************
|---------------------------------------------------------------------|
Use the following commands to adjust your environment:

'module avail'            - show available modules
'module load <module>'    - adds a module to your environment for
                            this session
|---------------------------------------------------------------------|

#########  IMPORTANT Update to /scratch   #########

/scratch has been migrated to new hardware.

Any link to the old mount point of /mnt/beegfs/scratch should be
replaced with just /scratch

###################################################
```

ACCESSING THE CLUSTER – AVAILABLE LOGIN/HEAD-NODES ON HOPPER

It is possible to ssh directly into either the AMD login nodes or the Intel login nodes.

To access the Intel head nodes hopper1/hopper2 directly, use

```
$ ssh netid@hopper-intel.orc.gmu.edu
```

To access the AMD head nodes hopper-amd-1/hopper-amd-2 directly, use

```
$ ssh netid@hopper-amd.orc.gmu.edu
```

Name	CPUS	GPU	Memory
hopper-intel [2]	48 Intel	0	384 GB RAM
hopper-amd [2]	64 AMD	1 NVIDIA T4	256 GB RAM

ACCESSING THE CLUSTER – HEAD NODE ETIQUETTE

The head nodes are used primarily for submitting jobs to the cluster through the resource manager SLURM.

It is ok to prepare jobs by, for example, compiling code, and running small tests.

The head nodes are not to be used for intensive long running work. Any such intensive use will result in your processes being killed and habitual abuse of this rule may result in suspension of access to the cluster.

If more extensive testing or debugging is required then start an interactive session on a compute node. We will go into this later in the tutorial.

ACCESSING THE CLUSTER – TRANSFERRING FILES

Move files to and from the cluster using the Secure Shell "scp":

```
$ scp local-file <GMUnetID>@hopper.orc.gmu.edu:/path/to/remote-file
```

```
$ scp <GMUnetID>@hopper.orc.gmu.edu:/path/to/remote-file local-file
```

For efficient transfer of large amounts of data, it is advisable to use the Globus File Transfer client. Check the ORC wiki for more information on setting up and using GLOBUS: http://wiki.orc.gmu.edu/index.php/Using_Globus

Datasets

/datasets/

FILES AND STORAGE

$HOME (/home/<UserID>)

Read/write on the login and compute nodes (hopper1 and hopper2, hop-amd-1 and hop-amd-2)

Limited to 60 GB per user

Backed up.

$SCRATCH (/scratch/<UserID>)

Read/write on every node (login and compute) - jobs can write output here

Scratch directories have no space limit

Temporary: Data purged after 90 days - So make sure to move your results to a safe place

Not backed up.

/projects/<project-owner>

Additional persistent storage for projects whose storage requirements exceed 50 GB

Faculty can request up to 1 TB of disk space if needed. Students, and Postdocs should request access via their advisor/supervisor.

Read/write on every node

Not backed up.

ENVIRONMENT MODULES

MANAGING YOUR ENVIRONMENT

Environment modules are used to manage the specialized software on the cluster.

Modules are short scripts that automatically configure your environment (i.e. set the PATH, MANPATH and the LD_LIBRARY_PATH and other environment variables) for the software you want to use.

You can see the list of available modules by typing:

```
[user@hopper-x ~]$ module avail
```

You can check the modules already in your environment by typing:

```
[user@hopper-x ~]$ module list
No Modulefiles Currently Loaded.
```

You can load your choice of module from the available modules using the following command:

```
[user@hopper-x ~]$ module load <module-name>
```

Remove any loaded module using the command:

```
[user@hopper-x ~]$ module unload <module-name>
```

To ensure all modules are unloaded you should use: `module purge`

To reset your shell environment to the defaults use: `module reset`

MANAGING YOUR ENVIRONMENT

To get the latest software stack on Hopper (compiled to run across all nodes), after logging in:

```
[user@hopper-x  ~]$  module  load gnu10
```

```
[user@hopper-x  ~]$  module  avail python
```

```
[user@hopper-x  ~]$  module  load python
```

SLURM SCHEDULER

SLURM – SIMPLE LINUX UTILITY FOR RESOURCE MANAGEMENT

- SLURM schedules the jobs on the compute nodes
- Job scripts are submitted to SLURM along with specific resource requests e.g.:
  - Time
  - Processor cores
  - Memory
  - GPU to run their jobs
- Commands to monitor jobs
  - sbatch, squeue, scancel, sacct, sinfo, scontrol, salloc

- Getting started with SLURM: http://wiki.orc.gmu.edu/mkdocs/Getting_Started_with_SLURM/

SLURM DEFAULTS

- The directory where a job runs is by default the working directory from which you submitted the script.

- Your existing environment is exported when you submit your job, and may cause unintended results. To prevent this, use:

  #SBATCH --export NONE

- By default SLURM will allocate 1 CPU and 2 GB of memory per cpu request.

SLURM SCRIPT

- A SLURM submission script is a UNIX shell script
  1. Give information to SLURM about your job using “#SBATCH …” commands. These are special comments that SLURM can interpret
  2. Load necessary modules that your job needs
  3. Execute your program or script

- Slurm sample template script:

  http://wiki.orc.gmu.edu/mkdocs/Hopper_Quick_Start_Guide/#sample-slurm-job-script

ANATOMY OF A BASIC SLURM SCRIPT

```
#!/bin/sh

## An SBATCH command is "commented out" (ignored) if line begins with 2 "#"
## Anything in <angle brackets> needs to be modified by you (remove <> too)

## Specify a name for Slurm to use when displaying your job.
#SBATCH --job-name=<MyJobName>

## Select a partition (queue) to submit your job to.
#SBATCH --partition=normal

## Deal with output and errors.  Separate into 2 files (not the default).
## May help to put your result files in a directory: e.g. /scratch/%u/logs/...
## NOTE: %u=userID, %x=jobName, %N=nodeID, %j=jobID, %A=arrayID, %a=arrayTaskID
#SBATCH --output=/scratch/%u/%x-%N-%j.out  # Output file
#SBATCH --error=/scratch/%u/%x-%N-%j.err   # Error file
#SBATCH --mail-type=BEGIN,END,FAIL         # ALL,NONE,BEGIN,END,FAIL,REQUEUE,..
#SBATCH --mail-user=<GMUnetID>@gmu.edu     # Put your GMU email address here

## This script will request 1 CPU and 2GB of memory

## Load relevant modules needed for the job
module load <module_name>
…
## Run your program or script
<command(s) to run your program>
…
```

```
## Number of nodes/cores
#SBATCH --nodes=<N>    # Number of nodes
#SBATCH --ntasks-per-node=<Z>  # <T> = <N> * <Z>
#SBATCH --cpus-per-task=<num-threads>

## Memory
#SBATCH --mem=<X>  # Memory per task (units: K,M,G,T)
#SBATCH --mem-per-cpu=<Y>  # Memory per thread

## Time
#SBATCH --time=<D-HH:MM>  # Run time Days-Hours:Min
```

SLURM PARTITIONS ON HOPPER

To run, jobs must be submitted to a partition (aka queue).

SLURM provides various partitions (aka queues) with differing Quality of Service (QoS)

Available partitions:

Partition	Time Limit (Days-Hours:Min)	Description
debug	0-01:00	intended for very short tests
normal	5-00:00	default queue, access to cpu nodes
bigmem	5-00:00	access to large memory nodes
contrib	5-00:00	users can submit jobs to this partition however they will be subject to preemption
gpuq	3-00:00	gpu node access
gpuq-contrib	3-00:00	Similar to the cpu contrib, but for gpu nodes
interactive	0-12:00	interactive jobs e.g Open OnDemand

RUNNING AND MONITORING JOBS

To submit a job to the scheduler: **`$ sbatch <your-submit-script>`**	**To start an interactive session:** **`$ salloc <slurm-parameters>`**

To monitor the status of jobs in the queue
`$ squeue  -u <username>`
 	
`$ sacct –X`
	Cancel a job
	`$ scancel <JobID>`

Review status of nodes
`$ sinfo`
	Check resource usage
	`$ seff <jobID>`
	
	`$ sacct -o JobID,Elapsed,MaxRSS`

SPECIFYING PARTITION AND QOS

The default partition is the normal partition, which comes paired with the ‘normal’ qos. To run on this partition

```
#SBATCH --partition=normal
#SBATCH --qos=normal ## defaults to normal
```

To run on the *contrib* partitions, you need the correct qos. Running with the normal qos exposes the job to preemption. Contributing group members get a group qos that gives them access to the group nodes

```
#SBATCH --partition=contrib
#SBATCH --qos=<group-name>
```

If using gpus, the gpu qos has to be specified as well

```
#SBATCH --partition=gpuq
#SBATCH --qos=gpu
```

PARALLEL JOBS

SPECIFYING NODES (AMD VS. INTEL)

When running multi-core or multi-node jobs, it is important to use constraints so that SLURM assigns similar nodes to the same job.

To run on INTEL nodes (CPU jobs)

```
#SBATCH --constraint=intel
```

To run on the AMD nodes (CPU or GPU jobs)

```
#SBATCH --constraint=amd
```

To run gpu jobs on the dgx nodes

```
#SBATCH --constraint=dgx
```

PARALLEL JOBS – JOB ARRAYS

Job Arrays simply run your program multiple times

```
#SBATCH --array=1-12    # Runs your program 12 times
#SBATCH --array=1-12%3  # Run 12 times, 3 at a time
```

You may want to change your output files

```
## NOTE: %u=userID, %x=jobName, %N=nodeID, %j=jobID, %A=arrayID, %a=arrayTaskID
#SBATCH --output=/scratch/%u/%x-%N-%A-%a.out  # Output file
#SBATCH --error=/scratch/%u/%x-%N-%A-%a.err   # Error file
```

Your program can find out which run it belongs to by reading the **'$SLURM_ARRAY_TASK_ID'** environment variable:

Java	`int ID=Integer.parseInt(System.getenv("SLURM_ARRAY_TASK_ID"));`
C	`int ID = atoi(getenv("SLURM_ARRAY_TASK_ID"));`
Python	`ID = int(os.environ["SLURM_ARRAY_TASK_ID"])`

Important: do not run array tasks that complete too quickly, as the time to setup and tear down after each job is longer than the runtime. **Rule of thumb: array tasks should run for 5 minutes or longer.**

PARALLEL JOBS – THREADS

- Most languages have a way to create threads
- Many tools and libraries can also use threads
  - OpenMP (C/C++), multiprocessing (Python), OpenBLAS (C/C++/R), etc.
- Simply ask Slurm for more CPUs

```
#SBATCH --cpus-per-task=<num-threads>
```

- Make sure that <num-threads> does not exceed the number of cores on a node.

PARALLEL JOBS - MPI

- Programs can communicate directly via sockets
- Almost everyone uses the Message Passing Interface (MPI) library - MPI libraries exist for almost all languages
- Slurm gives you the ***option*** to affect how processes will be distributed over the nodes
  - `#SBATCH --nnodes=<N>    # Number of nodes`
  - `#SBATCH --ntasks-per-node=<Z>  # <T> = <N> * <Z>`
- Slurm jobs can start multiple processes (tasks)
  - `#SBATCH --ntasks=<T>    # Number of processes`
  - `<T> >= <N>`

GPU JOBS

GPU JOBS

To use a GPU, your submit script must include:

1 - The partition and QOS:

```
#SBATCH --partition=gpuq  ## or “contrib-gpuq”
#SBATCH --qos=gpu         ## or group qos for “contrib-gpuq”
```

2 - The GRES parameter:

```
#SBATCH --gres=gpu:1g.10gb:<number-of-gpus>
```

3 – The number of nodes/tasks and cores:

```
#SBATCH --nodes=1
#SBATCH –-ntasks-per-node=<number-of-cpus> ##
```

More details on running GPU jobs: https://wiki.orc.gmu.edu/mkdocs/Running_GPU_Jobs/

GPU PARTITIONING ON HOPPER

Count /Node	Device	Compute	Memory	Total Available
8	1g.10gb	1/7 of A100.80gb	1/8 of A100.80gb	64
4	2g.20gb	2/7 of A100.80gb	1/4 of A100.80gb	32
4	3g.40gb	3/7 of A100.80gb	1/2 of A100.80gb	32

- 20 of the 24 A100.80GB GPUs (GPU ID 00-016) are not partitioned
- (GPUs 017-024) are partitioned into slices of different sizes [Multi-Instance GPU (MIG)]
- To request a full GPU use []:
    - --gres=gpu:A100.40gb:1
    - --gres=gpu:A100.80gb:1
- You can request smaller slices of a GPU using:
    - --gres=gpu:1g.10gb:1
- Analyze the demands of your job and determine which GPU size is available and suitable for it.

More details on running GPU jobs: https://wiki.orc.gmu.edu/mkdocs/Running_GPU_Jobs/

GPU JOBS

There are a number of libraries/frameworks that give you access to GPUs (CUDA, Caffe, OpenACC, Tensorflow, PyTorch, Keras, Gromax, NAMD, etc.) that are available on the cluster

A GPU job may receive errors related to CUDA libraries. To debug GPU scripts, either work from the AMD login nodes or, preferably, use the "salloc" command which will give you an interactive session on one of the GPU nodes

```
 $ salloc --partition=gpuq --qos=gpu –-nodes=1 --ntasks-per-node=8 \
    --gres=gpu:1g.10gb:<number-of-gpus>
```

More details on running GPU jobs: https://wiki.orc.gmu.edu/mkdocs/Running_GPU_Jobs/

ADDITIONAL INFORMATION

GETTING JOBS SCHEDULED SOONER

Slurm can give higher priority to a job when it knows more about the resources it needs

```
#SBATCH --mem=<X>  # Memory per task (units: K,M,G,T)
#SBATCH --time=<D-HH:MM>  # Run time Days-Hours:Min
```

If using threads, you may prefer to specify the memory requirements per thread:

```
#SBATCH --mem-per-cpu=<Y>  # Memory per thread
```

Be sure to add a safety margin to these values (sacct, seff). If they are exceeded, Slurm will kill your job

CHECKPOINTING

Checkpointing your jobs while running on the cluster is highly recommended.

Checkpointing stores the internal state of your calculation periodically so the job can be restarted from that state, e.g. if the node crashes or the time limit for the job partition is reached.

Ideally, checkpointing is done internally in your application (it is built into many open source and commercial packages)

For applications that don't have inbuilt support for checkpointing you can use checkpointing tools such as DMTCP. We have outlined the steps for using DMTCP to checkpoint your runs on the ORC wiki page: http://wiki.orc.gmu.edu/index.php/Creating_Checkpoints_(DMTCP)

MANAGING PYTHON PACKAGES

It is strongly recommended to **avoid using conda python environments** for managing python libraries.

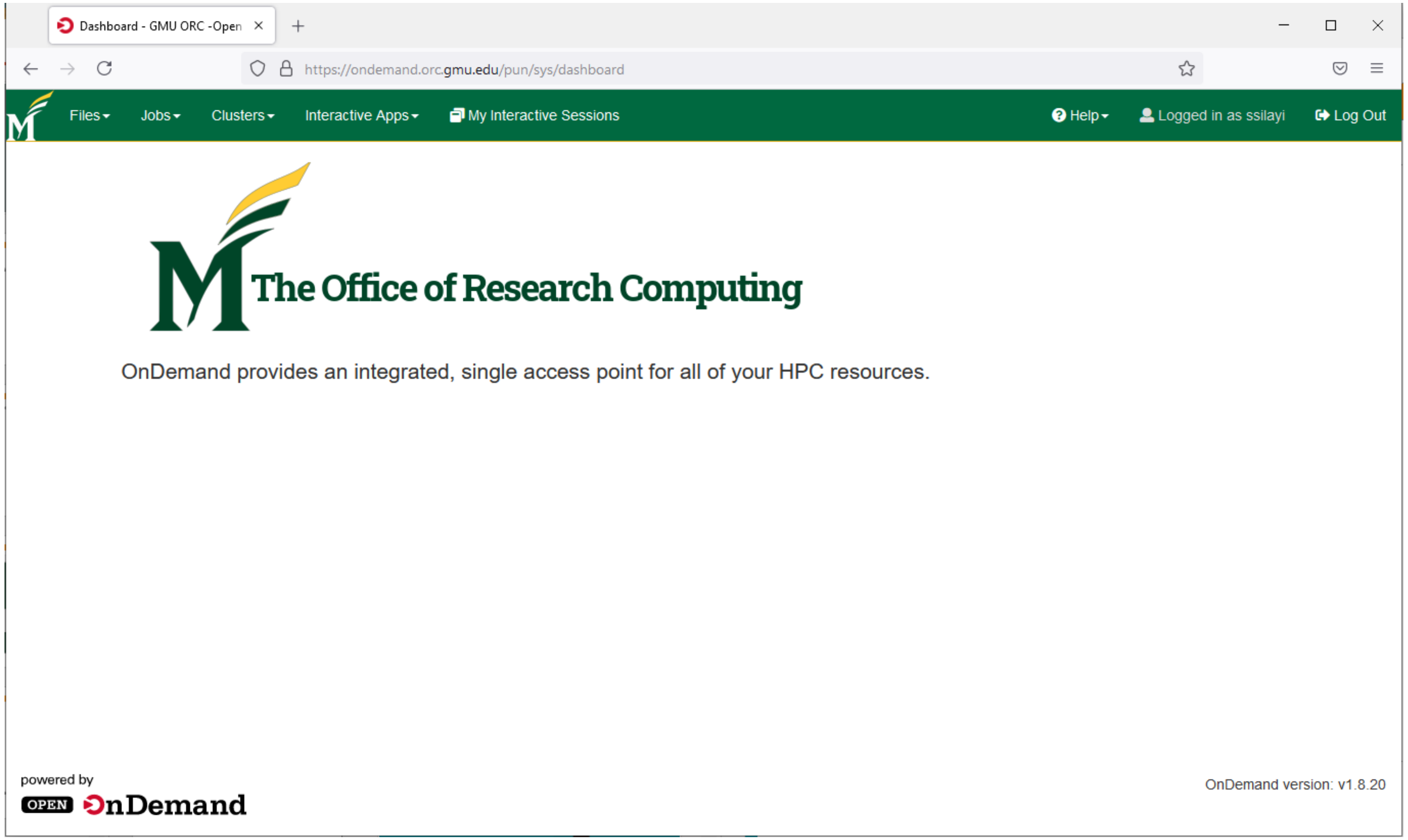
Instead, use python virtual environments to install and manage any external or additional python libraries you might need while working on the cluster.

[DEMO]

ACCESSING THE CLUSTER – OPEN ONDEMAND

THE OPEN ONDEMAND (OOD) WEB SERVER

Connect to the Open OnDemand web server by pointing your browser to: https://ondemand.orc.gmu.edu

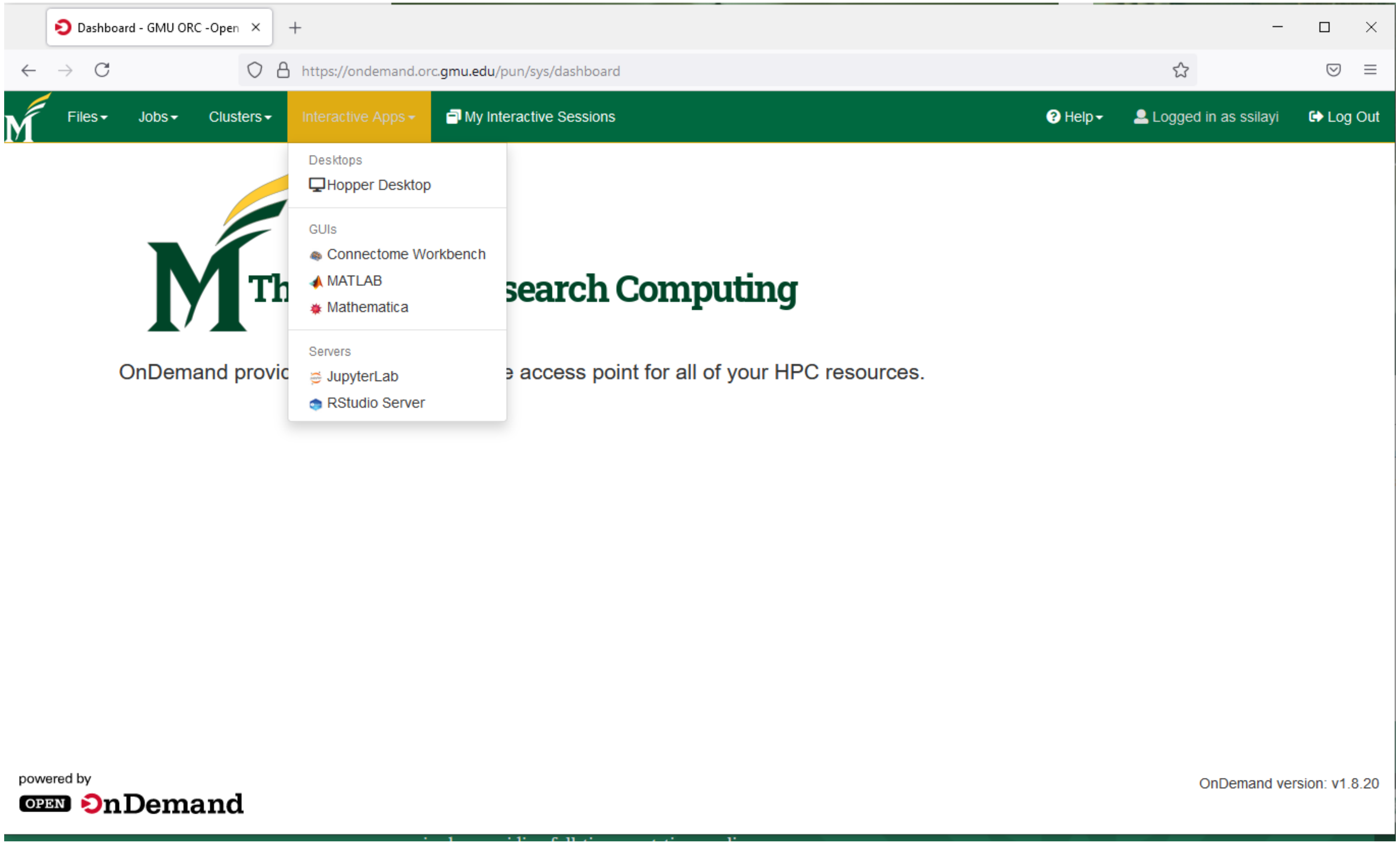

Dashboard - GMU ORC -Open
https://ondemand.orc.gmu.edu/pun/sys/dashboard
Files
Jobs
Clusters
Interactive Apps
My Interactive Sessions
Help
Logged in as ssilayi
Log Out
The Office of Research Computing

OnDemand provides an integrated, single access point for all of your HPC resources.

powered by OPEN OnDemand

OnDemand version: v1.8.20

OOD - APPS



Dashboard - GMU ORC -Open

https://ondemand.orc.gmu.edu/pun/sys/dashboard

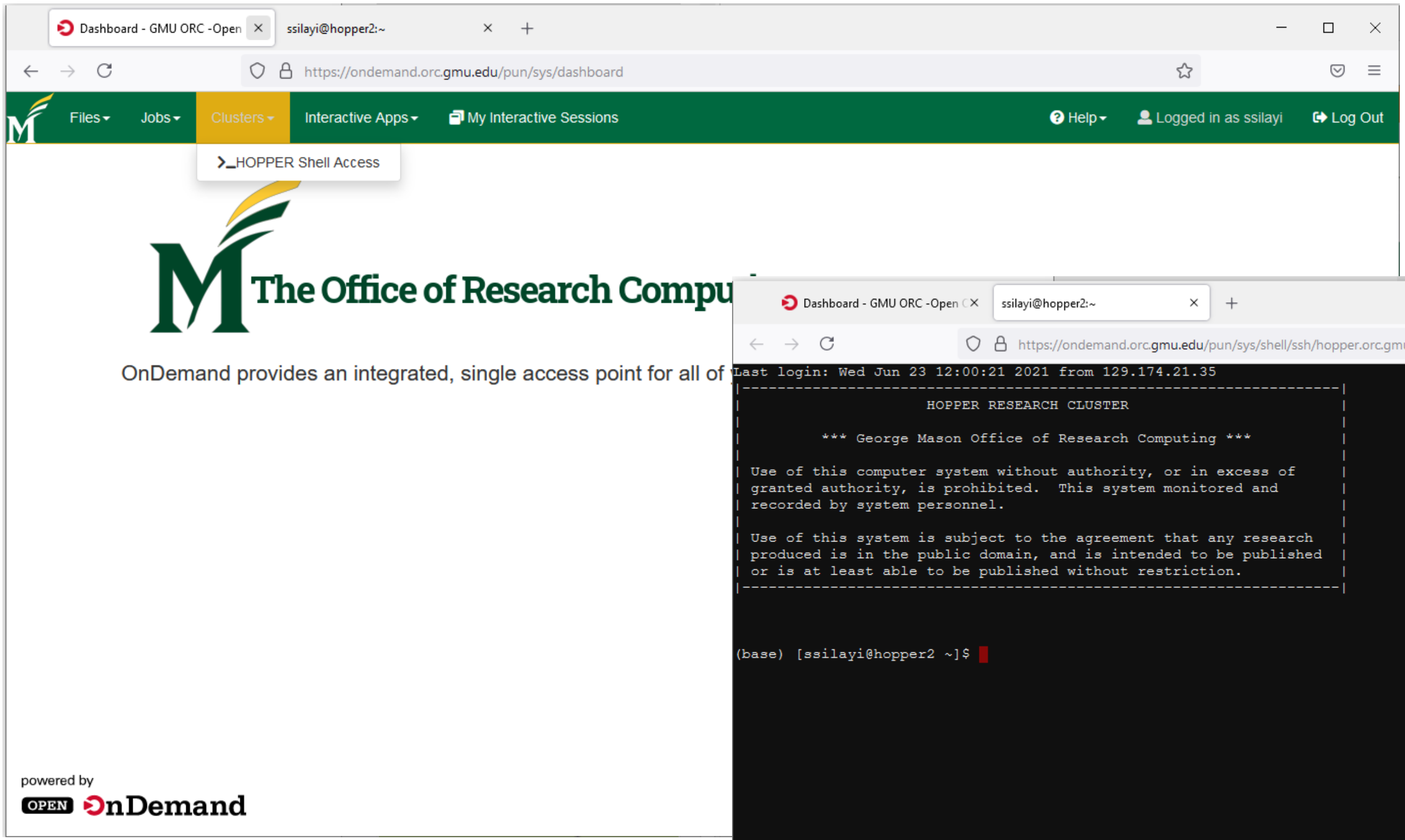
Th… search Computing

OnDemand provic… e access point for all of your HPC resources.

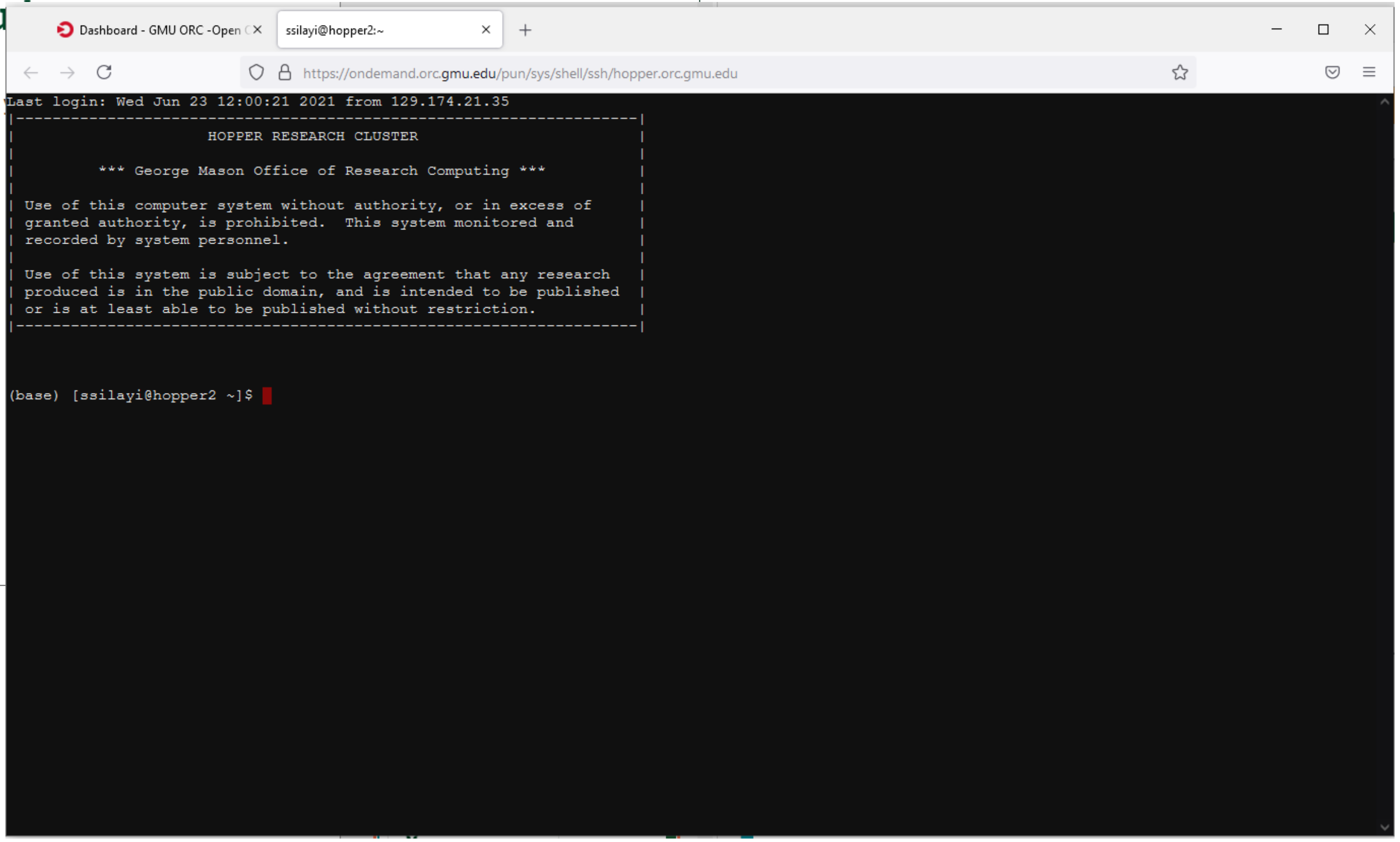
powered by OPEN OnDemand

OnDemand version: v1.8.20

OOD – SHELL ACCESS



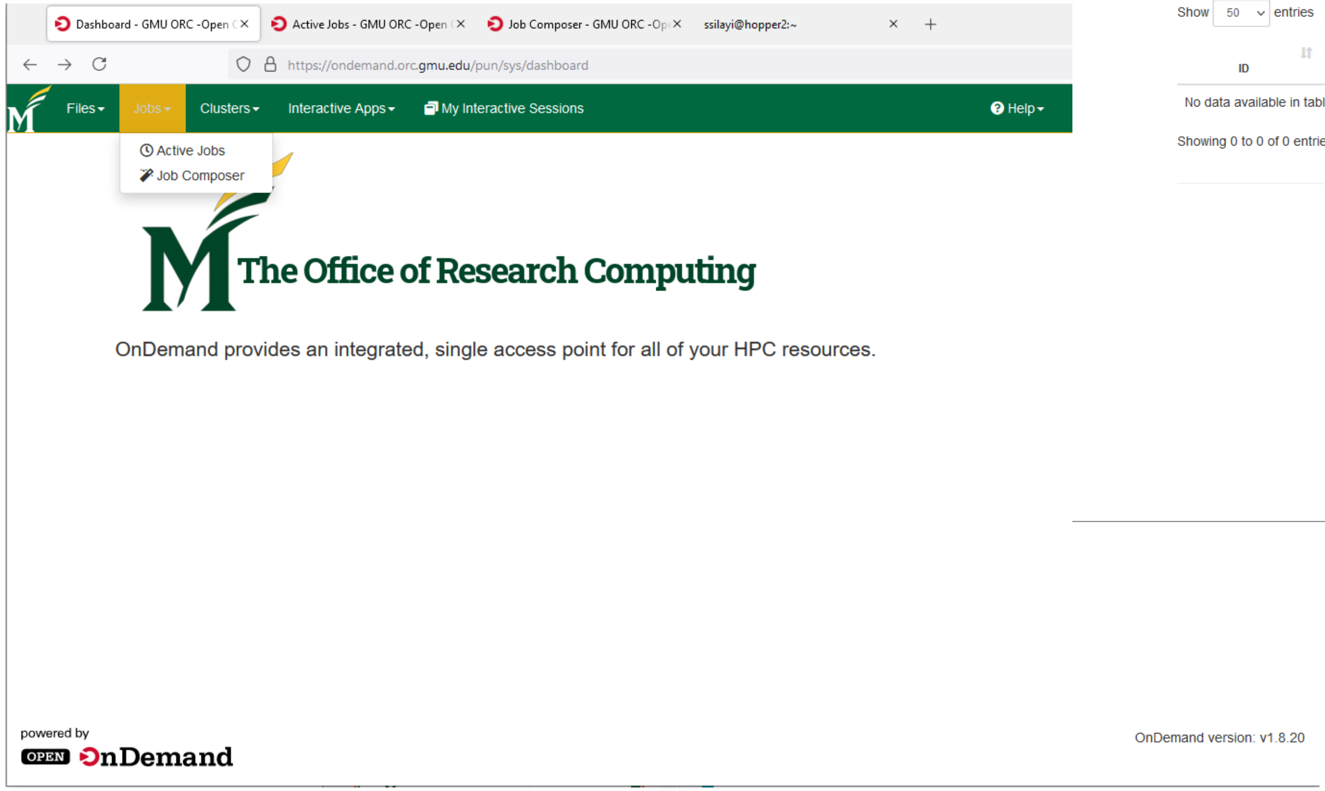
OnDemand provides an integrated, single access point for all of


Dashboard - GMU ORC -Open
ssilayi@hopper2:~
https://ondemand.orc.gmu.edu/pun/sys/shell/ssh/hopper.orc.gmu.edu

```
Last login: Wed Jun 23 12:00:21 2021 from 129.174.21.35
|-----------------------------------------------------------------------|
|                        HOPPER RESEARCH CLUSTER                        |
|                                                                       |
|          *** George Mason Office of Research Computing ***            |
|                                                                       |
| Use of this computer system without authority, or in excess of       |
| granted authority, is prohibited.  This system monitored and         |
| recorded by system personnel.                                         |
|                                                                       |
| Use of this system is subject to the agreement that any research     |
| produced is in the public domain, and is intended to be published    |
| or is at least able to be published without restriction.              |
|-----------------------------------------------------------------------|


(base) [ssilayi@hopper2 ~]$ 
```

OOD – MANAGING JOBS



Dashboard - GMU ORC -Open | Active Jobs - GMU ORC -Open | Job Composer - GMU ORC -Op | ssilayi@hopper2:~

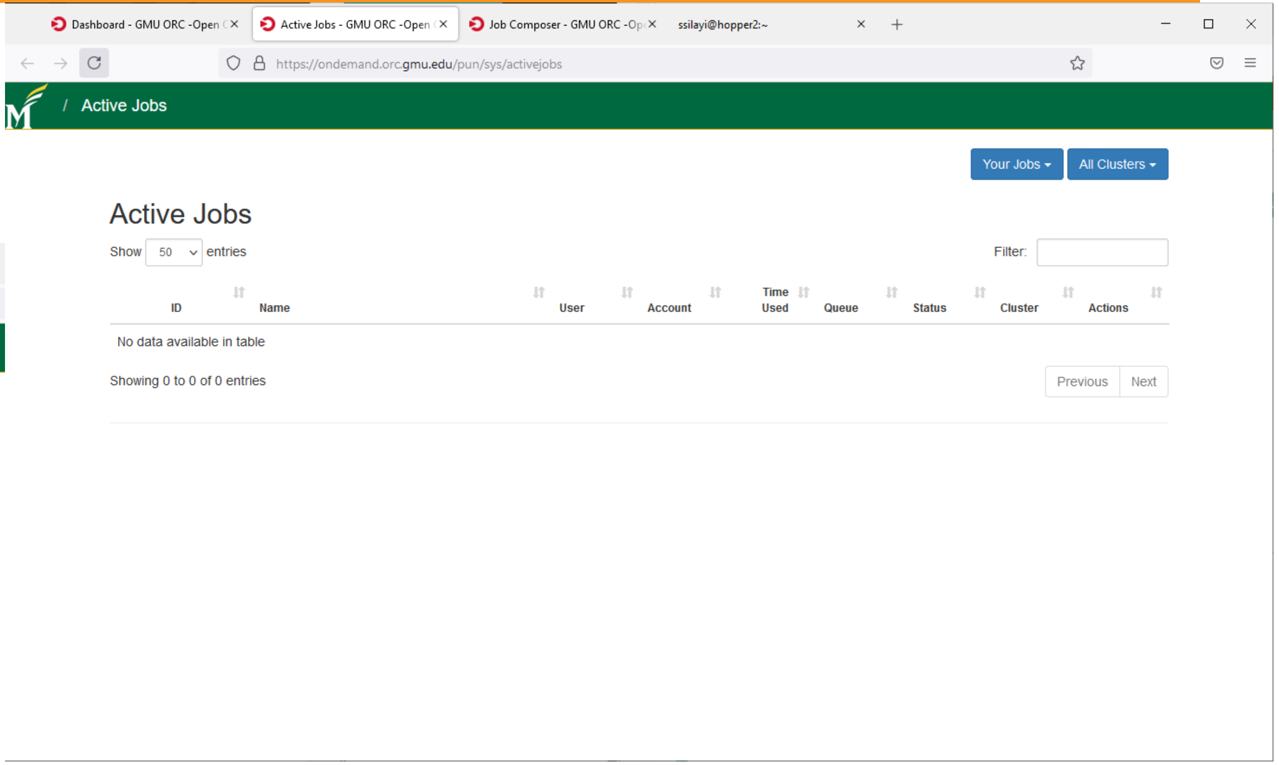
https://ondemand.orc.gmu.edu/pun/sys/dashboard

The Office of Research Computing

OnDemand provides an integrated, single access point for all of your HPC resources.

powered by OPEN OnDemand

OnDemand version: v1.8.20



Dashboard - GMU ORC -Open | Active Jobs - GMU ORC -Open | Job Composer - GMU ORC -Op | ssilayi@hopper2:~

https://ondemand.orc.gmu.edu/pun/sys/activejobs

Active Jobs

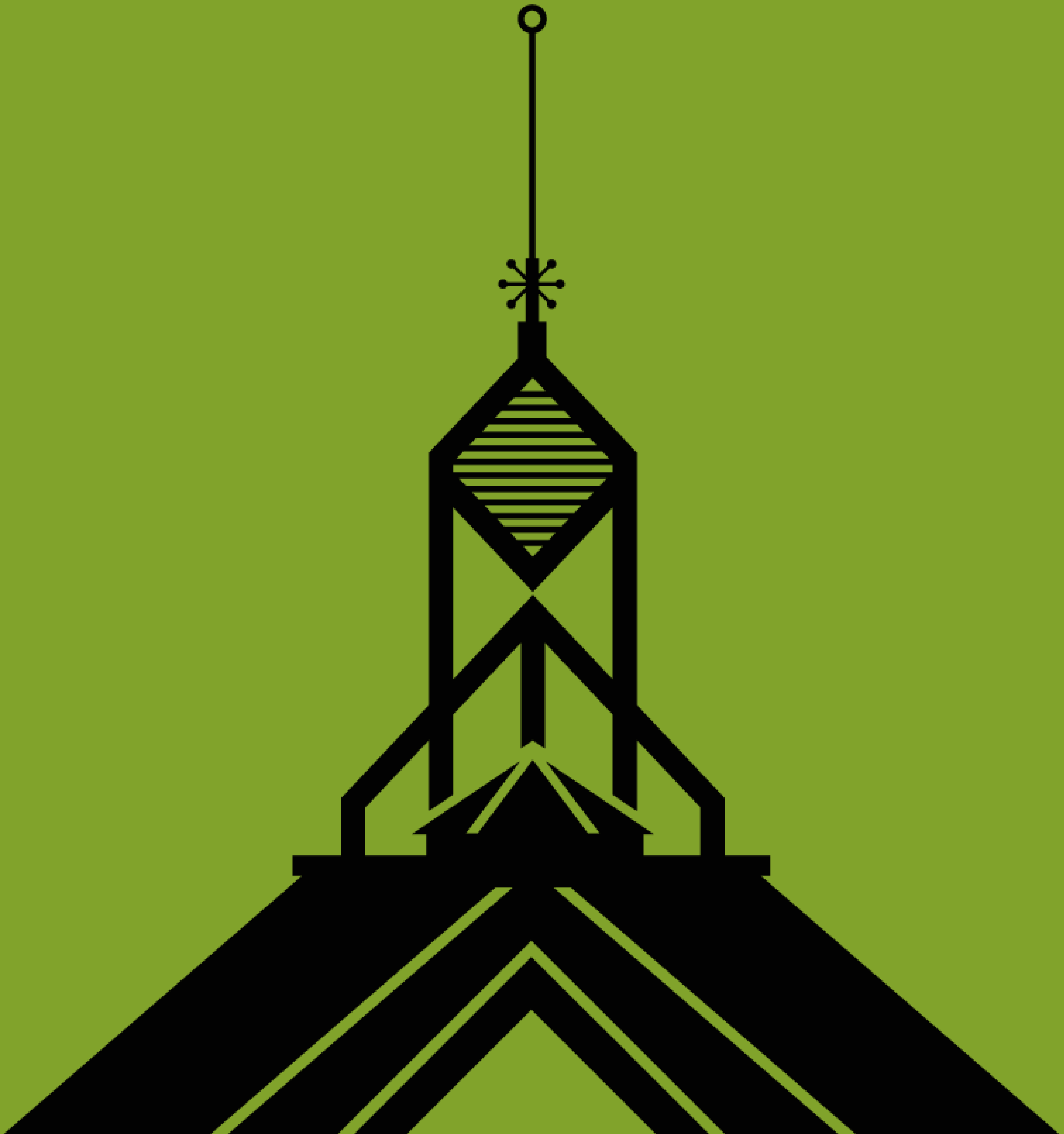
Show 50 entries Filter:

ID	Name	User	Account	Time Used	Queue	Status	Cluster	Actions
No data available in table								

Showing 0 to 0 of 0 entries

Previous Next

GETTING HELP



GETTING HELP

Our Wiki is full of "How To" information:

http://wiki.orc.gmu.edu

Send email to **orchelp@gmu.edu**

- This creates a ticket in our ticketing system
- Will be answered by available personnel
- Include the submit script, error message and any other relevant information that will help us debug the problem