HPC Documentation

Release 2.0.0

Research Computing, Northeastern University

Jul 20, 2023

WELCOME

A dedicated platform for researchers, scholars, and students to access, learn, and leverage our state-of-the-art HPC resources.

Open OnDemand (OOD) OOD provides a various available software interactively through your favorite browser.

Learn more »

Classroom Integration Request a reservation devoted for the classroom, whether an entire curriculum or specific assignments and lessons.

Learn more »

Best Practices Learn to optimize your projects by following some best practices.

Learn more »

Note: Access to the HPC Cluster is subject to university policy and guidelines. Please ensure that you read and understand these guidelines before using the facility. If you require assistance or have any questions, please do not hesitate to contact our dedicated support team.

The Northeastern University HPC Cluster is a high-powered, multi-node, parallel computing system designed to meet the computational and data-intensive needs of various academic and industry-oriented research projects. The cluster incorporates cutting-edge computing technologies and robust storage solutions, providing an efficient and scalable environment for large-scale simulations, complex calculations, artificial intelligence, machine learning, big data analytics, bioinformatics, and more.

Whether you are a seasoned user or just beginning your journey into high-performance computing, our portal offers comprehensive resources, including in-depth guides, tutorials, best practices, and troubleshooting tips. Furthermore, the platform provides a streamlined interface to monitor, submit, and manage your computational jobs on the HPC cluster.

We invite you to explore the resources, tools, and services available through the portal. Join us as we endeavor to harness the power of high-performance computing, enabling breakthroughs in research and fostering innovation at Northeastern University.

CHAPTER ONE

WELCOME

Welcome to the official online help system for Northeastern University's Research Computing team! This documentation will assist you in using Northeastern's HPC cluster system. It contains the most up-to-date information and instructions on obtaining an account, connecting to the system, loading modules, running jobs, and storing data. If you are unfamiliar with high-performance computing, we recommend taking our training courses before using the system. For information on training and services, visit our Research Computing team website.

This help system is frequently updated to reflect the latest discovery information and add new topics to assist you with your research. Check back often for updates.

We value your feedback! If you have any comments or suggestions for topics you want to be covered in this documentation, please submit a Research Computing Documentation request.

1.1 What is Discovery

Discovery is a high-performance computing (HPC) resource for the Northeastern University research community. The Discovery cluster is in the Massachusetts Green HPC Center (MGHPC) in Holyoke, MA. The MGHPC is a 90,000-square-foot, 15-megawatt research computing, and data center facility that houses computing resources for five institutions: Northeastern, BU, Harvard, MIT, and UMass.

CHAPTER

TWO

GETTING HELP

If you need help, you can contact the Research Computing (RC) team via email, ServiceNow ticket, or by scheduling an appointment through our Bookings page.

2.1 Email

To contact the RC team, email us at rchelp@northeastern.edu. This will generate a ticket in ServiceNow. Be sure to include details about your question or issue, including any commands or scripts you use, so that we can direct you to the right person.

2.2 Submit a ticket

To submit a ticket in ServiceNow, select from the RC ServiceNow catalog. You may need to sign in with your Northeastern username and password to view the catalog.

RC users can request services using our ticket system. Please select the appropriate category below to access the online ticket.

Get Assistance with RC RC Access Form Software Request Form Documentation Request Partition Access Request Storage Request Storage Extension Request Data Transfer Consultation Classroom Request Form Unsubscribe

2.3 Scheduling consultations

We encourage you to schedule a consultation with one of our staff members to receive personal, one-on-one assistance for your research computing and data storage needs. Consultations are available to any Northeastern student, faculty, or staff member. We can assist you with starting Discovery, optimizing your code, benchmarking, installing and using software packages, detailing data storage options, and more.

We offer consultations by appointment most weekdays during regular business hours (9 AM to 5 PM). Please note that we follow the Northeastern University holiday schedule, so no consultations are available on holidays or during breaks. All consultations are conducted online through the Teams app.

Use our Bookings page to view our availability and schedule an appointment. You must sign in using your **@northeastern.edu** email address (for example, **a.student@northeastern.edu**).

2.4 Update Ticket

To check for updates on a submitted ticket, please follow these steps:

1. Log in to your ServiceNow account.
2. Select "My Tickets" to access a list of all your active tickets.
3. In the ticket list, you will be able to view the latest updates made to each ticket.

In addition, you will receive an email notification from ServiceNow mentioning your incident number, you can directly access the ServiceNow portal to view the updates on your ticket by following these steps:

1. Open the email and locate the incident number mentioned in the message.
2. Select the incident number; this will redirect you to the ServiceNow portal.
3. In the ServiceNow portal, you can see the updates made to your ticket.

By following these steps, you can easily track the progress and stay informed about any updates related to your submitted tickets.

2.5 Status Page

We use the ITS Systems Status page to post important information, such as power outages and maintenance windows. You can subscribe to this page to receive email updates on the status of ITS systems.

CHAPTER THREE

GETTING ACCESS

3.1 Request an account

To access Discovery, you must first have an account. You can request an account through ServiceNow but need a Northeastern username and password. If you are new to the university or a visiting researcher, you should work with your sponsor to obtain a Northeastern username and password.

Important: If you previously had access to Discovery but are now working with a different PI, you should submit a ServiceNow RC Access Request form and enter the name of your current PI in the Sponsor field. This will link your account to your current PI and expedite updating your account with any of your current PI's resources on Discovery, such as shared storage or a private partition.

Valid NU Credentials

Access to the cluster is limited to Northeastern affiliates with a valid NU username and password. Research Computing cannot create or renew Northeastern accounts. You must work with your sponsor to obtain or update your credentials.

For **non-Northeastern personnel**, request a Northeastern sponsored account using this kb article.

To request an account, follow these steps:

1. Visit the ServiceNow RC Access Request form.
2. Complete the form, check the acknowledgment box, and submit it.

Your request may take up to 24 hours after your sponsor approves it (see Sponsor Approval Process below). You will receive an email confirmation when your access has been granted. Once you have access, if you are unfamiliar with Discovery, high-performance computing, or Linux, you may want to take one of our training courses. Visit the Research Computing website for more information about our training and services.

PI and instructor access

If you are a PI, professor, or instructor at Northeastern and need access to the cluster, use the access form in the above procedure and enter your name in the `Sponsor Name` field.

3.1.1 Sponsor Approval Process

HPC users need a sponsor, usually a NU PI or professor, to approve their request. PIs, professors, and instructors can sponsor themselves. Students (undergraduate or graduate), visiting researchers, or staff members must have a sponsor approve their request. When you fill out the ServiceNow form, an email is sent to the specified sponsor upon submitting the request. Sponsors will receive email reminders until they approve the request through the link in the email to ServiceNow. We recommend letting your sponsor know to look for the email with the approval link before submitting an access request.

3.2 Cluster Usage

Important: It is best not to use the login node for CPU-intensive activities, as this will impact the performance of this node for all cluster users. It will also not provide the best performance for the tasks you are trying to accomplish. For more information, please refer to our documentation on *Connecting Mac* or *Connecting Windows*.

If you are attempting to run a job, you should move to a compute node. You can do this interactively using the `srun` command or non-interactively using the `sbatch` command. Please see our documentation on *Batch Jobs: sbatch* and *Interactive Jobs: srun Command* for more information.

If you are attempting to transfer data, we have a dedicated transfer node that you should use. Please see our documentation on *Transferring Data* for more information.

If you have any questions or need further assistance, please email us at rchelp@northeastern.edu or book a consultation using the link on our Consultation page.

3.3 Cluster Maintenance

To ensure that your job scripts account for the scheduled shutdown period of the cluster, use the `t2sd` script in the `--time` option when submitting your jobs. This script calculates the remaining time until the cluster becomes unavailable and sets the appropriate time limit for your job. Here's an example of how to use it.

- If you usually use the `srun` command:

```
srun --time=$(t2sd) <srun args>
```

- If you usually use the `sbatch` command to submit batch jobs:

```
sbatch --time=$(t2sd) script.sbatch
```

Note that if you usually run your jobs on a partition with short time limits (e.g., debug or express), you only need to add the `$(t2sd)` option once it's closer to the start of the maintenance window. Use `$(t2sd)` only if the time remaining before the start of the maintenance period is less than the default time limit of the partition.

For instance, the default time limit for the express partition is 60 minutes. If you want to run a job on the express partition at 5 a.m. on June 1, you wouldn't need to add the `$(t2sd)` option. However, if you wanted to run a job at 7:30 a.m. on June 1 on the express partition, you would need to include the `$(t2sd)` option to account for the remaining time.

See also:

Partitions for more information about available partitions.

Moreover, we can help you set up a default and maximum time configuration on your partition. This configuration can significantly alleviate the issues you may experience with job runtime. By defining default and maximum time limits, you can establish a predefined window for job execution without explicitly specifying the runtime for each job.

However, note that even with the default and maximum time configuration in place, there will always be a time equal to the default time limit where explicitly specifying the job's runtime becomes helpful. This allows for better control and management of job scheduling within the available resources.

If you want to set up the default and maximum time configuration on your partition or have any concerns or questions regarding job runtime management, please let us know. We are here to assist you further.

Following these instructions ensures that your job scripts consider the maintenance period and set appropriate time limits. If you have any further questions, feel free to ask!

CHAPTER

FOUR

CONNECTING MAC

You connect to Discovery using a secure shell program to initiate an SSH session to sign in to Discovery. If you usually launch software from the command line that uses a graphical user interface (GUI), see *Using X11 on Mac* for tips and troubleshooting information.

4.1 Mac

Mac computers come with a Secure Shell (SSH) program called Terminal that you use to connect to the HPC using SSH. If you need to use software that uses a GUI, such as Matlab or Maestro, make sure to use the -Y option in the second step below (see *Using X11* for more tips and troubleshooting information).

Note: If you use Mac OS X version 10.8 or higher, and you have XQuartz running in the background to do X11 forwarding, you should execute the following command in Terminal once before connecting:

```
defaults write org.macosforge.xquartz.X11 enable_iglx -bool true
```

You should keep XQuartz running in the background. If you close and restart XQuartz, you will need to execute the above command again after restarting. Do not use the Terminal application from within XQuartz to sign in to Discovery. Use the default Terminal program that comes with your Mac (see Step 1 in the procedure below).

To connect to Discovery on a Mac:

1. Go to Finder > Applications > Utilities, and then double click Terminal.
2. At the prompt, type `ssh <username>@login.discovery.neu.edu`, where `<username>` is your Northeastern username. If you need to use X11 forwarding, type `ssh -Y <username>@login.discovery.neu.edu`.
3. Type your Northeastern password and press `Enter`.

You are now connected to Discovery at a login node.

Watch this video of how to connect to Discovery on a Mac. If you do not see any controls on the video, right-click on the video to see viewing options.

4.2 Using X11 on Mac

When you launch a software application that uses a graphical user interface (GUI) from the command line, this is completed through X11 forwarding. From a Mac Terminal log in using the -Y option (`ssh -Y <yourusername>@login.discovery.neu.edu`).

Tip: If you used the -Y option to enable X11 forwarding on your Mac, you can test to see if it is working by typing `xeyes`. This will run a small program that makes a pair of eyes appear to follow your cursor.

4.2.1 Passwordless ssh

You need to set up passwordless ssh to ensure that GUI-based applications will launch without any issues. You also need to make sure that your keys are added to the authorized.key file. This needs to be done anytime you regenerate your keys. If you're having an issue with opening an application that need X11 forwarding, such as MATLAB or Schrodinger, and you recently regenerated your keys, make sure to add your keys to the authorized.key file.

Note: Errors that you can see on both Mac and Windows when launching a GUI-based program include the following:

```
Error: unable to open display localhost:19.0
```

```
Launch failed: non-zero return code
```

If you are getting these types of errors, it could be because of following reasons:

1. You haven't set up passwordless SSH. If that's the case, you can follow the steps below to set up passwordless SSH.
2. When requesting a compute node from the login node, you may have forgotten to include the `--x11` option. In that case please see this example srun command for more details.

Setting up passwordless ssh:

Note: Make sure you're on your local computer for steps 1 through 4. If you are connected to the cluster, type `exit` to return to your local computer.

1. On a Mac, open Terminal and type `cd ~/.ssh`. This moves you to the ssh folder on your local computer.
2. Type `ssh-keygen -t rsa` to generate two files, `id_rsa` and `id_rsa.pub`.
3. Press `Enter` to all the prompts (do not generate a passphrase).
4. Type `ssh-copy-id -i ~/.ssh/id_rsa.pub <yourusername>@login.discovery.neu.edu` to copy `id_rsa.pub` to your `/home/.ssh` folder on Discovery. Enter your NU password if prompted. This copies the token from `id_rsa.pub` file to the `authorized_keys` file which will either be generated or appended if it already exists.
5. Connect to Discovery via `ssh <yourusername>@login.discovery.neu.edu`. You should now be connected without having to enter your password.

Note: If you are using a Windows machine using MobaXterm, sign in to the HPC as usual, then complete steps 6 through 9 to complete the passwordless ssh setup.

6. Type `cd ~/.ssh` to move to your ssh folder.
7. Type `ssh-keygen -t rsa` to generate your key files.
8. Press Enter to all the prompts (do not generate a passphrase). If prompted to overwrite a file, type `Y`.
9. Type `cat id_rsa.pub >> authorized_keys`. This adds the contents of your public key file to a new line in the `~/.ssh/authorized_keys` file.

Next Steps

After you are connected, you can run jobs either in interactive mode with `srun` or submit a script using `sbatch`. See *Interactive Jobs: srun Command* and *Batch Jobs: sbatch* for more information.

To load and run software, see *Software Overview*.

To find out more about the hardware and partitions on Discovery, see *Hardware Overview* and *Partitions*.

For our introductory training video, go to Northeastern's LinkedIn Learning page.

CHAPTER FIVE

CONNECTING WINDOWS

You connect to Discovery using a secure shell program to initiate an SSH session to sign in to Discovery. This topic is about how to connect using Windows. See *connect mac* for information about connecting with a Mac.

Before you can connect to Discovery on a Windows computer, you'll need to download a terminal program, such as MobaXterm or PuTTY. We recommend MobaXterm, as you can also use it for file transfer, whereas with other SSH programs, you would need a separate file transfer program.

To connect to Discovery with MobaXterm:

1. Open MobaXterm.
2. Click **Session**, then click **SSH** as the connection type.
3. In **Remote Host**, type `login.discovery.neu.edu`, make sure **Port** is set to 22, and click **OK**. (OPTIONAL: You can type your Northeastern username and password on MobaXterm, and it will save that information every time you sign in. If you opt to do this, you will be connected to Discovery after you click OK.)
4. At the prompt, type your Northeastern username and press Enter.
5. Type your Northeastern password and press Enter. Note that the cursor does not move as you type your password. This is expected behavior.

You are now connected to the cluster's login node.

Watch this video to see how to connect to the cluster with MobaXterm. If you do not see any controls on the video, right click on the video to see viewing options.

5.1 Using X11 on Windows

When you launch a software application that uses a graphical user interface (GUI) from the command line, this is completed through X11 forwarding. If you use MobaXterm on Windows, X11 forwarding is turned on by default.

Tip: You can test to see if x11 forwarding is working by typing `xeyes`. This will run a small program that makes a pair of eyes appear to follow your cursor.

5.2 Passwordless ssh on Windows

You need to set up passwordless ssh to ensure that GUI-based applications will launch without any issues. You also need to make sure that your keys are added to the authorized.key file. This needs to be done anytime you regenerate your keys. If you're having an issue with opening an application that need X11 forwarding, such as MATLAB or Schrodinger, and you recently regenerated your keys, make sure to add your keys to the authorized.key file.

Note: Errors that you can see on Windows when launching a GUI-based program include the following:

```
Error:  unable to open display localhost:19.0
```

```
Launch failed:  non-zero return code
```

If you are getting these types of errors, it could be because of following reasons:

1. You haven't set up passwordless SSH. If that's the case, you can follow the steps below to set up passwordless SSH.
2. When requesting a compute node from the login node, you may have forgotten to include the `--x11` option. In that case please see this example srun command for more details.

To set up passwordless ssh:

1. Sign in to the cluster using MobaXterm.
2. Type `cd ~/.ssh` to move to your ssh folder.
3. Type `ssh-keygen -t rsa` to generate your key files.
4. Press `Enter` to all the prompts (do not generate a passphrase). If prompted to overwrite a file, type `Y`.
5. Type `cat id_rsa.pub >> authorized_keys`. This adds the contents of your public key file to a new line in the `~/.ssh/authorized_keys`.

Next Steps

After you are connected, you can run jobs either in interactive mode with `srun` or submit a script using `sbatch`. See *Interactive Jobs: srun Command* and *Batch Jobs: sbatch* for more information.

To load and run software, see *Software Overview*.

To find out more about the hardware and partitions on Discovery, see *Hardware Overview* and *Partitions*.

For our introductory training video, go to Northeastern's LinkedIn Learning page.

CHAPTER

SIX

ACCESSING OPEN ONDEMAND

Open OnDemand (OOD) is a web portal to the Discovery cluster.

This topic is for connecting to the Discovery cluster through the browser application, Open OnDemand. If you are looking to access Discovery directly on your system rather than through a browser, please see *Connecting Mac* for connecting with a Mac and *Connecting Windows* for connecting with Windows.

A Discovery account is necessary for you to access OOD. If you need an account, see *Getting Access*. After you have created a Discovery account, proceed with the following steps in order to access Discovery.

To Access Discovery through Open OnDemand (OOD), do the following

1. In a web browser, go to http://ood.discovery.neu.edu.
2. At the prompt, enter your Northeastern username and password. Note that your username is the first part of your email without the @northeastern, such as **j.smith**.
3. Press **Enter** or click **Sign in**.

Watch the following video for a short tutorial. If you do not see any controls on the video, right-click on the video to see viewing options.

6.1 OOD: next steps

After you are connected, you use the interactive apps in OOD. See *OOD Interactive Apps* for more information.

CHAPTER

SEVEN

SHELL ENVIRONMENT ON THE CLUSTER

7.1 The Discovery Shell environment and .bashrc

Discovery uses a Linux-based Operating System (CentOS), where the Shell program is used to interface with the user. Bash (Bourne Again SHell) is one of the most popular Shell implementations which is the default Shell on Discovery.

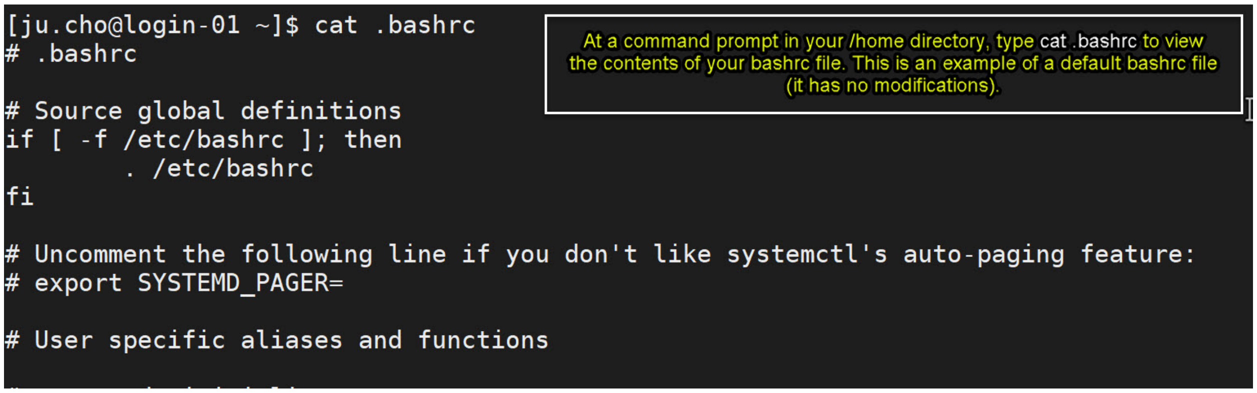
The Shell script `.bashrc` is used by Bash to initialize your Shell environment. For example, it is typically used to define aliases, functions and load modules. Note that environment variables settings (such as `PATH`) generally go in the `.bash_profile` or `.profile` files. Your `.bashrc`, `.bash_profile` and `.profile` files live in your `$HOME` directory. You can make changes to your .bashrc with a text editor like nano.

Caution: Making edits to your `.bashrc` file can result in many issues. Some changes may prevent you from launching apps or executing commands. Modifying your `PATH` variable may result in the inability to use basic Shell commands (such as `cd` or `ls`) if not done correctly.

Before making changes to your `.bashrc` file, make a backup of the default `.bashrc` file, so you can restore it if necessary. If you need help with editing `.bashrc`, reach out to mailto:rchelp@northeastern.edu or schedule a consultation with a staff member who can help suggest edits and troubleshoot any issues you might be having.

7.2 About your .bashrc file

You have a default `.bashrc` file in your home directory when your account is created. See the figure below for an example of a default `.bashrc` file.



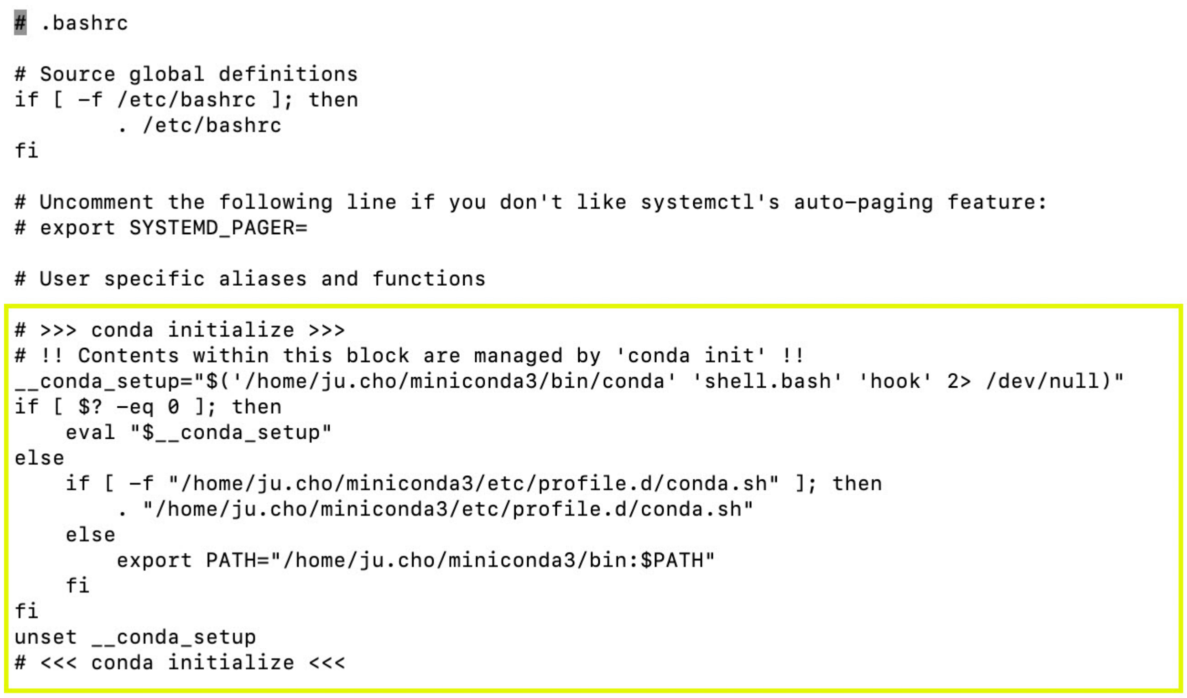
```
[ju.cho@login-01 ~]$ cat .bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
```

At a command prompt in your /home directory, type cat .bashrc to view the contents of your bashrc file. This is an example of a default bashrc file (it has no modifications).

Important: We recommend to keep .bashrc unchanged when using Discovery. You can source environment Shell scripts or load modules directly inside your job instead. This approach can prevent some runtime errors from loading incompatible modules, setting environment variables incorrectly, or from mixing multiple software and Conda environments.

7.3 Conda and `.bashrc`

In addition to editing your `.bashrc` file as outlined in the example above, programs that you install can also modify your .bashrc file. For example, if you follow the procedure outlined in *Using Miniconda*, there may be a section added to your `.bashrc` file (if you didn't use the `-b` batch option) that automatically loads your conda environment every time you sign in to Discovery. See the figure below for an example of this:



```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/home/ju.cho/miniconda3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/home/ju.cho/miniconda3/etc/profile.d/conda.sh" ]; then
        . "/home/ju.cho/miniconda3/etc/profile.d/conda.sh"
    else
        export PATH="/home/ju.cho/miniconda3/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<
```

You should not modify this section in the `.bashrc` file directly. If it was changed, remove this section manually using a file editor.

Caution: We recommend removing the conda initialization section from your `.bashrc` as it may interfere with the correct startup environment when using Open OnDemand apps. You should always load your Conda environment after your job already started.

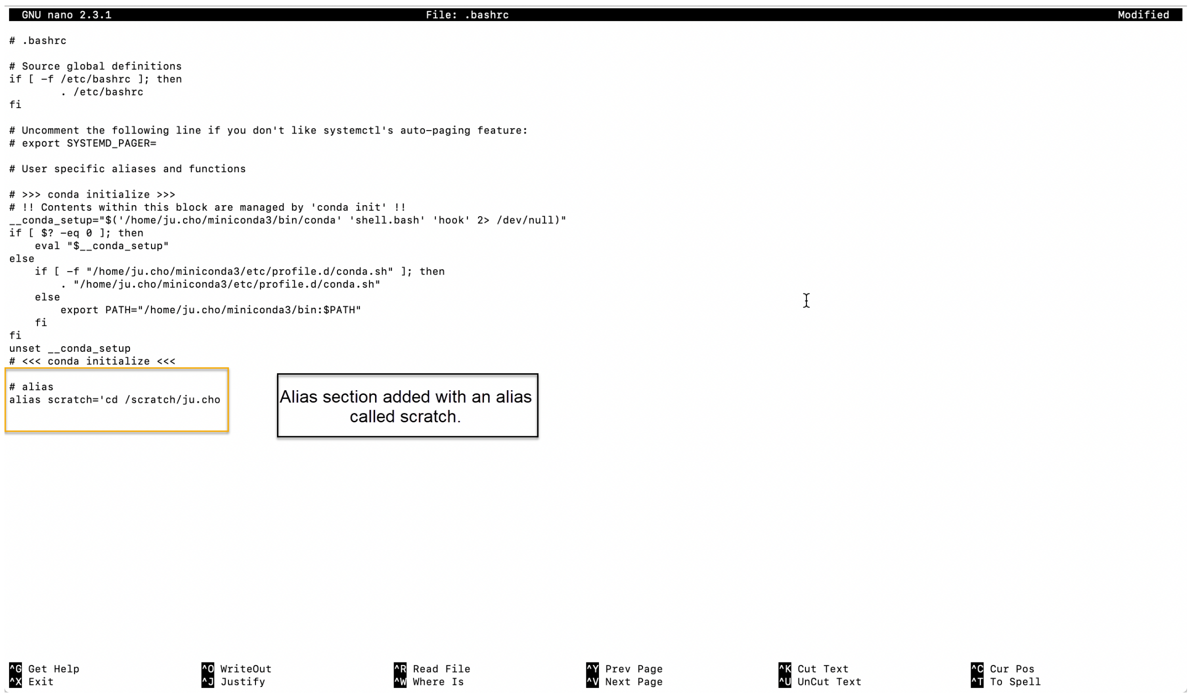
If you need help with your .bashrc file or would like it restored to its default, reach out to the RC team at mailto: rchelp@northeastern.edu, and we can provide you with a new, default `.bashrc` file and help troubleshoot issues with the file.

7.3.1 Editing your `.bashrc` file

The basic workflow for editing your `.bashrc` file is to sign in to Discovery, go to your `$HOME` directory, open the file in a text editor on the command line, make your edits, save the file, sign out of Discovery, then sign back in again. Your changes will take effect when you have signed back in again.

Example procedure for editing your `.bashrc` file:

1. Sign in to Discovery.
2. (Optional) Type `pwd` to make sure you are in your `/home` directory.
3. (Optional) Type `ls -a` to view the contents of your `/home` directory, including hidden files. Your .bashrc file is a hidden file (hidden files are preceded by a `.`). Using the `-a` option with `ls` displays hidden files.
4. (Recommended) Type `cp .bashrc .bashrc-default` to make a copy of your .bashrc file called `.bashrc-default`.
5. Type `nano .bashrc` to open your .bashrc file in the nano text editor.
6. Type the edits that you want to make to your file. In this example, an alias was added to create a shortcut to the user's `/scratch` space.



```
  GNU nano 2.3.1                        File: .bashrc                                        Modified

# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions

# >>> conda initialize >>>
# !! Contents within this block are managed by 'conda init' !!
__conda_setup="$('/home/ju.cho/miniconda3/bin/conda' 'shell.bash' 'hook' 2> /dev/null)"
if [ $? -eq 0 ]; then
    eval "$__conda_setup"
else
    if [ -f "/home/ju.cho/miniconda3/etc/profile.d/conda.sh" ]; then
        . "/home/ju.cho/miniconda3/etc/profile.d/conda.sh"
    else
        export PATH="/home/ju.cho/miniconda3/bin:$PATH"
    fi
fi
unset __conda_setup
# <<< conda initialize <<<

# alias
alias scratch='cd /scratch/ju.cho
```

7. Save the file and exit the editor.
8. Sign out of Discovery and sign back in for the changes to take effect.

7.3.2 Sourcing a Shell script example

A safe alternative to using `.bashrc` is to source a Shell script inside your runtime job environment. Below is an example script to load an Anaconda module and source a Conda environment which will then be used inside the Slurm script.

Create a Shell script `myenv.bash`:

```
#!/bin/bash
module load anaconda3/2021.05
module load cuda/11.1
source activate pytorch_env_training
```

Then, source the Shell script inside your sbatch Slurm script (see *Batch Jobs: sbatch*):

```
#!/bin/bash
#SBATCH --partition=gpu
#SBATCH --nodes=1
#SBATCH --gres=gpu:1
#SBATCH --time=01:00:00
#SBATCH --job-name=gpu_run
#SBATCH --mem=4GB
#SBATCH --ntasks=1
#SBATCH --output=myjob.%j.out
#SBATCH --error=myjob.%j.err

source myenv.bash
python <myprogram>
```

CHAPTER

EIGHT

HARDWARE OVERVIEW

The Discovery computing cluster provides you with access to over 1024 CPU nodes, 50,000 CPU cores, and over 200 GPUs and is connected to the university network over 10 Gbps Ethernet (GbE) for high-speed data transfer. Compute nodes are connected to each other with either 10 GbE or a high-performance HDR200 InfiniBand (IB) interconnect running at 200 Gbps (with some nodes running HDR100 IB, if HDR200 IB is not supported on those nodes).

8.1 CPU nodes

Table 1 below shows the feature names, number of nodes by partition type (public and private), and the RAM memory range per node. The feature name follows archspec microachitechture specification.

Feature Name	Number of Nodes - public, private	RAM memory per node
skylake	0, 170	186 - 3094 GB
zen2	40, 292	256 - 2000 GB
zen	40, 300	256 - 2000 GB
ivybridge	64, 130	31 - 1031 GB
sandybridge	8, 0	384 GB
haswell	230, 62	109 - 1031 GB
broadwell	756, 226	128 - 515 GB
cascadelake	260, 88	186 - 3094 GB

If you are looking for information about GPUs, see *Working with GPUs*.

If you are interested in more information about the different partitions on Discovery, including the number of nodes per partition, running time limits, job submission limits, and RAM limits, see *Partitions*.

8.2 Using the `--constraint` flag

When using `srun` or `sbatch`, you can specify hardware features as part of your job by using the `--constraint=` flag. This may be particularly useful when benchmarking, optimizing, or if you're using code that was compiled on a certain micro-architecture. Currently, you can use the `--constraint=` flag to restrict your job to a specific feature name (e.g., `haswell`, `ivybridge`) or you can use the flag: `ib` to only include nodes that are connected by InfiniBand (IB) with a job that needs to use multiple nodes.

A few examples using `srun`:

```
    srun --constraint=haswell --pty /bin/bash
    srun --constraint=ivybridge --pty /bin/bash
```

(continues on next page)

(continued from previous page)

```
    srun --constraint=ib --pty /bin/bash
    srun --constraint="[ivybridge|zen2]" --pty /bin/bash #this uses the OR operator | to select either an ivybridge or zen2 node.
```

You can add these same flags as an additional line in your `sbatch` script via (`#SBATCH --constraint=haswell`)

Note: Using the –constraint flag can mean that you will wait longer for your job to start, as the scheduler (Slurm) will need to find and allocate the appropriate hardware that you have specified for your job. For more information about running jobs, see *Using Slurm*. Finally, at this time only the OR operator `|` is supported when using `--constraint`.

CHAPTER

NINE

PARTITIONS

9.1 Introduction

Partitions are logical collections of nodes that comprise different hardware resources and limits to help meet the wide variety of jobs that get scheduled on Discovery. Occasionally, the Research Computing team might need to make updates to the partitions based on monitoring job submissions to help reduce job wait times. As our cluster grows, changes to the partitions also help to ensure the fair, efficient distribution of resources for all jobs being submitted to the cluster.

On Discovery, there are several partitions:

- General access (`debug`, `express`, `short`, `gpu`)
- Application only (`long`, `large`, `multigpu`)
- PI owned (accessed only by members of the PIs' group)

The general access and application only partitions span the hardware on Discovery, with `gpu` and `multigpu` spanning the GPUs on Discovery and the other partitions spanning the CPUs. For example, if you use the `debug` partition you're using the same hardware as `short`, just with different time, job, and core limits. Refer to the tables below for detailed information on the current partitions. Note that PI-owned partitions only include the hardware that those PIs own and are only accessible to the members of the PI's group.

Note: In the following table, the Running Jobs Per User/Per Research Group. Core and RAM limits are set per user, across all running jobs (not pending). **Keep in mind that the number of running jobs is limited by the available resources on the cluster at the time of the job submission and may not adhere to the number stated below.**

Name	Requires approval?	Time limit (default/max)	Running jobs	Submitted jobs	Core limit (per user)	RAM limit	Use Case
debug	No	20 minutes/20 minutes	10/25	5000	128	256GB	Best for serial and parallel jobs that can run under 20 minutes. Good for testing code.
express	No	30 minutes/60 minutes	50/250	5000	2048	25TB	Best for serial and parallel jobs that can run under 60 minutes.
short	No	4 hours/24 Hours	50/500	5000	1024	25TB	Best for serial or small parallel jobs (`--nodes=2` max) that need to run for up to 24 hours.
long	**Yes**	1 day/5 Days	25/250	1000 per user/5000 per group	1024	25TB	Primarily for serial or parallel jobs that need to run for more than 24 hours. Need to prove that your code cannot checkpoint to use this partition.
large	**Yes**	6 hours/6 Hours	100/100	1000 per user/5000 per group	N/A	N/A	Primarily for running parallel jobs that can efficiently use more than 2 nodes. Need to demonstrate that your code is optimized for running on more than 2 nodes.

Name	Requires approval?	Time limit (default/max)	Running jobs	Submitted jobs	GPU limit	GPU limit	Use Case
gpu	No	4 hours/8 Hours	25/250	50/100	1	8	For jobs that can run on a single GPU processor.
multigpu	**Yes**	4 hours/24 Hours	25/100	50/100	12	12	For jobs that require more than one GPU and take up to 24 hours to run.

9.2 Viewing partition information

Slurm commands allow you to view information about the partitions. Three commands that can show you partition information are `sinfo`, `sacct`, and `scontrol`. The following are common options to use with these commands:

```
sinfo -p <partition name> #displays the state of the nodes on a specific partition
sinfo -p <partition name> --Format=time,nodes,cpus,socketcorethread,memory,nodeai,
↪features #displays more detailed information using the Format option, including␣
↪features like the type of processors
sacct --partition <partition name> #displays the jobs that have been run on this␣
↪partition
scontrol show partition <partition name> #displays the Slurm configuration of the␣
↪partition
```

For more information about these commands, see the Slurm documentation.

9.3 Allocating partitions in your jobs

To specify a partition when running jobs, use the option `--partition=<partition name>` with either `srun` or `sbatch`. When using a partition with your job and specifying the options of `--nodes=` and `--ntasks=`, make sure that you are requesting options that best fit your job. It can actually have the opposite effect on jobs that are better suited to running with smaller requirements, as you have to wait for the extra resources that your job will not use. See *Using Slurm* for more information on using Slurm to run jobs.

Note: Requesting the maximum number of nodes or tasks will not make your job run faster or give you higher priority in the job queue.

Tip: You should always try to have job requests that will attempt to allocate the best resources for the job you want to run. For example, if you are running a job that is not parallelized, you only need to request one node (`--nodes=1`). For some parallel jobs, such as a small MPI job, you can also use one node (`--nodes=1`) with the `--ntasks=` option set to correspond to the number of MPI ranks (tasks) in your code. For example, for a job that has 12 MPI ranks, request 1 node and 12 tasks within that node (`--nodes=1 --ntasks=12`). If you request 12 nodes, Slurm is going to run code between those nodes, which could slow your job down significantly if it isn't optimized to run between nodes.

If your code is optimized to run on more than two nodes and needs less than one hour to run, you can use the express partition. If your code needs to run on more than 2 nodes for more than one hour, you should apply to use the large partition. See the section Partition Access Request below for more information.

9.4 Partition Access Request

If you need access to the `large`, `long`, or `multigpu` partition, you need to submit a ServiceNow ticket. Access is not automatically granted. You will need to provide details and test results that demonstrate your need for access for these partitions. If you need temporary access to multigpu to perform testing before applying for permanent access, you should also submit a ServiceNow ticket. All requests are evaluated by members of the RC team, and `multigpu` requests are also evaluated by two faculty members.

CHAPTER

TEN

SOFTWARE OVERVIEW

Discovery offers you many options for working with software. Two of the easiest and most convenient ways are using the `module` command on the command line and using the interactive apps on Open OnDemand (OOD), Discovery's web portal. If you need a specific software package, first check to see if it is already available through one of the preinstalled modules on Discovery. The Research Computing team adds new modules regularly, so use the `module avail` command to view the most up-to-date list. You can also try using Spack, a software package manager available on Discovery. Spack has over 5,000 packages that you can install.

Using Module For more information about working with module.

OOD Interactive Apps For more information about OOD.

Using Spack For more information about Spack.

You can also use Conda, Miniconda, and Anaconda to manage software packages. See *Using Conda* for more information.

10.1 Using Make

If you want to use `make` to add software locally to you path you must first download the software package from its source (such as a webpage or GitHub), and you need to unpack it or unzip it, if need be. Then, you must set the installation path to a directory where you have write access on Discovery, such as your home directory. You can use `./configure` to do this, such as `./configure --prefix=/home/<yourusername>/software` After you have set the installation path, you need to compile the code using `make` and then install the software using `make install`.

10.2 Requesting Software Installation Assistance

If the software that you need is not a module on Discovery, cannot be installed through Spack, or is not available through another way of self-installation (such as using `make`), you can submit a ServiceNow software request ticket. Be aware that there might be packages that cannot be installed on Discovery due to incompatibility with the hardware on Discovery.

CHAPTER

ELEVEN

USING MODULE

The module system on Discovery includes many commonly used scientific software packages that you can load in your path when you need it and unload it when you no longer need it.

Use the `module avail` command to show a list of the most currently available software on Discovery.

Note: Some modules might conflict with each other, resulting in the software not behaving as expected. Also, if there are multiple versions of software, and you load more than one version of the software, only the latest version will be used. Use `module list` to view the modules you currently have loaded in your path.

Tip: Use the `which` command to display which version of a software package you have in your path. For example, `which python` will display the version of python that you have in your path.

11.1 Module commands

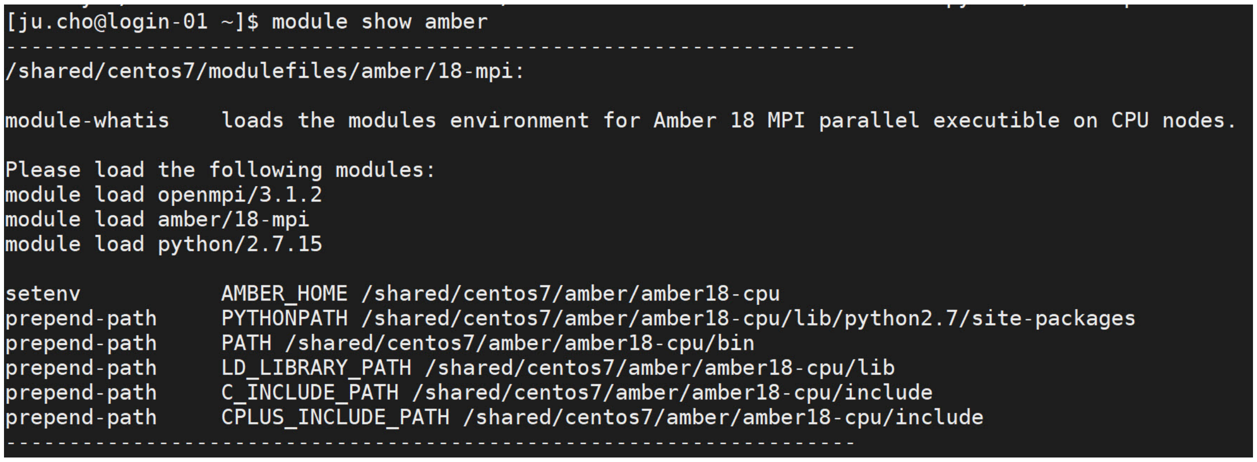
The following are common module commands that are useful for interacting with software packages on Discovery.

Module Command	Function
`module avail`	View a list of all of the available software packages on Discovery that you can load
`module list`	Displays a list of the software packages currently loaded in your path
`module show <module name>`	View the details of a software package (see the section "Module Show" below for more information)
`module load <module name>`	Load a software package into your environment
`module unload <module name>`	Remove a single software package from your environment
`module purge`	Removes all of the loaded software packages from your environment.

Caution: Using `module purge` will purge all modules from your environment, including the default module `discovery/2019-02-21`. This module contains the http proxy needed for nodes to have internet access. If you accidentally purge this module, it will be automatically reloaded the next time you log out and log back in again. You can also load it manually if you have purged it by using the `module load` command.

11.2 Module show example

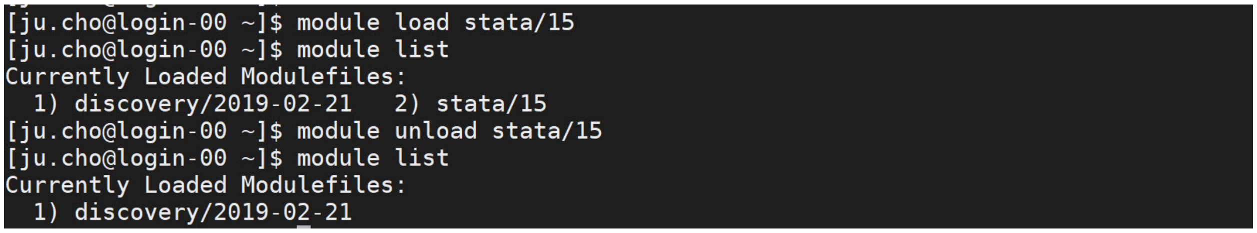
Before loading a module, type `module show <name of module>` to see if there are any dependencies or commands that you need to execute before loading the module. In some cases, a module might depend on having other modules loaded to work as expected. While modules are a convenient way of loading software to use on Discovery, scientific software can come with many packages and dependencies. In addition to module, you should review other ways of loading software on Discovery. See *Software Overview* for more information on different ways you can install software on Discovery. The figure below shows an example of `module show` with the software package called amber.



```
[ju.cho@login-01 ~]$ module show amber
-------------------------------------------------------------------
/shared/centos7/modulefiles/amber/18-mpi:

module-whatis    loads the modules environment for Amber 18 MPI parallel executible on CPU nodes.

Please load the following modules:
module load openmpi/3.1.2
module load amber/18-mpi
module load python/2.7.15

setenv           AMBER_HOME /shared/centos7/amber/amber18-cpu
prepend-path     PYTHONPATH /shared/centos7/amber/amber18-cpu/lib/python2.7/site-packages
prepend-path     PATH /shared/centos7/amber/amber18-cpu/bin
prepend-path     LD_LIBRARY_PATH /shared/centos7/amber/amber18-cpu/lib
prepend-path     C_INCLUDE_PATH /shared/centos7/amber/amber18-cpu/include
prepend-path     CPLUS_INCLUDE_PATH /shared/centos7/amber/amber18-cpu/include
-------------------------------------------------------------------
```

11.3 Module load and unload example

In the figure below, the software module stata/15 was loaded and then unloaded. After loading and unloading, module list was used to check that the STATA was loaded and unloaded.



```
[ju.cho@login-00 ~]$ module load stata/15
[ju.cho@login-00 ~]$ module list
Currently Loaded Modulefiles:
  1) discovery/2019-02-21   2) stata/15
[ju.cho@login-00 ~]$ module unload stata/15
[ju.cho@login-00 ~]$ module list
Currently Loaded Modulefiles:
  1) discovery/2019-02-21
```

11.4 Using software applications with X11 Forwarding

If you are attempting to open a GUI-based software application that uses X11 forwarding to display, such as MATLAB or Maestro, and you get an error such as `Error: unable to open display localhost:19.0`, this is most likely due to an issue with passwordless SSH. See *Using X11 on Mac* for tips and troubleshooting information opening applications that use X11 forwarding.

CHAPTER

TWELVE

USING MATLAB

MATLAB is available as a module on Discovery (see *Using Module* for more information), and it is also an interactive app on Open onDemand (see *Introduction to OOD* for more information). You can also download MATLAB for use with your personal computer through the Northeastern portal on the MATLAB website. Note that the procedures detailed below are specific to using MATLAB on Discovery and not with using MATLAB on your personal computer.

12.1 Installing MATLAB toolboxes

Use the following procedure if you need to install a MATLAB toolbox:

1. Download the toolbox from its source website.
2. Connect to Discovery.
3. Create a directory in your /home directory. We recommend creating a directory called `matlab` by typing:

```
mkdir /home/<username>/matlab  #where <username> is your username
```

4. Go to the directory you just created by typing:

```
cd /home/<username>/matlab
```

5. Unzip the toolbox file by typing:

```
unzip <toolboxname>
```

6. Load MATLAB by typing:

```
module load matlab
```

7. Start MATLAB by typing:

```
matlab
```

8. Add the toolbox to your PATH by typing:

```
addpath('/home/<username>/matlab/<toolbox>') #where <toolbox> is the␣
↪toolbox you just unzipped
```

9. If this is a toolbox you want to use more than once, you should save it to your path by typing:

```
savepath()
```

10. You can now use the toolbox within MATLAB. When you are done, type `quit`.

12.2 Using MATLAB Parallel Server

The Discovery cluster has MATLAB Parallel Server installed. This section details an example of how you can setup and use the MATLAB Parallel Computing Toolbox. This walkthrough uses MATLAB 2020a launched as an interactive app on the Open onDemand web portal. There are several parts to this walkthrough. We suggest that you read it through completely before starting. The parameters presented represent only one scenario.

This walkthrough will use Open onDemand, the web portal on Discovery, to launch MATLAB. You'll then create a cluster profile. This allows you to define cluster properties that will be applied to your jobs. Supported functions are *batch, parpool, and parcluster*. The Parallel Computing Toolbox comes with a cluster profile called *local*, which you will change in the walkthrough below.

Note: This walkthrough details submitting jobs through Discovery's Open onDemand web portal. Some parameters will vary if you are using MATLAB from the command line. This walkthrough does not apply to other versions of MATLAB.

Before starting, you should create a folder in your `/scratch/<yourusername>` directory. This folder is where you'll save your job data.

1. Go to your /scratch directory: `cd /scratch/<yourusername>` where `<yourusername>` is your NU username
2. Make a new folder. We suggest calling it *matlab-metadata*: `mkdir matlab-metadata`

To start MATLAB and add a Cluster Profile, do the following:

1. Go to http://ood.discovery.neu.edu. If prompted, sign in with your Discovery username and password.
2. Click **Interactive Apps**, and select **MATLAB**.
3. Select **MATLAB version 2020a**, and keep the default time of 1 hour and default memory of 2GB. Click **Launch**.
4. If necessary, adjust the **Compression** and **Image Quality**, and then click **Launch MATLAB**.
5. On the MATLAB Home tab, in the **Environment** section, select **Parallel**, then click **Create and Manage Clusters**. This opens the Cluster Profile Manager window.
6. On the Cluster Profile Manager window, select **Add Cluster Profile**, then click **Slurm**. If prompted, click **OK** to the notice about needing Parallel Server.
7. Double click the new profile name in the Cluster Profile column, and type a name such as **TestProfile**. Press **Enter** to save the change.
8. Select **Edit** in the **Manage Profile** section. This lets you edit the options on the **Properties** tab. For this walkthrough, make the following edits:
    1. In the **Folder where job data is stored on the client** option, type `/scratch/<yourusername>/matlab-metadata` (this is the directory that you created in the first procedure above).
    2. In the **Number of workers available to cluster** option, type a number between between 1 and 10. This field is the number of MPI processes you intend to run. This corresponds to the `--ntasks` Slurm option. The maximum is 128 per job; however, for this task, we recommend keeping it lower and use threading inside the nodes. The number you set here will be the default maximum for the job. You can set it for less than or equal to this number in the MATLAB Command Window when submitting your job.
    3. In the **Number of computational threads to use on each worker** option, type a number between 1 and 10. This field represents the number of threads that each worker will possess. This corresponds to `cpus-per-task` in Slurm. Do not exceed the number of available cores on the node.
9. When you have finished editing your properties, click **Done**.

10. (Optional) If you want to validate your setup, click the **Validation** tab (next to the Properties tab). Ensure all the stages are checked, then click the **Validate** button at the bottom of the page.

This will check the properties of your profile. You might need to wait a minute or two for this to complete.

Caution: Do not click the green **Validate** button. This will attempt validation using the maximum number of workers, which can cause the validation to hang or fail. If you accidentally click the green Validate button, click **Stop** to end the validation process.

(OPTIONAL) In the **Cluster Profile** column, right-click on the TestProfile name and select **Set as Default**. This sets your profile to be the default.

Now that you have set up your profile, you can use the default cluster profile you just created (*TestProfile*) with the following commands:

```
#with parpool
parallel.defaultClusterProfile(‘TestProfile’)
parpool

#with parcluster
c = parcluster(‘TestProfile’)
```

12.2.1 Using parcluster example

This section will detail how to submit batch jobs to the cluster to perform scaling calculations for an integer factorization sample problem. It’s a computationally intensive problem, where the complexity of the factorization increases with the magnitude of the number. We’ll use the myParallelAlgorithmFcn.m MATLAB function. This section assumes you have configured a MATLAB Cluster Profile according to the procedure above.

On Discovery, there are benchmarking scripts and examples located in the `/shared/centos7/matlab/R2020a/examples/parallel/main` folder. To add the path to this folder to the list of available paths, do one of the following:

- On the MATLAB Home tab, in the **Environment** section, click **Set Path** and add the path to the script.
- Alternatively, provide the full path of the script in the MATLAB command line.

The contents of myParallelAlgorithmFcn are as follows:

```
function [numWorkers,time] = myParallelAlgorithmFcn ()

complexities =  [2^18 2^20 2^21 2^22];
numWorkers = [1 2 4 6 16 32 64];

time = zeros(numel(numWorkers),numel(complexities));

% To obtain predictable sequences of composite numbers, fix the seed
% of the random number generator.
rng(0,'twister');

for c = 1:numel(complexities)

   primeNumbers = primes(complexities(c));
   compositeNumbers =    primeNumbers.*primeNumbers(randperm(numel(primeNumbers)));
   factors = zeros(numel(primeNumbers),2);
```

(continues on next page)

(continued from previous page)

```
   for w = 1:numel(numWorkers)
       tic;
       parfor (idx = 1:numel(compositeNumbers), numWorkers(w))
          factors(idx,:) = factor(compositeNumbers(idx));
       end
       time(w,c) = toc;
   end
end
```

To submit myParallelAlgorithmFcn as a batch job, in the MATLAB Command Window, type:

```
totalNumberOfWorkers = 65;
cluster = parcluster('TestProfile');
job = batch(cluster,'myParallelAlgorithmFcn',2,'Pool',totalNumberOfWorkers-1,
↪'CurrentFolder','.');
```

This specifies the `totalNumberOfWorkers` as 65, where 64 workers will be issued to run *parfor* in parallel (so the pool is set as 64), and the additional worker will run the main process.

To monitor the job after you submit it, click **Parallel**, then **Monitor Jobs** to open the Job Monitor. You can view some job information, such as the state of the job (i.e. running, failed, finished etc.), as well as the ability to fetch outputs if you right-click on the job line.

You can close MATLAB after you submit the job the scheduler. The job monitor tool will keep track of the jobs.

If you want to block MATLAB until the jobs are finished, type `Wait(job)`.

When the jobs complete, you can transfer the outputs of the function using the `fetchOutputs` command:

```
outputs = fetchOutputs(job);
numWorkers = outputs{1};
time = outputs{2};
```

You can plot the performance (speedup) by typing:

```
figure
speedup = time(1,:)./time;
plot(numWorkers,speedup);
legend('Problem complexity 1','Problem complexity 2','Problem complexity 3','Problem␣
↪complexity 4','Location','northwest');
title('Speedup vs complexity');
xlabel('Number of workers');
xticks(numWorkers(2:end));
ylabel('Speedup');
```

CHAPTER

THIRTEEN

USING CONDA

Conda is an open source environment and package manager. Miniconda is a free installer for Conda, Python, and comes with a few other packages. Anaconda is also a package manager that has a much larger number of packages pre-installed.

A question that frequently comes up is "Should I use Anaconda or Miniconda?"

Note: It is not recommended to build your Miniconda and Conda virtual environments inside your /home directory due to its limited space quota (see *Storage Accessible on Discovery*). Use the /work file system instead. If your group needs access to /work, the group PI can request it using: New Storage Space request.

13.1 Creating an Environment

Using a locally installed Conda virtual environment is highly recommended so that you can install the specific packages that you need. You can also have more than one environment with different packages for different research projects or for testing purposes. This procedure uses the Anaconda module already available on Discovery.

If you are on a login node, move to a compute node by typing:

Listing 1: Requesting 1 node with 1 cpu core and load anaconda.

```
srun --partition=short --nodes=1 --cpus-per-task=1 --pty /bin/bash
module load anaconda3/2022.05
```

To create a new Conda environment where `<environment-name>` is the path and name. You can see a list of your existing environments with `conda env list`.

```
conda create --prefix=/<path>/<environment-name> python=3.11 anaconda
```

Follow the prompts to complete the Conda install, then activate the environment.

```
source activate /<path>/<environment-name>
```

Your command line prompt will then include the path and name of environment.

```
(/<path>/<environment-name>) [username@c2001 dirname]$
```

Tip: `conda config --set env_prompt '({name}) '` modifies your `.condarc` to only show the environments name as such:

```
(<environment-name>) [username@c2000 dirname]$
```

With your Conda environment activated you can install a specific package with

```
conda install [packagename]
```

To deactivate the current active Conda environment

```
conda deactivate
```

To delete a Conda environment and all of its related packages, run:

```
conda remove -n yourenvironmentname --all
```

13.2 Using Miniconda

This procedure assumes that you have not installed Miniconda. If you need to update Miniconda, do not follow the installation procedure. Use `conda update`. This procedure uses the Miniconda3 version with Python version 3.8 in step 2, although there are other versions you can install (e.g., 3.9 or 3.11).

13.2.1 Installing Miniconda

Attention: Make sure to log on to a compute node.

```
srun --partition=short --nodes=1 --cpus-per-task=1 --pty /bin/bash
```

Download Miniconda, check the hash key, and install as follows:

```
wget --quiet https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
sha256sum Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh -b -p <dir>
```

Where <dir> is the full path to your desired installation directory (e.g., `/work/mygroup/mydirectory/miniconda3`).

Activate the base Miniconda environment

```
source <dir>/bin/activate
```

You can now create a new environment with this command where we're using python version 3.8:

```
conda create --name my-py38env python=3.8
```

Type `y` if asked to proceed with the installation.

Now you can activate your new environment

```
conda activate my-py38env
```

To deactivate the environment, type `conda deactivate`. You can type this command again to deactivate the base Miniconda environment.

13.3 Conda Best Practices

See also:

Best practices for home storage: *Conda*.

1. Your `~/.conda` may get very large if you install multiple packages and create many virtual Conda environments. Make sure to clean the Conda cache and clean unused packages with: `conda clean --all`.
2. Clean unused Conda environments by first listing the environments with: `conda env list` , and then removing unused ones: `conda env remove --name <environment-name>`.
3. You can build Conda environments in different locations to save space on your home directory (see *Storage Accessible on Discovery*). You can use the `--prefix` flag when building your environment. For example: `conda create myenv --prefix=/work/<mygroup>/<mydirectory>`.
4. Another recommended step is to update your Conda version (possible only when using Miniconda): `conda update conda -y`

CHAPTER

FOURTEEN

USING SPACK

Research Computing recommends using Spack to conveniently install software packages locally to your path. Please refer to the Spack documentation for the latest information about the packages that Spack contains. To use Spack, you first need to copy it to your `/home` directory or a `/work` directory, then you need to add it to your local environment.

Note: Spack software installations are part of your research and should preferably be stored in your PI's `/work` directory.

14.1 Install Spack

These instructions will demonstrate how to install Spack in your `/home` (*non-shared*) or `/work` (*shared*) directory and then how to add Spack to your local environment while on a compute node, so you have access to the Spack commands (steps 4-5).

Non-shared

Copy Spack's Git repository to '$HOME'

```
git clone -c feature.manyFiles=true https://github.com/spack/spack.git
```

Shared

Copy Spack's Git repository to '/work' and modify directory permissions to give write access to the members of your PI's `/work`.

```
cd /work/<PI-Project-Dir>
git clone -c feature.manyFiles=true https://github.com/spack/spack.git
chmod -R 775 spack/
```

14.2 Install a software using Spack

1. Request a compute node interactively: `srun -p short --pty -N 1 -n 28 /bin/bash`. While building the software Spack will attempt to run `make` in parallel. Hence, you need to request a compute node with multiple cores. This `srun` request is for 28 cores on one node (`-N 1 -n 28`).
2. Any module that is required for your software installation needs to be in your `$PATH` prior to adding Spack to your local environment. For example, to use a newer version of python for compatibility with Spack, type: `module load python/3.8.1`.
3. Add Spack to your local environment so you can use the Spack commands. If Spack has been installed on `$HOME`:

```
For Spack on $HOME
export SPACK_ROOT=/home/<yourusername>/spack
. $SPACK_ROOT/share/spack/setup-env.sh

For Spack on /work/<PI-Project-Dir>
export SPACK_ROOT=/work/<PI-Project-Dir>/spack
. $SPACK_ROOT/share/spack/setup-env.sh
```

4. After you have the Spack commands in your environment, type `spack help` to ensure Spack is loaded in your environment and to see the commands you can use with Spack. You can also type `spack list` to see all the software that you can install with Spack, but note this command can take a few moments to populate the list.
5. To check your spack version: `spack --version` .
6. To see information about a specific software package, including options and dependencies: `spack info <software name>`. Make sure to note the options and/or dependencies that you want to add or not add before installing the software.
7. To install a software package plus any dependencies or options: `spack install <software name> +<any dependencies or options>`; you can specify `-<any dependencies or options>`. You can also list + or - different options and dependencies within the same line. Do not put a space between each option/dependency that you list.
8. To view information about your installed software packages: `spack find <software package name>` or `spack info <software package name>` .
9. To Install a specific version of the software: `spack install <softwarename@version>`.

When you have installed a software package, you can add it to the module system by executing this command: `. $SPACK_ROOT/share/spack/setup-env.sh`

14.3 Installing LAMMPS with Spack example

This section details how to install the LAMMPS application with the KOKKOS and User-reaxc packages using Spack. This example assumes that you do not have any previous versions of LAMMPS installed. If you have any previous versions of LAMMPS, you must uninstall them before using this procedure. To see if you have any previous versions of LAMMPS, type `spack find lammps`. If you do have a previous version, you will need to uninstall LAMMPS by typing `spack uninstall --dependents lammps`. Then, you can follow the instructions below. Note that the installation can take about two hours to complete. As part of the procedure, we recommend that you initiate a tmux session so that you can have the installation running as a separate process if you need to do other work on Discovery. If you decide to use tmux, make note of the compute node number (compute node numbers start with c or d with four numbers, such as c0123) to make it easier to check on the progress of the installation.

If LAMMPS has a dependency on a specific `gcc` compiler, then do the following before starting the installation procedure. This will update the `compilers.yaml` file located in `$HOME/.spack/linux`.

1. `cd $HOME/.spack/linux/`
2. Open `compilers.yaml` and copy-paste a `compiler` entry at the end of the file.
3. Edit 'spec' and 'path' to indicate the version of the gcc compiler that is required for installation.

```
For example:
     spec: gcc@=8.1.0
         paths:
           cc: /shared/centos7/gcc/8.1.0/bin/gcc
            cxx: /shared/centos7/gcc/8.1.0/bin/g++
           f77: /shared/centos7/gcc/8.1.0/bin/gfortran
             fc: /shared/centos7/gcc/8.1.0/bin/gfortran
```

4. The `compilers.yaml` file should now have the desired `gcc` version as its latest `compiler` entry.
5. Assuming that Spack has already been installed at a desired location. For installing gpu-supported LAMMPS, request a GPU node for 8 hours:

```
srun --partition=gpu --nodes=1 --ntasks=14 --pty --gres=gpu:1 --mem=16GB --
↪time=08:00:00 /bin/bash
```

6. Load compatible cuda, gcc, and python modules and activate Spack from the installed location.

```
 module load cuda/10.2 gcc/8.1.0 python/3.8.1
 export SPACK_ROOT=/work/<PI-Project-Dir>/spack
 . $SPACK_ROOT/share/spack/setup-env.sh
```

7. (Optional) Initiate a `tmux` session:
   - Start a tmux session: `tmux`.
   - List tmux sessions: `tmux ls`
   - Detach from tmux session: `Ctrl+b d`
   - Attach to tmux session: `tmux attach-session -t 0`
   - Exit a tmux session: `Ctrl+d`
8. Type:

```
spack install lammps +asphere +body +class2 +colloid +compress +coreshell +cuda \
cuda_arch=70 +cuda_mps +dipole +granular +kokkos +kspace +manybody +mc +misc␣
↪+molecule \
+mpiio +peri +python +qeq +replica +rigid +shock +snap +spin +srd +user-reaxc +user-
↪misc
```

9. Type `spack find LAMMPS` to view your installed software package.
10. Type `spack load lammps`.

CHAPTER FIFTEEN

USING R

R is available as a module (see *Using Module* for more information) and it is also an interactive app on Open onDemand (see *Introduction to Open OnDemand (OOD)* for more information). You can also use R with Anaconda. See *Working with Conda/Miniconda/Anaconda* and the Anaconda documentation for more information

If you work with R packages, using a Packrat environment can be helpful. Use the procedure below to create a Packrat environment on Discovery.

15.1 Creating a Packrat Environment (for R)

Packrat is an application that helps you manage packages for R. After you create a new directory for your R project, you can then use Packrat to store your package dependencies inside it. For more information about Packrat, see the website: https://rstudio.github.io/packrat/.

1. Connect to Discovery.
2. Type `module load R/4.2.1`.
3. Create a new directory for your R project by typing, `mkdir /scratch/<username>/<directoryname>` where `<username>` is your username, and `<directoryname>` is the name of the directory you want to create for your R project. For example, `/scratch/j.smith/packrat_r`.
4. Create a new directory for your R project by typing, `mkdir /scratch/<yourusername>/<directoryname>` where `yourusername` is your user name, and `directoryname` is the name of the directory you want to create for your R project. For example, `/scratch/j.smith/r_testlab`
5. Open the R interface and install Packrat:

```
install.packages("packrat") # install in a local directory, as you cannot install as␣
↪root
```

5. Initialize the environment where R packages will be written to:

```
packrat::init("/scratch/<yourusername>/<directoryname>")
```

You can then install R packages that you need. For example, to install a package called `rtracklayer`, type the following:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
install.packages("BiocManager")
BiocManager::install("rtracklayer")
```

When using RStudio in the OOD App:

The instructions below can be applied on any RStudio "flavor" available (i.e., RStudio, Geospatial, and Tidyverse). Once a packrat snapshot is created it can easily be transfered between flavors and even machines (e.g., personal laptop, Discovery).

1. Launch an RStudio instance on the OOD. Specify the flavor and other parameters as usual.
2. In the RStudio console type: `install.packages("packrat)`.

Note: This will install by default in `$HOME/R/x86_64-pc-linux-gnu-library/<version>/` as long as you don't have previous environments or those have been turned off (see below). For packrat installation, it is best to specify a "project folder" in your `$HOME`, `/scratch` or `/work` directory (if you do not have `/work` please see here for access). The location `/tmp/Rtmp8CbQCA/downloaded_packages` would not work because `/tmp` corresponds to the compute node that you were on while running the R session. Optimally, you would like to have the packrat location in a persistent place so that all packages and libraries are available to you at all times regardless of the compute node you are on.

---

3. Create a packrat project directory at the desired location by selecting "New Folder" in the "Files" tab in the lower right hand side of the RStudio screen. Alternatively, use `mkdir` in the terminal tab on the lower left-hand side of the RStudio screen. For example: `mkdir projectfolder`.
4. In the RStudio console, initialize the packrat. If your current directory is the project folder (i.e., `getwd() ==` "packrat project folder") you can omit the path here.

```
packrat::init("<path-to-project-folder>")
```

5. You can now record all the currently installed packages to your packrat with the snapshot command. This may take some time if you have installed a lot of packages.

```
packrat::snapshot()
```

6. And now you can check on the status of your packrat with:

```
packrat::status()
```

7. Now turn packrat on. Packrat will now stay on for all your RStudio sessions and across the RStudio flavors (RStudio, geospatial, and tidyverse).

```
packrat::on()
```

8. You can now install packages as normal. You should see the install location for your packrat project folder. E.g., "Installing package into '/work/groupname/username/packrat_R/'"

```
install.packages("viridis")
```

15.2 Packrat Tips

- At any time you can check the status of your packrat with `packrat::status()`.
- Packrat can be toggled on and off with `packrat::on()` and `packrat::off()`, respectively.
- To disconnect packrat and allow for package installation outside of your packrat project folder: `packrat::disable(project = NULL, restart = TRUE)`, where `project` refers to the current packrat project folder, and `restart = TRUE` will restart the R session.
- To re-initialize packrat run: `packrat::init("<path-to-packrat-project-folder>")`. This will automatically restart your R session.
- A package can be removed from packrat via: `remove.packages("viridis)`, but will remain in your packrat snapshot and can be restored with: `packrat::restore()`.
- The function `packrat::clean(dry.run=T)` will list any unused packages that were installed in your snapshot. You can remove them with: `packrat::clean()`.

Note: For most cases, having a single packrat directory is sufficient, unless you notice specific package conflicts or need different versions of the same package. A single packrat directory also saves from having to install the same dependencies multiple times in different locations.

If the installation location is not setting to your project folder you may need to turn off these environments. In some cases, these folders could also be present in your `/work/groupname/<project-name>` directory.

```
mv ~/.rstudio ~/.rstudio-off
mv ~/.local ~/.local-off
mv ~/ondemand ~/ondemand.off
mv ~/.Rprofile ~/.Rprofile.off
mv ~/.Rhistory ~/.Rhistory.off
```

CHAPTER

SIXTEEN

USING MPI

Messaging Passing Interface (MPI) is a standardized and portable message-passing system designed to function on a wide variety of parallel computing architectures. It provides a library of functions that enable a program to distribute a computational task across multiple nodes in a cluster.

There are multiple implementations of MPI including OpenMPI (Open Source Message Passing Interface), MPICH, and Intel MPI. OpenMPI is a widely used MPI implementation in the HPC community and the one we will be working with through our documentation.

16.1 Getting Started with MPI

To get started with MPI on a Slurm-based HPC cluster, you should have:

- Basic knowledge of Linux/Unix commands
- Familiarity with a programming language supported by MPI (e.g., C, C++, Fortran) if you are developing a program
- Understand how to load MPI module on the HPC
- Understand how to compile your source code and run the binaries (compiled languages) or to run the interpreted language with MPI

16.1.1 MPI libraries on Discovery

There are many versions of OpenMPI, MVAPICH, and MPICH that are available on the HPC as modules compiled with different compilers and additional libraries and features. To see them, use the `module avail openmpi`, `module avail mpich`, and `module avail mvapich` respectively.

Use the `module show` command to view information about the compilers you need to use with these libraries and if they support InfiniBand (IB) or not. For example, `module show openmpi/4.1.0-zen2-gcc10.1`.

Output for `module show openmpi/4.1.0-zen2-gcc10.1`

```
/shared/centos7/modulefiles/openmpi/4.1.0-zen2-gcc10.1:

module-whatis         Loads the executables, libraries and headers for OpenMPI v. 4.1.1.␣
↪Built using Intel 2021 compilers on AMD EPYC architecture (zen2).

Please note - this MPI module supports communication through the HDR200 InfiniBand␣
↪network by using the Mellanox (OFED 5.3) UCX (1.10.1) framework with cross platform␣
↪unified API. To make sure InfiniBand is being used, make sure to compile and run your␣
↪applications using this module only on AMD EPYC architectures (zen2).

To allocate the zen2 arch compute node, add the following flag to your SLURM command: --
↪constraint=zen2
For more details:
https://rc-docs.northeastern.edu/en/latest/hardware/hardware_overview.html

To use the module, type:
module load gcc/10.1.0
module load openmpi/4.1.0-zen2-gcc10.1


conflict        openmpi
prepend-path        PATH /shared/centos7/openmpi/4.1.0-zen2-gcc10.1/bin
prepend-path        MANPATH /shared/centos7/openmpi/4.1.0-zen2-gcc10.1/share/man
prepend-path        LD_LIBRARY_PATH /shared/centos7/openmpi/4.1.0-zen2-gcc10.1/lib
prepend-path        CPATH /shared/centos7/openmpi/4.1.0-zen2-gcc10.1/include
prepend-path        LIBRARY_PATH /shared/centos7/openmpi/4.1.0-zen2-gcc10.1/lib
setenv              OMPI_MCA_btl ^vader,tcp,openib,uct
```

16.2 Running a MPI Program

The following is a basic slurm script for running an MPI program with annotations:

```
#!/bin/bash
#SBATCH --job-name=test_job        # Set the job name
#SBATCH --output=res_%j.out        # Set the output file name (%j expands to jobId)
#SBATCH --ntasks=4                 # Request 4 tasks
#SBATCH --time=01:00:00            # Request 1 hour runtime
#SBATCH --mem-per-cpu=2000         # Request 2000MB memory per CPU

module load openmpi/4.0.5          # Load the necessary module(s)
mpirun -n 4 ./your_program         # Run your MPI executable
```

Note: For MPI tasks, `--ntasks=X` is used, where `X` requests the number of cpu cores for tasks.

This script specifies that it needs 4 tasks (i.e., CPU cores), a maximum of 10 minutes of runtime, and 2000MB of memory per CPU. It then loads the OpenMPI module and runs the MPI program using mpirun.

Tip: Best practice for writing your sbatch script is including the versions of the modules you are loading to ensure you always have your expected environment on the HPC.

16.3 OpenMPI Tuning for Performance Optimization

OpenMPI provides a variety of environment variables that can be used to optimize the runtime characteristics of your MPI program for maximum performance. For instance, you can specify which network interfaces to use by setting the `OMPI_MCA_btl` variable:

```
export OMPI_MCA_btl=self,vader,tcp
```

Tip: You can also include or exclude certain network interfaces by setting the `OMPI_MCA_btl_tcp_if_include` or `OMPI_MCA_btl_tcp_if_exclude` variables.

Also you can check if certain MPI modules already have certain `OMPI_MCA_btl` set by using the `module show` command and looking for the `setenv` options listed.

In addition, OpenMPI lets you control the placement of processes on nodes, which can be critical for performance. The `--map-by` and `--bind-to` options dictate how processes are mapped to hardware resources and how they are bound to those resources, respectively.

Remember, optimizing for performance often requires a thorough understanding of your application, your hardware, and MPI.

16.4 Troubleshooting and Debugging MPI Programs

Debugging MPI programs can be challenging due to their parallel nature. Fortunately, OpenMPI provides several tools and techniques to help with this.

One useful feature is verbose error reporting. To enable this, set the `OMPI_MCA_mpi_abort_print_stack` to 1:

```
export OMPI_MCA_mpi_abort_print_stack=1
```

If you have a parallel debugger such as TotalView or DDT, you can use it with OpenMPI using the `mpiexec` command with the `-tv` or `-debug` options, respectively.

Finally, remember to check your slurm job output files for any error messages or abnormal output. Sometimes, the issue may be with how you're running your job rather than with your MPI program itself.

16.5 Benchmarking OpenMPI Performance

Benchmarking is a method used to measure the performance of a system or one of its components under different conditions. For MPI, benchmarks can be used to measure its communication and computation efficiency on different high-performance computing (HPC) systems. By comparing these benchmarks, you can identify potential bottlenecks and areas for improvement to optimize the performance of MPI.

16.5.1 Tools for Benchmarking

There are several tools available for benchmarking MPI, including the following:

- **HPC Challenge (HPCC):** This benchmark suite measures a range of metrics, including latency and bandwidth, as well as floating-point computation performance.
- **Intel MPI Benchmarks (IMB):** A suite of benchmarks provided by Intel specifically for MPI. It includes a set of MPI-1 and MPI-2 function benchmarks and measures point-to-point communication, MPI data types, collective communication, and more.
- **OSU Micro-Benchmarks (OSU-MB):** A lightweight set of benchmarks designed to measure latency, bandwidth, and other performance metrics for various MPI functions.

To use these tools, you generally need to download and compile them, and then run them using a slurm job script.

16.6 Developing with MPI

16.6.1 Hello world program

The fundamental concept in MPI is the communicator, which defines a group of processes that can send messages to each other. By default, all processes belong to the `MPI_COMM_WORLD` communicator. Here's a simple C++ program that using MPI:

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(NULL, NULL);

    // Get the number of processes
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    // Get the rank of the process
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    printf("Hello world from processor %d out of %d processors\n", world_rank, world_
↪size);

    MPI_Finalize();
}
```

In this code, `MPI_Init` initializes the MPI environment, `MPI_Comm_size` gets the number of processes, `MPI_Comm_rank` gets the rank (ID) of the process, and `MPI_Finalize` ends the MPI environment. In the C/C++ language, the `#include <mpi.h>` header file needs to be added to compile MPI code.

To understand how to run an MPI program, let's write a simple program that prints a `"Hello, World!"` message from each process.

First, create a file called `hello_world.c` in your preferred text editor and add the following code:

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(NULL, NULL);
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    printf("Hello, World! I am process %d out of %d\n", world_rank, world_size);
    MPI_Finalize();
    return 0;
}
```

This program initializes the MPI environment, gets the rank of the process, gets the total number of processes, and then prints a message. Finally, it finalizes the MPI environment.

Next, compile the program using the `mpicc` command, which is a wrapper for the C compiler that includes the OpenMPI libraries:

```
mpicc hello_world.c -o hello_world
```

Where `-o` is the output flag, naming the executable `hello_world`. If omitted (i.e., `mpicc helloworld_c` would generate a compiled executable named `a.out` by default).

Finally, create a slurm job script to run the program:

```bash
#!/bin/bash
#SBATCH --job-name=hello_world
#SBATCH --output=result.txt
#SBATCH --ntasks=4
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=2000

module load openmpi/4.0.5
mpirun -n 4 ./hello_world
```

Submit this script to slurm with the `sbatch` command:

```
sbatch job_script.sh
```

You should see output in the `result.txt` file that shows `"Hello, World!"` messages from each process.

16.6.2 MPI Communication: Send and Receive Operations

MPI allows processes to communicate by sending and receiving messages. These messages can contain any type of data. Here's a simple example of using `MPI_Send` and `MPI_Recv` to send a number from one process to another:

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    MPI_Init(NULL, NULL);

```

(continues on next page)

(continued from previous page)

```c
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    int number;
    if (world_rank == 0) {
        number = -1;
        MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
    } else if (world_rank == 1) {
        MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process 1 received number %d from process 0\n", number);
    }

    MPI_Finalize();
}
```

16.6.3 MPI Monte Carlo

A key aspect of using OpenMPI is the ability to implement parallel algorithms, which can significantly speed up computation. Here is an example of a parallel version of the Monte Carlo method for estimating the number π:

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char** argv) {
    MPI_Init(NULL, NULL);

    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
    srand(time(NULL) * world_rank); // Ensure random numbers on all processes

    int local_count = 0;
    int global_count = 0;
    int flip = 1 << 24;
    double x, y, z;

    // Calculate hits within circle locally
    for (int i = 0; i < flip; i++) {
        x = (double)rand() / (double)RAND_MAX;
        y = (double)rand() / (double)RAND_MAX;
        z = sqrt((x*x) + (y*y));
        if (z <= 1.0) {
            local_count++;
        }
    }

    // Combine all local sums into the global sum
    MPI_Reduce(&local_count, &global_count, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    // Process 0 calculates pi and prints the result
```

(continues on next page)

(continued from previous page)

```
    if (world_rank == 0) {
        double pi = ((double)global_count / (double)(flip * world_rank)) * 4.0;
        printf("The estimated value of pi is %f\n", pi);
    }

    MPI_Finalize();
}
```

In this code, each process performs its own Monte Carlo simulation and then combines its results with those from other processes using the `MPI_Reduce` function.

16.6.4 Using OpenMPI with Python's mpi4py

mpi4py is a Python package that provides bindings to the MPI standard. It allows Python programs to take advantage of the distributed memory model and scale across multiple nodes of a high performance computing cluster, just like MPI.

The mpi4py package has been designed to be as close as possible to the MPI standard, providing Python developers with a familiar and straightforward interface to MPI.

In this program, `process 0` sends the number `-1` to `process 1`, which receives it and prints it.

To install mpi4py inside of a conda environment:

```
srun -n 4 --pty /bin/bash
module load anaconda3/2022.05
mkdir -p /path/to/mpi4py_env
conda create --prefix=/path/to/mpi4py_env -y
source activate /path/to/mpi4py_env
conda install -c conda-forge mpi4py
```

In your preferred text editor, write a file `hello.py` that contains the following:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    print('Hello from the master process')
else:
    print(f'Hello from process {rank}')
```

This program gets the communicator for the current process, obtains the rank of the process, and then prints a message. If the rank is 0, the process is the master, otherwise, it is a worker.

You can run this program using the `mpirun` command:

```
srun -n 4 --pty /bin/bash
mpirun -np 4 python hello_world.py
```

This will run the program on 4 processes.

Just like in MPI, mpi4py allows you to perform point-to-point communication using the `send` and `recv` methods, and collective communication using methods like `bcast` (broadcast), `gather`, and `reduce`.

Here's an example of point-to-point communication:

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = {'a': 1, 'b': 2, 'c': 3}
    comm.send(data, dest=1)
else:
    data = comm.recv(source=0)
    print(f'Received data {data} at process {rank}')
```

In this program, the master process sends a dictionary to a specific process and that process receives the dictionary.

Note: Anything greater than rank 2 will make this program hang.

16.6.5 Writing Efficient MPI Code

Efficiency and scalability are crucial when writing MPI code. Here are some tips to follow:

- **Overlap Computation and Communication:** Whenever possible, organize your code so that computation can occur while communication is ongoing. This will reduce the waiting time for communication to complete.
- **Minimize Communication:** Communication is often the bottleneck in parallel programs. Therefore, design your algorithms to minimize the amount of data that needs to be sent between processes.
- **Use Collective Operations:** MPI provides collective operations like `MPI_Bcast` and `MPI_Reduce`. These operations are often optimized for the underlying hardware and should be used whenever possible.
- **Use Non-Blocking Operations:** MPI also provides non-blocking versions of its send and receive functions. These functions (`MPI_Isend` and `MPI_Irecv`) return immediately, allowing the program to continue executing while communication is happening in the background.

16.7 Getting Help with MPI

For assistance with getting started with using MPI or troubleshooting using MPI libraries on Discovery, reach out to us at rchelp@northeastern.edu or schedule a consultation with one of our team members.

CHAPTER

SEVENTEEN

USING SLURM

17.1 Slurm Overview

Slurm (Simple Linux Utility for Resource Management) is an open-source, highly configurable, fault-tolerant, and adaptable workload manager. It is extensively used across High-Performance Computing (HPC) environments.

Slurm is designed to accommodate the complex needs of large-scale computational workloads. It can efficiently distribute and manage tasks across clusters comprising thousands of nodes, offering seamless control over resources, scheduling, and job queuing. It is the software on the HPC that provides functionalities such as *Slurm Job Arrays*, *Job Management*, view *Account information*, and check the *Cluster and Node States: sinfo*.

17.1.1 Slurm on HPC

HPC systems are designed to perform complex, computationally intensive tasks. For example, users can specify complex workflows of jobs where specific jobs depend on others, and Slurm will manage the scheduling and execution of these workflows. Efficiently managing these tasks and resources in such an environment is a daunting challenge. That's where Slurm comes into play.

Slurm allows users to submit their computational tasks as jobs to be scheduled on the cluster's compute nodes. Its role-based access control ensures proper resource allocation and job execution, preventing resource conflicts.

Slurm is crucial in research environments, where it ensures fair usage of resources among a multitude of users, helps optimize the workload for the available resources, and provides precise job accounting and statistics.

17.1.2 Page Objective:

To understand the Slurm workload manage, which will allow you to properly leverage the HPC. It starts with the basics - the resources that Slurm manager. Then, useful Slurm features (e.g., job submission, monitoring, canceling, etc.) are mentioned with code examples. We discuss jobs that are both interactive (i.e., *Interactive Jobs: srun Command*) and batch (i.e., *Batch Jobs: sbatch*), along with the slurm array variants (i.e., *Slurm Job Arrays*). *Advanced Usage*, *Common Problems and Troubleshooting*, and *Best Practices* are also covered.

17.1.3 Who Should Use This Guide?

This guide is for HPC users: researchers intending to use Slurm-based clusters for their computation tasks, system administrators managing HPC environments, and even seasoned HPC users looking to brush up on their knowledge. It progresses from fundamental to advanced topics, making it a valuable resource for a broad audience.

17.2 Slurm: Basic Concepts

Before we delve into using Slurm, it's essential to grasp some fundamental concepts related to its operation.

17.2.1 Nodes

In the context of Slurm, a 'node' refers to a server within the HPC cluster. Each node possesses its resources, such as CPUs, memory, storage, and potentially GPUs. Slurm manages these nodes and allocates resources to the tasks.

17.2.2 Partition(s)

A 'partition' is a grouping of nodes. You can think of partitions as virtual clusters within your HPC system. They allow administrators to segregate the compute environment based on factors like job sizes, hardware type, or resource allocation policies.

See also:

Our *Partitions* documentation.

17.2.3 Account information

When running a job with either `srun` or `sbatch`, if you have more than one account associated with your username, we recommend you use the `--account=` flag and specify the account that corresponds to the respective project.

To find out what account(s) your username is associated with, use the following command:

```
sacctmgr show associations user=<yourusername>
```

After you have determined what accounts your username is associated with, if you have more than one account association, you can use the `account=` flag with your usual `srun` or `sbatch` commands.

17.2.4 Jobs, Job Steps, and Job Arrays

A **job** in Slurm is a user-defined computational task that's submitted to the cluster for execution. Each job has one or more **job steps**, sub-tasks that are part of a larger job and can be executed in parallel or sequentially.

Job arrays are a series of similar jobs that differ only by the array index. They're especially useful when you want to execute the same application with different inputs.

17.2.5 Tasks

Tasks are the individual processes that run within a job step. They could be single-threaded or multi-threaded and can run on one or more nodes.

17.2.6 Understanding the Configuration File

Note: Most users should not be concerned with slurm configurations, beyond being aware of the configurations.

See also:

Slurm documentation for a complete list of available parameters as shown in the example config file belong, along with their meanings.

The slurm.conf file is the primary configuration file (i.e., `slurm.conf`) for Slurm. It contains the parameters that govern the behavior of the Slurm controller, nodes, and partitions.

Example of a very basic `slurm.conf` file:

```
# Basic slurm.conf file
ControlMachine=slurm-controller  # Hostname of the control node
AuthType=auth/munge              # Authentication type
CryptoType=crypto/munge          # Cryptography type
MpiDefault=none                  # Default MPI type
ProctrackType=proctrack/pgid     # Process tracking type
ReturnToService=1                # Return failed nodes to service
SlurmctldPidFile=/var/run/slurmctld.pid  # PID file for the Slurm controller
SlurmctldPort=6817               # Communication port for the Slurm controller
SlurmdPidFile=/var/run/slurmd.pid        # PID file for the Slurm daemon
SlurmdPort=6818                  # Communication port for the Slurm daemon
SlurmdSpoolDir=/var/spool/slurmd # Spool directory for the Slurm daemon
SlurmUser=slurm                  # User the Slurm daemon runs as
StateSaveLocation=/var/spool/slurmctld # Where state information is saved
SwitchType=switch/none           # Job switch type
TaskPlugin=task/none             # Task plugin
#
# Node definitions
NodeName=compute[1-16] CPUs=1 State=UNKNOWN
#
# Partition definitions
PartitionName=test Nodes=compute[1-16] Default=YES MaxTime=INFINITE State=UP
```

17.3 Basic Slurm Usage

To submit your job script to Slurm, you use the `sbatch` command:

```
sbatch job_script.sh
```

Slurm will then schedule your job to run when resources are available. It returns a job ID that you can use to monitor your job's status.

To check the status of your job, you use the `squeue` command:

```
squeue -u  <username>
```

To cancel a job, you use the `scancel` command with the job ID:

```
scancel  <job_id>
```

To check detailed information about a job, use the `scontrol` command:

```
scontrol show job  <job_id>
```

This information is crucial for managing your jobs and ensuring they are running as expected.

To view cluster information, use `sinfo`: this command allows to view partition and node information. Use option -a to view all partitions.

```
sinfo <options>
```

Details on each of the commands above, and more, is covered in the following sections.

17.4 Allocating Resources

You have two options when running tasks, run interactively via *Interactive Jobs: srun Command* or by batch via *Batch Jobs: sbatch*.

For parallel tasks, one can treat each task as a separate job and run them independently. The other option is to allocate resources for all the jobs simultaneously, allowing them to overlap (share CPUs, RAM, etc.). This is done with the `--overlap` flag: the assumption must be that not all tasks require all resources simultaneously, creating a more natural working environment, and resources are not wasted on idle time.

Note: While the `sbatch` and `srun` commands request resource allocation if none exists, using `salloc` allows us to separate the allocation and submission processes.

Some things Slurm assumes, like there is no overlap between different CPUs by default: tasks do not share CPUs with others running parallel. If overlap is needed, use the following Slurm flag:

```
--overlap
```

Also, set the environment variable `SLURM_OVERLAP=1` via

```
export SLURM_OVERLAP=1
```

Important: Run `export SLURM_OVERLAP=1` prior to logging onto a compute node when using MPI interactively.

17.5 Batch Jobs: `sbatch`

The `sbatch` command is used to submit a job script for later execution. The script includes the `SBATCH` directives that control the job parameters like the number of nodes, CPUs per task, job name, etc.

17.5.1 Syntax: `sbatch`

```
sbatch [options] <script_file>
```

17.5.2 Options and Usage: `sbatch`

- `n, --ntasks= <number>` : specify the number of tasks
- `N, --nodes=<minnodes[-maxnodes]>` : specify the number of nodes
- `J, --job-name=<jobname>` : specify a name for the job

```
#!/bin/bash
#SBATCH -J MyJob                # Job name
#SBATCH -N 2                    # Number of nodes
#SBATCH -n 16                   # Number of tasks
#SBATCH -o output_%j.txt        # Standard output file
#SBATCH -e error_%j.txt         # Standard error file

# Your program/command here
srun ./my_program
```

To submit this job script, save it as `my_job.sh` and run:

```
sbatch my_job.sh
```

17.5.3 Examples using `sbatch`

Single node

Run a job on one node for four hours on the short partition:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --time=4:00:00
#SBATCH --job-name=MyJobName
#SBATCH --partition=short

# <commands to execute>
```

Single node with additional memory

The default memory per allocated core is 1.95GB. If calculations attempt to access more memory than allocated, Slurm automatically terminates the job. Request a specific amount of memory in the job script if calculations require more than the default. The example script below requests 100GB of memory (`--mem=100G`). Use one capital letter to abbreviate the unit of memory (i.e., kilo `K`, mega `M`, giga `G`, and tera `T`) with the `--mem=` option, as that is what Slurm expects to see:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --time=4:00:00
#SBATCH --job-name=MyJobName
#SBATCH --mem=100G
#SBATCH --partition=short

# <commands to execute>
```

Single with exclusive use of a node

If you need exclusive use of a node, such as when you have a job that has high I/O requirements, you can use the exclusive flag. The example script below specifies the exclusive use of one node in the short partition for four hours:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --time=4:00:00
#SBATCH --job-name=MyJobName
#SBATCH --exclusive
#SBATCH --partition=short

# <commands to execute>
```

17.6 Interactive Jobs: `srun` Command

The `srun` command is used to submit an interactive job which runs a single task directly via the shell. This method is useful when you want to test a short computation or run an interactive session like a shell, Python, or an R terminal.

17.6.1 Syntax: `srun`

```
srun [options] [command]
```

17.6.2 Options and Usage: `srun`

- **`n, --ntasks=<number>`**: specify the number of tasks
- **`N, --nodes=<minnodes[-maxnodes]>`**: specify the number of nodes
- **`J, --job-name=<jobname>`**: specify a name for the job

```
srun -N 1 -n 1 --pty bash
```

This command starts an interactive bash shell on one node with one task.

17.6.3 Examples using `srun`

The user needs to review the *Hardware Overview* and *Partitions* to be familiar with the available hardware and partition limits on Discovery. This way, user can tailor the request to fit both the needs of the job and the limits of partitions. For example, if the user specifies `--partition=short` and `--time=01:00:00`, it will result in an error because the time specified exceeds the limit for that partition.

This simple `srun` example is to move to a *compute* node after you first log into the HPC:

```
srun --pty /bin/bash
```

To request one node and one task for 30 minutes with X11 forwarding on the short partition, type:

```
srun --partition=short --nodes=1 --ntasks=1 --x11 --mem=10G --time=00:30:00 --pty /bin/
↪bash
```

To request one node, with 10 tasks and 2 CPUs per task (a total of 20 CPUs), 1 GB of memory, for one hour on the express partition, type:

```
srun --partition=short --nodes 1 --ntasks 10 --cpus-per-task 2 --pty --mem=1G --
↪time=01:00:00 /bin/bash
```

To request two nodes, each with 10 tasks per node and 2 CPUs per task (a total of 40 CPUs), 1 GB of memory, for one hour on the express partition, type:

```
srun --partition=short --nodes=2 --ntasks 10 --cpus-per-task 2 --pty --mem=1G --
↪time=01:00:00 /bin/bash
```

To allocate a GPU node, you should specify the `gpu` partition and use the `-gres` option:

```
srun --partition=gpu --nodes=1 --ntasks=1 --gres=gpu:1 --mem=1Gb --time=01:00:00 --pty /
↪bin/bash
```

17.7 Slurm Job Arrays

Job arrays are a convenient way to submit and manage large numbers of similar jobs quickly. They can process millions of tasks in milliseconds, provided they are within size limits. Job arrays are particularly useful when running similar jobs, such as performing the same analysis with different inputs or parameters.

Using job arrays can save time and reduce the amount of manual work required. Instead of submitting each job individually, you can submit a single job array and let Slurm handle the scheduling of individual jobs. This approach is beneficial if you have limited time or resources, as it allows you to use the cluster's computing power more efficiently by running multiple jobs in parallel.

There are several ways to define job arrays, such as specifying the range of indices or providing a list of indices in a file. Slurm also offers various features to manage and track job arrays, such as options to simultaneously suspend, resume, or cancel all jobs in the array.

17.7.1 Syntax: Job Arrays

The most basic configuration for a job array is as follows:

```
#!/bin/bash
#SBATCH --partition=short
#SBATCH --job-name=jarray-example
#SBATCH --output=out/array_%A_%a.out
#SBATCH --error=err/array_%A_%a.err
#SBATCH --array=1-6
```

This command runs the same script six times using Slurm job arrays. Each job array has two additional environment variable sets. `SLURM_ARRAY_JOB_ID` (`%A`) is set to the first job ID of the array, and `SLURM_ARRAY_TASK_ID` (`%a`) is set to the job array index value.

Note: Both the `SLURM_ARRAY_JOB_ID` (`%A`) and `SLURM_ARRAY_TASK_ID` (`%a`) are referenced when naming outputs so file do not overwrite when a "task" (i.e., one of the executions of the script through the job array) finishes.

Tip: Generally, we want to pass the former as an argument for our script. If you are using R, you can retrieve the former using `task_id <- Sys.getenv("SLURM_ARRAY_TASK_ID")`. If you are using job arrays with Python, you can obtain the task ID using the following:

```
import sys
taskId = sys.getenv('SLURM_ARRAY_TASK_ID')
```

When submitting an array and setting its size with many dimensions, please use the `%` symbol to indicate how many tasks run simultaneously. For example, the following code specifies an array of 600 jobs, with 20 running at a time:

```
#!/bin/bash
#SBATCH --partition=short
#SBATCH --job-name=jarray-example
#SBATCH --output=out/array_%A_%a.out
#SBATCH --error=err/array_%A_%a.err
#SBATCH --array=1-600%20
```

Whenever you specify the memory, number of nodes, number of CPUs, or other specifications, they will be applied to each task. Therefore, if we set the header of our submission file as follows:

```
#!/bin/bash
#SBATCH --partition=short
#SBATCH --job-name=jarray-example
#SBATCH --output=out/array_%A_%a.out
#SBATCH --error=err/array_%A_%a.err
#SBATCH --array=1-600%20
```

(continues on next page)

(continued from previous page)

```
#SBATCH --mem=128G
#SBATCH --nodes=2
```

Slurm will submit 20 jobs simultaneously. Each job, represented by a task ID, will use two nodes with 128GB of RAM each. In most cases, setting up a single task is sufficient.

Lastly, we usually use job arrays for embarrassingly parallel jobs. If your case is such that the job executed at each job ID does not use any multi-threading libraries, you can use the following header to avoid wasting resources:

```
#!/bin/bash
#SBATCH --partition=short
#SBATCH --job-name=jarray-example
#SBATCH --output=out/array_%A_%a.out
#SBATCH --error=err/array_%A_%a.err
#SBATCH --array=1-600%50  # 50 is the maximum number
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem-per-cpu=2G ## debug prior to know how much RAM
```

Warning: 50 is the maximum number of jobs allowed to be run at once per user-account.

The above examples apply for interactive mode, as well. For instance:

```
sbatch --array=<indexes> [options] script_file
```

Indexes can be listed as `1-5` (i.e., one to five), `=1,2,3,5,8,13` (i.e., each index listed), or `1-200%5` (i.e., produce a 200 task job array with only 5 tasks active at any given time). **The symbol used is the % sign, which tasks to be submitted at once** (again, cannot be set larger than 50).

17.7.2 Use-cases: Job Arrays

Job arrays can be used in situations where you have to process multiple data files using the same procedure or program. Instead of creating multiple scripts or running the same script multiple times, you can create a job array, and Slurm will handle the parallel execution for you.

17.7.3 Example using Job Array Flag

In the following script, the `$SLURM_ARRAY_TASK_ID` variable is used to differentiate between array tasks.

```
#!/bin/bash
#SBATCH -J MyArrayJob          # Job name
#SBATCH -N 1                   # Number of nodes
#SBATCH -n 1                   # Number of tasks
#SBATCH -o output_%A_%a.txt    # Standard output file (%A for array job ID, %a for␣
↪array index)
#SBATCH -e error_%A_%a.txt     # Standard error file

# Your program/command here
srun ./my_program input_$SLURM_ARRAY_TASK_ID
```

To submit this job array, save it as `my_array_job.sh` and run:

```
sbatch --array=1-50 my_array_job.sh
```

This command will submit 50 jobs, running `my_program` with `input_1` through `input_100`.

17.8 Job Management

Managing jobs in a Slurm-based HPC environment involves monitoring running jobs, modifying job parameters, and canceling jobs when necessary. This section will cover the commands and techniques you can use for these tasks.

17.8.1 Monitoring Jobs

The `squeue` command allows you to monitor the state of jobs in the queue. It provides information such as the job ID, the partition it's running on, the job name, and more.

Syntax: `squeue`

```
squeue [options]
```

Options and Usage

- **`j, --jobs=<job_id>`**: display information about specific job(s)
- **`u, --user=<user_name>`**: display jobs for a specific user
- **`l, --long`**: display more information (long format)

Code Example of Job Monitoring

To monitor all jobs of a specific user, use the following command:

```
squeue -u <username>
```

To monitor a specific job, use:

```
squeue -j <job_id>
```

17.8.2 Modifying Jobs

The `scontrol` command is a command-line utility that allows users to view and control Slurm jobs and job-related resources. It provides a way to check the status of jobs, modify job properties, and perform other job-related tasks.

You can monitor your jobs by using the Slurm `scontrol` command. Type `scontrol show jobid -d <JOBID>`, where `JOBID` is the number of your job. In the figure at the top of the page, you can see that when you submit your `srun` command, Slurm displays the unique ID number of your job (`job 12962519`). This is the number you use with `scontrol` to monitor your job.

Using `scontrol`

Some of the tasks that can done using `scontrol` include:

- **Viewing job status and properties:** `scontrol` can display detailed information about a job, including its status, node allocation, and other properties.
- **Modifying job properties:** `scontrol` allows users to modify job properties such as the job name, the number of nodes, the time limit, and other parameters.
- **Managing job dependencies:** `scontrol` provides a way to specify job dependencies and view the dependencies between jobs.
- **Suspending and resuming jobs:** `scontrol` can stop and resume running jobs, allowing users to temporarily halt jobs or continue them as needed.
- **Canceling jobs:** `scontrol` can cancel jobs that are running or queued, allowing users to stop jobs that are no longer needed.

Overall, `scontrol` is a powerful tool for managing Slurm jobs and job-related resources. Its command-line interface allows users to perform a wide range of tasks, from checking the status of jobs to modifying job properties and managing dependencies.

17.8.3 Controlling jobs: `scontrol`

Place a hold on a pending job, i.e., prevent specified job from starting. <job_list> is either a space separate list of job IDs or job names.

```
scontrol hold <jobid>
```

Release a held job, i.e., permit specified job to start (see hold).

```
scontrol release <jobid>
```

Re-queue a completed, failed, or cancelled job

```
scontrol requeue <jobid>
```

For more information on the commands listed above, along with a complete list of `scontrol` commands run below:

```
scontrol --help
```

Syntax: `scontrol`

```
scontrol [command] [options]
```

Example: `scontrol`

```
scontrol show jobid -d <JOBID>
```

Options and Usage: `scontrol`

- **`update`**: used to modify job or system configuration
- **`hold jobid=<job_id>`**: hold a specific job
- **`release jobid=<job_id>`**: release a specific job
- **`requeue jobid=<job_id>`**: requeue a specific job

17.8.4 Examples using `scontrol`

View information about a specific node:

```
scontrol show node -d <node_name>
```

For information on all reservations, this command will show information about a specific node in the cluster, including the node name, state, number of CPUs, and amount of memory:

```
scontrol show reservations
```

View information about a specific job. This command will show information about a specific job, including the job ID, state, username, and partition name:

```
scontrol show job <job_id>
```

To view information about a specific reservation (e.g., found via `scontrol show res` listed above), and print information about a specific reservation in the cluster, including the reservation name, start time, end time, and nodes included in the reservation:

```
scontrol show reservation <reservation_name>
```

17.8.5 Cancelling Jobs: `scancel`

The `scancel` command is used to cancel a running or pending job. Once cancelled, a job cannot be resumed.

Syntax: `scancel`

```
scancel [options] [job_id]
```

Options and Usage: `scancel`

- **u, --user=<user_name>**: cancel all jobs of a specific user
- **-name=<job_name>**: cancel all jobs with a specific name

Examples using `scancel`

To cancel a specific job, use:

```
scancel <job_id>
```

To cancel all jobs of a specific user:

```
scancel -u <username>
```

To cancel all jobs with a specific name:

```
scancel --name=<job_name>
```

The job management section aims to give you a solid understanding of how to manage and control your jobs effectively. Always ensure to monitor your jobs regularly and adjust parameters as needed to achieve the best performance.

17.9 Cluster and Node States: `sinfo`

Below are some more examples of using `sinfo` and `scontrol` to provide information about the state of the cluster and specific nodes.

17.9.1 Using `sinfo`

The `sinfo` command will show information about all partitions in the cluster, including the partition name, available nodes, and status. By default, `sinfo` reports:

```
| PARTITION | The list of the cluster's partitions; a set of compute nodes grouped␣
↪logically |
| --- | --- |
| AVAIL | The active state of the partition (up, down, idle) |
| TIMELIMIT | The maximum job execution wall-time per partition |
| NODES | The total number of nodes per partition |
| STATE | See STATE table below |
| NODELIST(REASON) | The list of nodes per partition |
```

17.9.2 Examples using `sinfo`

View information about all partitions:

```
sinfo -a
```

Or, a specific partition, which gives all the nodes and the states the nodes are in at the current time:

```
sinfo -p gpu
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
gpu          up    8:00:00      5 drain* c[2171,2184,2188],d[1008,1014]
gpu          up    8:00:00      3  down* c2162,d[1006,1017]
gpu          up    8:00:00      1  drain d1025
gpu          up    8:00:00      2   resv c2177,d1029
gpu          up    8:00:00     50    mix c[2160,2163-2170,2172-2176,2178-2179,2185-2187,
↪2189-2195,2204-2207],d[1001,1003-1005,1007,1009-1013,1016,1018,1020-1024,1026-1028]
gpu          up    8:00:00      3  alloc d[1002,1015,1019]
gpu          up    8:00:00      4   idle c[2180-2183]
```

The current `TimeLimit` for the queues:

```
sinfo  -o "%12P %.10A %.11l"
PARTITION    NODES(A/I)   TIMELIMIT
debug           402/174       20:00
express         403/180     1:00:00
short*          401/178  1-00:00:00
long             224/47  5-00:00:00
large           376/172     6:00:00
gpu               41/17     8:00:00
multigpu          41/17  1-00:00:00
lowpriority     118/102  1-00:00:00
reservation     617/402 100-00:00:0
ai-jumpstart       2/15  2-00:00:00
allshouse           5/7    infinite
bansil             15/4 30-00:00:00
ce-mri             3/10 30-00:00:00
chen               0/12 30-00:00:00
ctbp               0/20 30-00:00:00
.
.
.
```

View information about a specific partition (e.g., `short`, `gpu`, `long`):

```
sinfo -p <partition_name>
```

Or, only view nodes in a certain state:

```
sinfo -p <partition> -t <state>
```

You can use the `--Format` flag to get more information or a specific format for the output:

```
sinfo -p <partition> -t idle --Format=gres,nodes
```

Below command will show detailed information about all nodes in the cluster, including the node name, state, CPU architecture, memory, and available features:

```
sinfo -N -l
```

View what features a node has:

```
sinfo -n <node> --Format=nodes,nodelist,statecompact,features
```

View what nodes are in what state in a partition using `statecompact`:

```
sinfo -p <partition> --Format=time,nodes,statecompact,features,memory,cpus,nodelist
```

17.10 Advanced Usage

Advanced usage of Slurm involves working with *Multi-node Jobs*, *GPU Jobs*, and understanding *Priority and QoS* parameters. It also involves *Memory Management* and *Using Environment Variables in Job Scripts*.

17.10.1 Multi-node Jobs

Multi-node jobs involve executing a single job across multiple nodes. Such jobs are typically used for computationally intensive tasks that require significant parallelization.

Use-cases: Multi-node Jobs

Multi-node jobs are used in scenarios where tasks can be broken down into sub-tasks that can be executed in parallel, such as simulation and modeling, machine learning training, or big data analysis.

Code Example for Multi-node Job Submission

```
#!/bin/bash
#SBATCH -J MultiNodeJob     # Job name
#SBATCH -N 4                # Number of nodes
#SBATCH -n 16               # Number of tasks

# Your program/command here
srun ./my_program
```

17.10.2 GPU Jobs

Slurm can also manage GPU resources, allowing you to specify GPU requirements in your job scripts.

Use-cases: GPUs

GPU jobs are used in scenarios where tasks are parallelized and can benefit from the high computational capabilities of GPUs, such as machine learning and deep learning workloads, image processing, or simulations.

Code Example for GPU Job Submission

```
#!/bin/bash
#SBATCH -J GPUJob           # Job name
#SBATCH -N 1                # Number of nodes
#SBATCH -n 1               # Number of tasks
#SBATCH --gres=gpu:1        # Number of GPUs
#SBATCH -p gpu             # gpu partition

# Your program/command here
srun ./my_gpu_program
```

17.10.3 Priority and QoS

Slurm uses priority and Quality of Service (QoS) parameters to determine the order in which jobs are scheduled.

Understanding Job Priorities and Quality of Service (QoS) Parameters

The job priority is a numerical value assigned to each job, determining its position in the queue. Higher priority jobs are scheduled before lower priority ones. Quality of Service (QoS) parameters control various job limits, such as the maximum allowed job runtime, the maximum number of CPUs or nodes a job can use, etc.

Code Example to Manipulate Job Priority

```
scontrol update jobid=<job_id> priority=<new_priority>
```

17.10.4 Memory Management

In Slurm, memory allocation can be controlled on the job or task level using the `--mem` or `--mem-per-cpu` options, respectively.

Code example for specifying memory in job scripts

```
#!/bin/bash
#SBATCH -J MyJob             # Job name
#SBATCH -N 1                 # Number of nodes
#SBATCH -n 4                 # Number of tasks
#SBATCH --mem=8G             # Memory for the entire job

# Your program/command here
srun ./my_program
```

We can also specify memory per CPU.

```
#!/bin/bash
#SBATCH -J MyJob              # Job name
#SBATCH -N 1                  # Number of nodes
#SBATCH -n 4                  # Number of tasks
#SBATCH --mem-per-cpu=2G      # Memory per task (CPU)

# Your program/command here
srun ./my_program
```

Note: Either `--mem-per-cpu` or `--mem` can be specified as a sbatch directive, but not both.

17.10.5 Using Environment Variables in Job Scripts

Slurm sets several environment variables that you can use in your job scripts to control the job behavior dynamically. Some of these include `SLURM_JOB_ID`, `SLURM_JOB_NUM_NODES`, `SLURM_JOB_NODELIST`, etc.

Code Example Showcasing Use of Environment Variables

```
#!/bin/bash
#SBATCH -J MyJob              # Job name
#SBATCH -N 2                  # Number of nodes
#SBATCH -n 8                  # Number of tasks

echo "Job ID: $SLURM_JOB_ID"
echo "Number of nodes: $SLURM_JOB_NUM_NODES"
echo "Node list: $SLURM_JOB_NODELIST"

# Your program/command here
srun ./my_program
```

17.11 Common Problems and Troubleshooting

Despite its flexibility and robustness, it's not uncommon to encounter issues when using Slurm. Here we'll explore some common problems and provide strategies for debugging and optimizing job scripts.

17.11.1 Commonly Encountered Issues in Using Slurm

1. **Job Stuck in Queue:** If your job is stuck in the queue and not getting scheduled, it may be due to insufficient available resources, low priority, or system limits set by the Quality of Service (QoS) parameters.

   Solution: Check the job requirements, priority, and QoS parameters using `scontrol show job <job_id>`. Ensure your job requirements do not exceed available resources and system limits.

2. **Job Failed with Non-zero Exit Code:** If your job script or the program it's running exits with a non-zero code, it indicates an error.

   Solution: Check the job's output and error files for any error messages. They can provide valuable clues about what went wrong.

3. **Insufficient Memory:** If your job fails with "Out Of Memory" (OOM) errors, it means it's using more memory than allocated.

   Solution: Increase the `--mem` or `--mem-per-cpu` value in your job script. Remember that requesting more memory might increase the time your job spends in the queue.

17.11.2 Strategies for Debugging and Optimizing Job Scripts

1. **Testing Job Scripts Interactively:** Use the `srun` command to run your job script interactively for debugging purposes. This approach allows you to observe the program's behavior in real-time.

```
srun --pty bash -i
```

2. **Using echo Command for Debugging:** Use the `echo` command in your job script to print the values of variables, command outputs, etc., to the output file. This method can help you understand the script's flow and pinpoint any issues.

```
echo "Value of variable x is $x"
```

3. **Optimizing Job Resources:** Monitor your job's resource usage using `sstat <job_id>` and adjust the resource requirements accordingly in your job script. Requesting more resources than needed can result in your job spending more time in the queue, while requesting less than needed can lead to job failures. Remember, troubleshooting requires patience and a systematic approach. Begin by identifying the problem, then hypothesize potential causes, test those hypotheses, and apply solutions. Make small changes one at a time and retest after each change. With experience, you will be able to troubleshoot effectively and make the most of your HPC resources.

17.12 Best Practices

In this section, we will discuss some best practices that can help you make efficient use of resources, write optimized job scripts, and ensure maximum throughput and minimal queue time in a Slurm-based HPC environment.

17.12.1 Efficient Usage of Resources

1. **Request Only What You Need:** When submitting a job, request only the resources that you need. Overestimating your requirements can result in longer queue times as Slurm waits for the requested resources to become available.
2. **Use Job Arrays for Similar Jobs:** If you need to run multiple similar jobs, consider using job arrays. This approach makes job management more straightforward and reduces overhead.
3. **Use Appropriate Partitions:** Select the appropriate partition for your job based on your requirements. Each partition may have different limits and priorities, so choose the one that suits your needs best.

17.12.2 Writing Optimized Job Scripts

1. **Use Environment Variables:** Slurm provides several environment variables that can be used to customize job behavior dynamically. Use these variables to make your scripts more flexible and efficient.
2. **Specify All Necessary Options:** Ensure to specify all necessary SBATCH options in your job script. Missing options can lead to unpredictable job behavior or performance.
3. **Check Exit Codes:** Always check the exit codes of commands in your job script. Non-zero exit codes usually indicate an error, and failing to check these can lead to undetected job failures.

```
command
if [ $? -ne 0 ]; then
  echo "Command failed"
  exit 1
fi
```

17.12.3 Guidelines to Ensure Maximum Throughput and Minimal Queue Time

1. **Monitor Job Performance:** Regularly monitor your jobs' performance using `sstat` and adjust resource requests as needed. This can help improve job scheduling efficiency.
2. **Use the `time` Option Judiciously:** While it's crucial to give your job enough time to complete, overestimating can lead to longer queue times. Start with a reasonable estimate and adjust based on actual run times.
3. **Prioritize Critical Jobs:** If you have a critical job that needs to be run immediately, you can temporarily increase its priority using `scontrol`. However, use this feature sparingly to maintain fairness.

Remember, these best practices aim to ensure efficient and fair usage of shared HPC resources. Always be mindful of other users and try to use the resources in a way that maximizes productivity for everyone.

17.12.4 Proper resource request syntax

```
#!/bin/bash
#SBATCH --job-name=my_job       # Job name
#SBATCH --ntasks=4              # Number of tasks
#SBATCH --cpus-per-task=2       # Number of CPUs per task
#SBATCH --mem-per-cpu=4G        # Memory per CPU
#SBATCH --time=02:00:00         # Time limit
#SBATCH --partition=my_partition # Partition (queue) to use
```

Use environment modules: Load the necessary modules before running your job. In this example, we load the Python and TensorFlow modules:

```
module load python/3.8
module load tensorflow/2.5.0
```

For further reading and resources, consider the following:

- Slurm Quick Start User Guide
- Slurm Workload Manager Documentation
- High Performance Computing For Dummies, IBM Limited Edition If you need further support, please contact your system administrator or visit the Slurm community page for mailing lists and support forums.

17.13 Appendix

17.13.1 Glossary of Slurm Terms

1. **Node:** A computer or server in the HPC cluster.
2. **Partition:** A group of nodes with specific attributes and limits.
3. **Job:** A user-submitted work request.
4. **Task:** A unit of work within a job.

17.13.2 Full List of Slurm Commands and Their Options

- Slurm Commands
- Slurm Options

17.13.3 Sample Job Scripts for Various Use-cases

- Sample Job Scripts

17.13.4 Slurm FAQs

- Frequently Asked Questions.

17.13.5 Slurm References

1. SchedMD. (2023). Slurm Workload Manager. https://slurm.schedmd.com
2. SchedMD. (2023). Slurm Quick Start User Guide. https://slurm.schedmd.com/quickstart.html
3. IBM. (2023). High Performance Computing For Dummies, IBM Limited Edition. https://www.ibm.com/downloads/cas/WQDZWBYJ Thank you for following along with this guide, and we wish you success in your HPC journey!

CHAPTER

EIGHTEEN

RECURRING JOBS

You can use `scrontab` to schedule recurring jobs. Its syntax is similar to that of `[crontab](https://man7.org/linux/man-pages/man5/crontab.5.html)`, which is a standard Unix/Linux utility for running programs at specified intervals.

Tip: `scrontab` vs `crontab` If you are familiar with `crontab`, there are some important differences to note:

- The scheduled times for `scrontab` indicate when your job is *eligible* to start. They are not start times like in traditional Cron jobs.
- Jobs managed with `scrontab` won't start if an earlier iteration of the same job is still running. Cron will happily run multiple copies of a job at the same time.
- You have one `scrontab` file for the entire cluster, unlike `crontabs`, which are stored locally on each computer.

18.1 Set Up Your `scrontab`

See also:

Using Slurm

18.1.1 Edit Your `scrontab`

To edit your `scrontab` file, run `scrontab -e`. If you prefer to use `nano` to edit files, run `EDITOR=nano scrontab -e`.

Lines that start with `#SCRON` are treated like the beginning of a new batch job and work like `#SBATCH` directives for batch jobs. Slurm will ignore `#SBATCH` directives in scripts that you run as `scrontab` jobs. You can use most common `sbatch` options just as you would when using Slurm. The first line after your `SCRON` directives specifies the schedule for your job and the command to run.

Note: All of your `scrontab` jobs will start with your home directory as the working directory. You can change this with the `--chdir` Slurm option.

18.2 Cron syntax

Crontab's syntax is specified in five columns, which specify minutes, hours, days of the month, months, and days of the week. If you're new to crontab, it may be easiest to use a helper application to generate your cron date fields. Two popular options are crontab-generator and cronhub.io. Alternatively, you can use shorthand syntax such as `@hourly`, `@daily`, `@weekly`, `@monthly`, and `@yearly` instead of the five separate columns.

18.3 What to Run

If you're running a script, it must be marked as executable. Jobs handled by `scrontab` do not run in a full login shell. Therefore, if you have customized your `.bashrc` file, you need to add the following line to your script to ensure that your environment is set up correctly:

```
source ~/.bashrc
```

Note that the command specified in the `scrontab` file is executed via bash, NOT `sbatch`. You can list multiple commands separated by `;` and use other shell features, such as redirects. Additionally, any `#SBATCH` directives in executed scripts will be ignored. To use the `scrontab` file, you must use `#SCRON` instead.

Note: Your `crontab` jobs will appear to have the same JobID every time they run until the next time you edit your `crontab` file. This means that only the most recent job will be logged to the default output file. If you want a deeper history, redirect the output in your scripts to filenames with more unique names, such as a date or timestamp. For example:

```
python my_script.py > $(date +"%Y-%m-%d")_myjob_scrontab.out
```

If you want to see the accounting of a job that was handled by crontab, for example job `12345`, run the following command to view the slurm accounting:

```
sacct --duplicates --jobs 12345
# or with short options
sacct -Dj 12345
```

18.4 Recurring Job Examples

18.4.1 Running a Daily Simulation

This example demonstrates how to submit a 6-hour simulation that is eligible to start at 12:00 AM every day.

```
#SCRON --time 6:00:00
#SCRON --cpus-per-task 4
#SCRON --name "daily_sim"
#SCRON --chdir /home/netid/project
#SCRON -o my_simulations/%j-out.txt
@daily ./simulation_v2_final.sh
```

18.4.2 Running a Weekly Transfer Job

This example demonstrates how to submit a transfer script that is set to start every Wednesday at 8:00 PM.

```
#SCRON --time 1:00:00
#SCRON --partition transfer
#SCRON --chdir /home/netid/project/to_transfer
#SCRON -o transfer_log_%j.txt
0 20 * * 3 ./rclone_commands.sh
```

18.4.3 Capture output from each run in a separate file

By default, `crontab` overwrites the output file from the previous run when the same `jobid` is used. To avoid this, you can redirect the output to a file with a date-stamp.

```
0 20 * * 3 ./commands.sh > myjob_$(date +%Y%m%d%H%M).out
```

CHAPTER

NINETEEN

WORKING WITH GPUS

The cluster has various NVIDIA Graphics Processing Units (GPUs) available on gpu-equipped partitions, as listed in the table below.

See also:

Learn more about partitions.

GPU Type	GPU Memory	Tensor Cores	CUDA Cores	Nodes in Public GPUs	Nodes in Private GPUs
p100 (Pascal)	12GB	N/A	3,584	12 (x3-4 GPUs)	3 (x4 GPUs)
v100-pcie (Volta)	32GB	640	5,120	4 (x2 GPUs)	1 (x2 GPUs, 16GB)
v100-sxm2 (Volta)	32GB	640	5,120	24 (x4 GPUs)	10 (x4 GPUs, 16GB); 8 (x4 GPUs, 32GB)
t4 (Turing)	15GB	320	2,560	2 (x3-4 GPUs)	1 (x4 GPUs)
quadro (Quadro RTX 8000)	46GB	576	4,608	0	2 (x3 GPUs)
a30 (Ampere)	24GB	224	3,804	0	1 (x3 GPUs)
a100 (Amperea100)	41 & 82GB	432	6,912	3 (x4 GPUs)	15 (x2-8 GPUs)
a5000 (Ampere RTX A5000)	24GB	256	8,192	0	6 (x8 GPUs)
a6000 (Ampere RTX A6000)	49GB	336	10,752	0	3 (x8 GPUs)

The `gpu` partition is the general GPU resource for HPC users looking to use a GPU; `multigpu` is the alternative, where more than one GPU are accessible.

Anyone with a Discovery account can use the `gpu` partition. However, you must submit a ServiceNow ticket to request temporary access to `multigpu`: the `multigpu` partition is available for times in need for predefined time window. In other words, instances that require `multigpu` must demonstrate the need; furthermore, the specifics of the working code, as the partition is only accessible for a limited of time (e.g., 48 hours), so best to make sure to use at full capacity. Your request will be evaluated by members of the RC team to ensure that the resources in this partition will be used appropriately.

Note: All user limits are subject to the availability of cluster resources at the time of submission and will be honored according to that.

Name	Requires Approval?	Time limit (Default/Max)	Submitted Jobs	GPU per job Limit	Max GPUs per user Limit
gpu	No	4 hours/8 Hours	50/100	1	8
multi-gpu	**Yes**	4 hours/24 Hours	50/100	12	12

19.1 Requesting GPUs with Slurm

Use `srun` for interactive and `sbatch` for batch mode. The `srun` example below is requesting 1 node and 1 GPU with 4GB of memory in the `gpu` partition. You must use the `--gres=` option to request a gpu:

```
srun --partition=gpu --nodes=1 --pty --gres=gpu:1 --ntasks=1 --mem=4GB --time=01:00:00 /
↪bin/bash
```

Note: On the `gpu` partition, requesting more than 1 GPU (`--gres=gpu:1`) will cause your request to fail. Additionally, one cannot request all the CPUs on that gpu node as they are reserved for other GPUs.

The `sbatch` example below is similar to the `srun` example above, but it submits the job in the background, gives it a name, and directs the output to a file:

```
#!/bin/bash
#SBATCH --partition=gpu
#SBATCH --nodes=1
#SBATCH --gres=gpu:1
#SBATCH --time=01:00:00
#SBATCH --job-name=gpu_run
#SBATCH --mem=4GB
#SBATCH --ntasks=1
#SBATCH --output=myjob.%j.out
#SBATCH --error=myjob.%j.err

## <your code>
```

19.1.1 Specifying a GPU type

You can add a specific type of GPU to the `--gres=` option (with either `srun` or `sbatch`). For a list of available GPU types, refer to the GPU Types column in the table, at the top of this page, that are listed as `Public`. The following is an example for requesting a single p100 GPU:

```
--gres=gpu:p100:1
```

Note: Requesting a specific type of GPU could result in longer wait times, based on GPU availability at that time.

19.2 Using CUDA

There are several versions of CUDA Toolkits available on the HPC, including:

```
cuda/9.0
cuda/9.2
cuda/10.0
cuda/10.2
cuda/11.0
cuda/11.1
cuda/11.2
cuda/11.3
cuda/11.4
cuda/11.7
cuda/11.8
cuda/12.1
```

Use the `module avail` command to check for the latest software versions on Discovery. To see details on a specific CUDA toolkit version, use `module show`. For example, `module show cuda/11.4`.

To add CUDA to your path, use `module load`. For example, type `module load cuda/11.4` to load version 11.4 to your path.

Use the command `nvidia-smi` (NVIDIA System Management Interface) inside a GPU node to get the CUDA driver information and monitor the GPU device.

19.3 GPUs for Deep Learning

See also:

Deep learning frameworks tend to cost storage that can quickly surpass *Home Directory Storage Quota*: follow best practices for *Conda environments*.

First, log onto `gpu` interactively, and load anaconda and CUDA 11.8:

```
srun --partition=gpu --nodes=1 --gres=gpu:v100-sxm2:1 --cpus-per-task=2 --mem=10GB --
↪time=02:00:00 --pty /bin/bash
module load anaconda3/2022.05 cuda/11.8
```

Note: Be aware of compatibility regarding the GPU type: some installations do not work on k40m or k80 GPUs. To see what `non-Kepler` GPUs might be available execute the following command:

```
sinfo -p gpu --Format=nodes,cpus,memory,features,statecompact,nodelist,gres
```

This will indicate the state (idle or not) of a certain gpu-type that could be helpful in requesting an `idle` gpu. However, the command does not give real-time information of the state and should be used with caution.

Select the tab with the desire deeplearning framework.

Warning: Each tab assumes you are on a GPU node before with CUDA 11.8 and anaconda modules loaded as done above.

PyTorch

The following example demonstrates how to build PyTorch inside a conda virtual environment for CUDA version 11.8.

Listing 1: PyTorch's installation steps (with a specific GPU-type):

```
conda create --name pytorch_env python=3.10 -y
source activate pytorch_env
conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia -y
```

Now, let's check the installation:

```
python -c 'import torch; print(torch.cuda.is_available())'
```

If CUDA is detected by PyTorch, you should see the result, `True`.

See also:

PyTorch documentation for the most up-to-date instructions and for different CUDA versions.

TensorFlow

We recommend that you use CUDA 11.8 (the latest supported version) when working on a GPU with the latest version of TensorFlow (TF).

See also:

Compatibility of CUDA and TensorFlow versions, and detailed installation instructions.

For the latest installation, use the TensorFlow pip package, which includes GPU support for CUDA-enabled devices:

```
conda create --name TF_env python=3.9 -y
source activate TF_env
conda install -c "nvidia/label/cuda-11.8.0" cuda-toolkit -y
pip install --upgrade pip
pip install tensorflow==2.13.*
```

Verify the installation:

```
python3 -c 'import tensorflow as tf; print(tf.test.is_built_with_cuda())' # True
```

Note: Ignore the `Warning` messages that get generated after executing the above commands.

PyTorch + TensorFlow

```
conda create --name deeplearning-cuda11_8 python=3.9 -y
source activate deeplearning-cuda11_8
conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia -y
conda install -c "nvidia/label/cuda-11.8.0" cuda-toolkit -y
pip install --upgrade pip
pip install tensorflow==2.13.*
```

Verify installation:

```
python -c 'import torch; print(torch.cuda.is_available())' # True
python3 -c 'import tensorflow as tf; print(tf.test.is_built_with_cuda())' # True
```

Tip: Install `jupyterlab` and few other commonly used datascience packages in the `pytorch_env` environment:

```
conda install pandas scikit-learn matplotlib seaborn jupyterlab -y
```

CHAPTER

TWENTY

TRANSFERRING DATA

The HPC has a dedicated transfer node that you must use to transfer data to and from the cluster. You are not allowed to transfer data from any other node to or from the HPC to your local machine. The node name is `<username>@xfer.discovery.neu.edu:` where `<username>` is your Northeastern username to login into the transfer node.

You can also transfer files using Globus. This is highly recommended if you need to transfer large amounts of data. See *Using Globus* for more information.

If you are transferring data from different directories on the HPC, you need to use a compute node (see *Interactive Jobs: srun Command* or *Batch Jobs: sbatch*) with scp, rsync, or with the copy command to complete this tasks. You should use the `--constraint=ib` flag (see *Hardware Overview*) to ensure the fastest data transfer rate.

Caution: The `/scratch` space is for temporary file storage only. It is not backed up. If you have directed your output files to `/scratch`, you should transfer your data from `/scratch` to another location as soon as possible. See *Data Storage Options* for more information.

20.1 Transfer via Terminal

SCP

You can use `scp` to transfer files/directories to and from your local machine and the HPC. As an example, you can use this command to transfer a file to your `/scratch` space on the HPC from your local machine:

```
scp <filename> <username>@xfer.discovery.neu.edu:/scratch/<username>
```

where `<filename>` is the name of the file in your current directory you want to transfer and `<username>` is your Northeastern username. Note, this command is run on your local machine.

If you want to transfer a directory in your `/scratch` called `test-data` from the HPC to your local machine's currenting working directory, an example of that command would be:

```
scp -r <username>@xfer.discovery.neu.edu:/scratch/<username>/test-data .
```

where `-r` flag is for the recurssive transfer because it is a directory. Note, this command is run on your local machine.

Rsync

You can use the `rsync` command to transfer data to and from the HPC and local machine. You can also use `rsync` to transfer data from different directories on the cluster.

The syntex of `rsync` is

```
rsync [options] <source> <destination>
```

An example of using rsync to transfer a directory called `test-data` in your current working directory on your local machine to your `/scratch` on the HPC is

```
rsync -av test-data/ <username>@xfer.discovery.neu.edu:/scratch/<username>
```

where this command is run on your local machine in the directory that contains `test-data`.

Similarily, `rsync` can be used to copy from the current working directory on the HPC to your current working directory on your local machine:

```
rsync -av <username>@xfer.discovery.neu.edu:/scratch/<username>/test-data .
```

where this command is run on your local machine in the current directory that you want to save the directory `test-data`.

You can also use rsync to copy data from different directories on the HPC:

```
srun --partition=short --nodes=1 --ntasks=1 --time=01:05:00 --constraint=ib --pty /bin/
↪bash
rsync -av /scratch/<username>/source_folder /home/<username>/destination_folder
```

sbatch

You can use an sbatch job to complete data transfers by submitting the job to the HPC queue. An example of using `rsync` through an sbatch script is as follows:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=2
#SBATCH --time=0:05:00
#SBATCH --job-name=DataTransfer
#SBATCH --mem=2G
#SBATCH --partition=short
#SBATCH --constraint=ib
#SBATCH -o %j.out
#SBATCH -e %j.err

rsync -av /scratch/<username>/source_folder /home/<username>/destination_folder
```

where we are transfering the data from `source_folder` to the `destination_folder`.

SSHFS

If you want to use `sshfs`, you will need to use it with the dedicated transfer node `xfer.discovery.neu.edu`. It will not work on the login or compute nodes. On a Mac, you will also have to install macFUSE and sshfs (please refer to macFUSE) to use the `sshfs` command.

Use this syntax to perform file transfers with `sshfs`:

```
sshfs <username>@xfer.discovery.neu.edu:</your/remote/path> <your/local/path> -<options>
```

For example, this will mount a directory in your `/scratch` named `test-data` to a local directory on your machine `~/mount_point`:

```
sshfs <username>@xfer.discovery.neu.edu:/scratch/<username>/test-data ~/mount_point
```

where you can interact with the directory from your GUI or using the terminal to perform tasks on it.

20.2 Transfer via GUI Application

OOD's File Explorer

You can use OOD's File Explorer application to transfer data from different directories on the HPC and also to transfer data to and from your local machine to the HPC. For more information to complete this please see *OOD File Explorer*.

MobaXterm

You can use MobaXterm to transfer data to and from the HPC. Please checkout MobaXterm to download MobaXterm.

1. Open MobaXterm.
2. Click **Session**, then select **SFTP**.
3. In the **Remote host** field, type `xfer.discovery.neu.edu`
4. In the **Username** field, type your Northeastern username.
5. In the **Port** field, type 22.
6. In the **Password** box, type your Northeastern password and click **OK**. Click **No** if prompted to save your password.

You will now be connected to the transfer node and can transfer files through MobaXterm. Please refer to MobaXterm for further information.

FileZilla

You can use FileZilla to transfer data to and from the HPC. Please checkout FileZilla to download MobaXterm.

1. Open FileZilla.
2. In the **Host** field, type `sftp://xfer.discovery.neu.edu`
3. In the **Username** field, type your Northeastern username.
4. In the **Password** field, type your Northeastern password.
5. In the **Port** field, type 22.

You will now be connected to the transfer node and can transfer files through FileZilla. Please refer to FileZilla for further information.

CHAPTER

TWENTYONE

USING GLOBUS

Globus is a data management system that you can use to transfer and share files. Northeastern has a subscription to Globus, and you can setup a Globus account with your Northeastern credentials. If you have another account, either personal or through another institution, you can also link your accounts.

To use Globus, you must first set up an account as detailed below. Then, you must install Globus Connect to create an endpoint on your local computer, as also detailed below. After you have completed these two initial setup procedures, you can then use the Globus web app to perform file transfers. See *Using the Northeastern endpoint* for a walkthrough of using the Northeastern endpoint on Globus.

21.1 Globus Account Set Up

Use the following instructions to setup an account with Globus using your Northeastern credentials.

1. Go to Globus.
2. Click **Log In**.
3. From the Use your existing organizational login, select **Northeastern University**, and then click **Continue**.
4. Enter your Northeastern username and password.
5. If you do not have a previous Globus account, click **Continue**. If you have a previous account, click Link to existing account.
6. Check the agreement checkbox, and then click **Continue**.
7. Click **Allow** to give Globus permissions to access your files.

You will then be able to access the Globus File Manager app.

Tip: If you received an account identity that includes your NUID number (for example 000123456@northeastern.edu), you can follow the “Creating and linking a new account identity” instructions below to get a different account identity if you want a more user-friendly account identity. You can then link the two accounts together.

21.1.1 Creating and linking a new account identity (Optional)

If you created an account through the Northeastern University existing organizational login and received a username that includes your NUID, you can create a new identity with a different username and then link the two accounts together. A username that you have selected, as opposed to one with your NUID, can make it easier for you to remember your login credentials.

1. Go to Globus.
2. Click **Log In**.
3. Click **Globus ID** to sign in.
4. Click **Need a Globus ID? Sign up**.
5. Enter your Globus ID information.
6. Enter the verification code that Globus sends to your email.
7. Click **Link to an existing account** to link this new account with your primary account.
8. Select Northeastern University from the dropdown box, and click **Continue** to be taken to the Northeastern University single sign on page.
9. Enter your Northeastern username and password.

You should now be able to see your two accounts linked in the Account section on the Globus web app.

21.2 Install Globus Connect Personal (GCP)

Use Globus Connect Personal (GCP) to use your personal laptop as an endpoint. You first need to install GCP using the following procedure. You need to be logged into Globus before you can install GCP.

1. Go to Globus File Manager.
2. Enter a name for your endpoint in the Endpoint Display Name field.
3. Click **Generate Setup Key** to generate a setup key for your endpoint.
4. Click the **Copy** icon next to the setup key that was generated to copy the key to your clipboard. You will need this key during the installation of GCP in step 6.
5. Click the appropriate OS icon for your computer to download the installation file.
6. After the installation file has been downloaded to your computer, double click on the file to launch the installer.

Accept the defaults on the install wizard. After the install completes, you can now use your laptop as an endpoint within Globus.

Note: You can't modify an endpoint after you have created it. If you need an endpoint with different options, you'll need to completely delete the endpoint and recreate it. Follow the instructions on the Globus website for deleting and recreating an endpoint.

21.3 Working with Globus

After you have an account and set up a personal endpoint using Globus Connect personal, you can perform basic file management tasks using the Globus File Manager interface such as transferring files, renaming files, and creating new folders. You can also download and use the Globus Command Line Interface (CLI) tool. Globus also has extensive documentation and training files for you to practice with.

21.3.1 Using the Northeastern endpoint

To access the Northeastern endpoint on Globus, on the Globus web app, click **File Manager**, then in the **Collection** text box, type Northeastern. The endpoints owned by Northeastern University display in the collection area. The general Northeastern endpoint is `northeastern#discovery`. Using the File Manager interface, you can easily change directories, switch the direction of transferring to and from, and specify options such as transferring only new or changed files. Below is a procedure for transferring files from Discovery to your personal computer, but with the flexibility of the File Manager interface, you can adjust the endpoints, file view, direction of the transfer, and many other options.

To transfer files from Discovery to your personal computer, do the following

1. Create an endpoint on your computer using the procedure above "Install Globus Connect", if you have not done so already.
2. In the File Manager on the Globus web app, in the **Collections** textbox, type Northeastern, then in the collection list click the `northeastern#discovery` endpoint.
3. In the right-pane menu, click **Transfer or Sync to**.
4. Click in the **Search** text box, and then in on the **Your Collections** tab, click the name of your personal endpoint. You now can see the list of your files on Discovery on the left and a list of your files on your personal computer on the right.
5. Select the file or files from the right-side list of Discovery files that you want to transfer to your personal computer.
6. Select the destination folder from the left-side list of the files on your personal computer.
7. (Optional) Click **Transfer & Sync Options** and select the transfer options that you need.
8. Click **Start**.

21.3.2 Connecting to Google Drive

The version of Globus currently on Discovery allows you to connect to Google Drive by first setting up the connection in GCP. This will add your Google Drive to your current personal endpoint. You'll need to first have a personal endpoint, as outlined in the procedure above.This procedure is slightly different from using the Google Drive Connector with Globus version 5.5. You'll need your Google Drive downloaded to your local computer.

To add Google Drive to your personal endpoint, do the following

1. Open the GCP app. On Windows, right click on the **G** icon in your taskbar and select **Options**. On Mac, click the **G** icon in the menu bar and select **Preferences**.
2. On the **Access** tab, click the + button to open the **Choose a directory** dialog box.
3. Navigate to your Google Drive on your computer and click **Choose**.
4. Click the **Shareable** checkbox to make this a shareable folder in Globus File Manager, and then click **Save**.

You can now go to Globus File Manager and see that your Google Drive is available as a folder on your personal endpoint.

21.3.3 Command Line Interface (CLI)

The Globus Command Line Interface (CLI) tool allows you to access Globus from a command line. It is a stand-alone app that requires a separate download and installation. Please refer to the Globus CLI documentation for working with this app.

21.3.4 Globus documentation and test files

Globus provides detailed instructions on using Globus and also has test files for you to practice with. These are free for you to access and use. We encourage you to use the test files to become familiar with the Globus interface. You can access the Globus documentation and training files on the Globus How To website.

CHAPTER

TWENTYTWO

DATA STORAGE OPTIONS

RC is responsible for the procurement and ongoing maintenance of several data storage options, including active, and archive storage solutions. If you are affiliated with Northeastern, you can request one or more storage solutions to meet your storage needs. If you anticipate needing storage as part of a grant requirement, please schedule a storage consultation with an RC staff member to understand what storage options would best meet your research needs.

22.1 Active Storage

There are two main storage systems connected to Northeastern's HPC cluster: `/home` and `/scratch`; these options have specific quotas and limitations. The list below details the storage options available to you on the HPC cluster by having an account. Every individual with an account has both a `/home` and `/scratch`. While research groups can request additional storage on the `/work` storage system, `/work` storage is not currently provisioned to individuals.

Important: The `/scratch` space is only for temporary storage; this storage is not backed up, and there is a purge policy for data older than 28 days. Please review the `/scratch` policy on our Policy page.

$HOME: `/home/<username>` where `username` is your NU login, e.g., `/home/j.smith`

- **Description:** All users are given a `/home` automatically when their account is created. This storage is mainly intended for storing relatively small files such as script files, source code, and software installation files. While `/home` is permanent storage that is backed up and replicated, `/home` is not performant storage. `/home` also has a small quota, so you should frequently check your space usage (use a command such as, `du -h /home/<yourusername>` where `<yourusername>` is your username, to see the total space usage). For running jobs and directing output files, you should use your `/scratch`.
- **Quota:** 75GB

Scratch: `/scratch/<username>`

- **Description:** All users are given a `/scratch` automatically when their account is created. Scratch is a shared space for all users. The total storage available is 1.8PB; however, while this is performant storage, it is for temporary use only. **It is not backed up.** Data on `/scratch` should be moved as soon as possible to another location for permanent storage. You should run your jobs from and direct output to your `/scratch` for best performance. However, it is best practice to move your files off of scratch to avoid any potential data loss.
- **Quota:** N/A

Work: `/work/<groupname>`

- **Description:** Research groups can request additional storage on `/work`. A PI can request this extra storage through the New Storage Space request. This is a performant, persistent, and long-term storage that is meant for storing data being actively used for research. It can be accessed by all members of the research group who have necessary access permissions, facilitating collaboration and seamless sharing of data within the group.

- **Quota:** Each group can request up to **35TB** of free storage across all supplemental storage tiers: `/work/<groupname>` and `/nese`.
- **Access Request:** Students with research groups can request access to the PI's storage on `/work`. To expedite the request process, we recommend that you inform the group owner they will be receiving an email requesting their permission to grant you access to `/work` before you submit the request.

1. To request access to `/work`, students can either create a ServiceNow ticket with RC or email rchelp@northeastern.edu to automatically generate a ticket in ServiceNow. Please include both the storage space name and the PI's name.
2. Once you have been added to the unix group for the space on `/work`, please ensure to close all open connections to the HPC and login again for the changes to reflect on your end. Please note that UNIX groups are assigned at login time, and this step ensures that your access privileges are updated accordingly. To confirm you have been added to the group, you can run the command `groups`.

- **Default Permission:** By default, users are given read and write access when added to `/work`. However, specific permissions might be granted at the PI's request.

Attention: The `/research` storage tier is no longer provided. Please contact Research Computing if you are a former user of `/research` and have questions or issues related to `/research` by submitting a ticket. Other storage options include `/work`, Sharepoint, and OneDrive.

22.2 Archival Storage

Important: If you are not connected to the campus internet, you must be connected to the university's VPN (GlobalProtect) before you can access the `/nese` system. You can find detailed information about downloading and using the GlobalProtect VPN in the FAQ: VPN and remote access.

NAME: `/nese`

- **Description:** This is archival, non-performant storage intended for researchers who need to have a long-term storage option for their data.
- **Quota:** Each group can request up to **35TB** of free storage across all supplemental storage tiers: `/work/<groupname>` and `/nese`.

CHAPTER

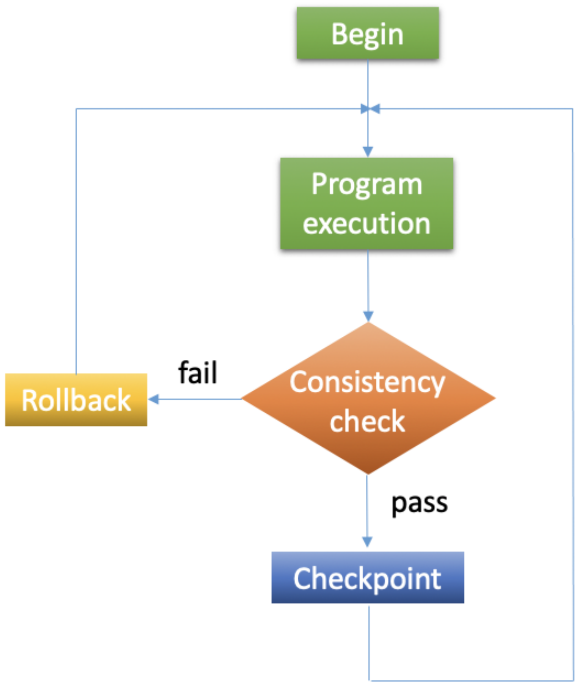
TWENTYTHREE

CHECKPOINTING JOBS

The complexity of HPC systems can introduce unpredictable behavior in hardware or software components, leading to job failures. Applying fault tolerance techniques to your HPC workflows can make your jobs more resilient to crashes, partition time limits, and hardware failures.

23.1 The Checkpointing technique

Checkpointing is a fault tolerance technique based on the Backward Error Recovery (BER) technique, designed to overcome "fail-stop" failures (interruptions during the execution of a job).



To implement checkpointing:

- Use data redundancy to create checkpoint files, saving all necessary calculation state data. Checkpoint files are generally created at constant time intervals during the run.
- If a failure occurs, start from an error-free state, check for consistency, and restore the algorithm to the previous error-free state.

Checkpointing allows you to:

- Create resilient workflows in the event of faults.
- Overcome most scheduler resource time limitations.
- Implement an early error detection approach by inspecting intermediate results.

23.2 Checkpointing types

Checkpointing can be implemented at different levels of your workflow:

- **Application-level** - This is the recommended approach for most Discovery users. Utilize the checkpointing tool that is already available in your software application. For example, most software designed for HPC has a checkpointing option, and information on proper usage is often available in the software user manual.
- **User-level** - This approach is suitable if you develop your code or possess sufficient knowledge of the application code to integrate checkpointing techniques effectively. We recommend this approach for some Discovery users with advanced proficiency and familiarity with checkpointing mechanisms.
- **System-level** - Checkpointing is done on the system side, where the user saves the state of the entire process. This option is less efficient than User-level or Application-level checkpointing as it introduces a lot of redundancy.
- **Model-level** - This approach is suitable for saving a model's internal state (its weights, current learning rate, etc.) so that the framework can resume the training from this point whenever desired. This is often the intent of users doing machine learning on Discovery.

23.2.1 Which checkpoint type to use?

There are several options for checkpointing, depending on the needs of your software.

- If your software already includes **built-in checkpointing**, this is often the preferred option, as it is the most optimized and efficient way to checkpoint.
- **Application-level** checkpointing is the easiest to use, as it exists within your application. It does not require significant changes to your scripts and saves only the relevant data for your specific application.
- **User-level** checkpointing is recommended if you are writing your own code. You can use DMTCP or implement your own checkpointing.
- **ML Model-level** checkpointing is specific to model training and deployment, as detailed in the `ML Model-level_` section.

Note: If you are developing in Python, Matlab or R, there are packages that can be used to implement checkpointing. Some examples include Python PyTorch checkpointing, TensorFlow checkpointing, MATLAB checkpointing and R checkpointing. Additionally, many Computational Chemistry and Molecular Dynamics software have built-in checkpointing options (e.g., GROMACS and LAMMPS).

Implementing checkpointing can be achieved by the following:

- Some save-and-load mechanism of your calculation state.
- The use of Slurm Job Arrays.

Note: To overcome partition time limits, replace your single long job with multiple shorter jobs. Then, use job arrays to set each job to run one after the other. If checkpointing, each job will write a checkpoint file. The following job will use the latest checkpoint file to continue from the latest state of the calculation.

23.3 Application-level checkpointing

23.3.1 GROMACS checkpointing example

The following example shows how to implement a 120-hour GROMACS job using multiple shorter jobs on the *short* partition. We use Slurm job arrays and the GROMACS built-in checkpointing option to implement checkpointing.

See also:

https://manual.gromacs.org/documentation/current/user-guide/managing-simulations.html

The following script `submit_mdrun_array.sh` creates a Slurm job array of 10 individual array jobs:

```
#!/bin/bash
#SBATCH --partition=short
#SBATCH --constraint=cascadelake
#SBATCH --nodes=1
#SBATCH --time=12:00:00
#SBATCH --job-name=myrun
#SBATCH --ntasks=56
#SBATCH --array=1-10%1  # execute 10 array jobs, 1 at a time
#SBATCH --output=myrun-%A_%a.out
#SBATCH --error=myrun-%A_%a.err

module load cuda/10.2
module load gcc/7.3.0
module load openmpi/4.0.5-skylake-gcc7.3
module load gromacs/2020.3-gpu-mpi
source /shared/centos7/gromacs/2020.3-gcc7.3/bin/GMXRC.bash

srun --mpi=pmi2 -n $SLURM_NTASKS gmx_mpi mdrun -ntomp 1 -s myrun.tpr -v -dlb yes -cpi␣
↪state
```

The script above sets checkpoint flag `-cpi state` preceding the filename to dump checkpoints. This directs `mdrun` to checkpoint to `state.cpt` when loading the state. The Slurm option `--array=1-10%1` creates 10 Slurm array tasks, and runs one task job serially for 12 hours. The variable `%A` denotes the main job ID, while `%a` denotes the task ID (i.e., spanning `1-10`).

To submit this array job to the scheduler, use the following command:

```
sbatch submit_mdrun_array.bash
```

23.3.2 DMTCP checkpoint example

DMTCP (Distributed MultiThreaded checkpointing) is a tool checkpoints without changing code. It works with most Linux applications such as Python, Matlab, R, GUI, and MPI.

The program runs in the background of your program, without significant performance loss, and saves the process states into checkpoint files. DMTCP is available on the cluster

```
module avail dmtcp
module show dmtcp
module load dmtcp/2.6.0
```

Since DMTCP runs in the background, it requires some changes to your shell script. See examples of checkpointing with DMTCP, which use DMTCP with a simple C++ program (scripts modified from RSE-Cambridge).

23.3.3 Application-level checkpointing tips

What data to save?

- Non-temporary application data
- Any application data that has been modified since the last checkpoint
- Delete checkpoints that are no longer useful - keep only the most recent checkpoint file.

How frequently to checkpoint?

- Too often – will slow down your calculation, maybe I/O heavy and memory-limited.
- Too infrequently – leads to large/long rollback times.
- Consider how long it takes to checkpoint and restart your calculation.
- In most cases a rate of every 10-15 minutes is ok.

23.4 ML Model-level Checkpointing

Model-level checkpointing is a technique used to periodically save the state of a machine learning (ML) model during training, enabling the training process to get resumed from the saved checkpoint in case of interruptions or premature termination. The saved state typically includes the model's parameters, optimizer state, and essential training information (e.g., the epoch number and loss value). Model checkpoints are especially critical for long-running training jobs.

23.4.1 Why is Checkpointing Important in Deep Learning?

Checkpointing is crucial in deep learning because the training process can be time-consuming and require significant computational resources. Additionally, the training process may be interrupted due to hardware or software issues. Checkpoints solve this problem by saving the model's current state to resume from where it left off.

Moreover, checkpointing also saves the best-performing model, which can then load for evaluation. For instance, the model's performance can vary based on the initialization and optimization algorithm, so checkpointing provides a way to select the best model based on a performance metric.

In summary, checkpointing is essential in deep learning as it provides a way to save progress, resume training, and select the best-performing model.

23.4.2 TensorFlow checkpoint example

The following example demonstrates how to implement a longer ML job using the *tf.keras* checkpointing API and multiple shorter Slurm job arrays on the gpu partition.

The following example of the `submit_tf_array.bash` script:

```
#!/bin/bash
#SBATCH --job-name=myrun
#SBATCH --time=00:10:00
#SBATCH --partition=gpu
#SBATCH --nodes=1
#SBATCH --gres=gpu:1
#SBATCH --mem=10Gb
#SBATCH --output=%A-%a.out
#SBATCH --error=%A-%a.err
#SBATCH --array=1-10%1  #execute 10 array jobs, 1 at a time.

module load miniconda3/2020-09
source activate tf_gpu

# Define the number of steps based on the job id:
numOfSteps=$(( 500 * SLURM_ARRAY_TASK_ID ))

# Run python and save outputs to a log file corresponding to the current job task ID
python train_with_checkpoints.py $numOfSteps &> log.$SLURM_ARRAY_TASK_ID
```

The following checkpoint example is given in the program `train_with_checkpoints.py`:

```
checkpoint_path = "training_2/{epoch:d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)
cp_callback = tf.keras.callbacks.ModelCheckpoint(
   filepath=checkpoint_path,
   verbose=1,
   save_weights_only=True,
   period=5)
```

See scripts, which are modified from TensorFlow Save and load models.

The Slurm option `--array=1-10%1` will create 10 Slurm array tasks, and will run one task job at a time. Note that the saved variable `%A` denotes the main job ID, while variable `%a` denotes the task ID (spanning values 1-10). Note that also the output/error files are unique in order to prevent different jobs writing to the same files. The Shell variable `SLURM_ARRAY_TASK_ID` holds the unique task ID value and can be used within the Slurm Shell script to point to different files or variables.

To submit this job to the scheduler, use the command:

```
sbatch submit_tf_array.bash
```

23.4.3 PyTorch checkpoint example

See also:

GPUs for Deep Learning

Setting up a Sample Project

Next, we will set up a sample project to demonstrate checkpointing in PyTorch. We will create a simple neural network in PyTorch and train it on a sample dataset. The following code creates a simple neural network in PyTorch:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128)
        self.fc2 = nn.Linear(128, 10)
    def forward(self, x):
        x = x.view(-1, 28 * 28)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Once the neural network is defined, we can load a sample dataset and train the model. In this example, we will use the MNIST dataset, which is available in the `torchvision` library. The following code loads the MNIST dataset and trains the model:

```
import torchvision
import torchvision.transforms as transforms

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True,␣
↪transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)
net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

for epoch in range(2):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
```

(continues on next page)

(continued from previous page)

```
        optimizer.step()
        running_loss += loss.item()
    print('Epoch %d loss: %.3f' % (epoch + 1, running_loss / len(trainloader)))
```

With this simple project setup, we can demonstrate checkpointing in PyTorch.

What is the model state in PyTorch?

In PyTorch, the model state refers to the values of all the parameters, weights, and biases that define the model's behavior. This state is stored in the model's `state_dict`, which is a dictionary-like object that maps each layer to its parameter tensors. Additionally, the `state_dict` contains information about the model's architecture and the values of the parameters learned during training.

The `state_dict` can be saved to a file using PyTorch's `torch.save` function and can be loaded back into memory using `torch.load`. This allows for checkpointing, which saves the state of a model periodically during training so that it can be recovered in the case of a crash or interruption.

PyTorch also provides the ability to save and load the entire model, including the model's architecture and optimizer state, using the `torch.save` and `torch.load` functions. This allows for complete checkpointing of a model's state.

It's important to note that the `state_dict` only contains information about the model's parameters, not the optimizer or other training-related information. Therefore, to save the optimizer state, you must also save it separately and load it back when loading the model.

In conclusion, the model state in PyTorch represents the values of all the parameters, weights, and biases that define the model's behavior. The state is stored in the `state_dict` and can be saved and loaded for checkpointing purposes. By saving the model state periodically during training, you can ensure that your progress is not lost in the case of a crash or interruption.

Saving a PyTorch Model

Saving a Model's `state_dict`

To save the learnable parameters of a PyTorch model and the optimizer, we need to save the model's **`state_dict`**. We can use the **`torch.save`** function and pass the state_dict and a file name as arguments. The following code demonstrates how to save the state_dict of the model:

```
checkpoint_path = 'checkpoint.pth'
state = {'epoch': epoch + 1,
         'state_dict': net.state_dict(),
         'optimizer': optimizer.state_dict()}
torch.save(state, checkpoint_path)
```

Saving the Entire Model

In addition to saving the state_dict, we can also save the entire model, which includes the architecture, the learnable parameters, and the optimizer state. We can use the `torch.save()` function to save the entire model by passing the model and a filename as arguments. The following code demonstrates how to save the entire model:

```
model_path = 'model.pth'
torch.save(net, model_path)
```

Note: When saving the entire model, the custom classes used to define the model must be importable, either as a part of the project or in a separate file that can be imported.

We have seen how to save the state_dict and the entire model in PyTorch. The following section will show how to load a saved model.

Loading a PyTorch Model

Loading a PyTorch `state_dict`

We can use the **`torch.load`** function to load the `state_dict` of a saved model by passing the file name as an argument. After loading the `state_dict`, we can set the `state_dict` of the model using the **`model.load_state_dict`** method. The following code demonstrates how to load the `state_dict` of a saved model:

```
codecheckpoint = torch.load(checkpoint_path)
net.load_state_dict(checkpoint['state_dict'])
optimizer.load_state_dict(checkpoint['optimizer'])
epoch = checkpoint['epoch']
```

Loading the Entire PyTorch Model

To load the entire model, use the **`torch.load`** function and pass the file name as an argument. The loaded model can then be assigned to a variable, as shown in the following code:

```
loaded_model = torch.load(model_path)
```

In PyTorch, we can load a saved model's `state_dict` or the entire model. This section will cover how to resume training from a checkpoint.

Resuming Training from a Checkpoint

To resume training from a checkpoint, we need to load the model's `state_dict` and the state of the optimizer (see *Loading a PyTorch Model*). After loading these, we can continue the training process from where it was left off. The following code demonstrates how to resume training from a checkpoint:

```
checkpoint = torch.load(checkpoint_path)
net.load_state_dict(checkpoint['state_dict'])
optimizer.load_state_dict(checkpoint['optimizer'])
epoch = checkpoint['epoch'] # Continue training
for epoch inrange(epoch, num_epochs): # Train the model
```

(continues on next page)

(continued from previous page)

```
    # Save the checkpoint
    state = {'epoch': epoch + 1,
             'state_dict': net.state_dict(),
             'optimizer': optimizer.state_dict()
             }
torch.save(state, checkpoint_path)
```

This code loads the `state_dict` and the optimizer's state from the checkpoint file, and resumes training from the `epoch` value saved in the checkpoint file. After each epoch, the model's `state_dict` and the optimizer's state are saved in the checkpoint file, as described in Section III. This shows how to checkpoint PyTorch models, save and load the `state_dict`, save and load the entire model, and resume training from a checkpoint.

23.4.4 Model-level tips and tricks

Save only the model's State_dict

Save only the model's state_dict and optimizer state, allowing us to save only information needed to resume training. By this, we reduce the checkpoint file's size and make it easier to load the model. Hence, avoid keeping unnecessary information in the checkpoint file, such as irrelevant metadata or tensors we can define during training.

Save regularly

To prevent losing progress in case of a crash or interruption, save the checkpoint file regularly (i.e., after each epoch).

Save to multiple locations

Save the checkpoint file to multiple locations, such as a local drive and the cloud, to ensure that the checkpoint is recovered in case of failure.

Use the latest versions of libraries

Use the latest version of PyTorch and other relevant libraries is vital; changes in these libraries may cause compatibility issues with older checkpoints. With these best practices, you can ensure that your PyTorch models are saved efficiently and effectively and that your progress is not lost in case of a crash or interruption.

Naming conventions

Use a consistent naming convention for checkpoint files; include information such as the date, time, and epoch number in the file name to make tracking multiple checkpoint files easier and ensure you are choosing the proper checkpoint to load.

Checkpoint Validation

Validate the checkpoint after loading it to ensure that the model's state_dict and the state of the optimizer are correctly loaded by making a prediction using the loaded model and checking that the results are as expected.

Periodic clean-up

Periodically, remove old checkpoint files to avoid filling up storage. This can be done by keeping only the latest checkpoint or keeping only checkpoint files from the last few epochs.

Document checkpoints

Document the purpose of each checkpoint and what it contains: include the model architecture, the training data, the hyperparameters, and the performance metrics. This will help to keep track of the progress and make it easier to compare different checkpoints. With these additional best practices, you can ensure that your checkpointing process is efficient, effective, and well-organized.

CHAPTER

TWENTYFOUR

HOME DIRECTORY STORAGE QUOTA

There are strict quotas for each home directory (i.e., /home/<username>), and staying within the quota is vital for preventing issues on the HPC. This page provides some best practices for keeping within the quota. For more information about data storage on the HPC, see *Data Storage Options*.

Important: All commands on this page should be run from a compute node because they are cpu intensive. You can find more information on getting a job on a compute node from *Interactive Jobs: srun Command*.

24.1 Analyze Disk Usage

From a compute node, run the following command from your /home/<username> directory:

```
du -shc .[^.]* ~/*
```

This command will output the size of each file, directory, and hidden directory in your /home/<username> space, with the total of your /home directory being the last line of the output. After identifying the large files and directories, you can move them to the appropriate location, (e.g., /work for research) or you can back up and delete the files and directories if they are no longer required. An example output would look like:

```
[<username>@<host> directory]$  du -shc .[^.]* ~/*
39M     .git
106M    discovery-examples
41K     README.md
3.3M    software-installation
147M    total
```

24.2 Utlilize /work and /scratch

Use /work for long-term storage. PIs can request a folder in /work via New Storage Space Request, and they can request additional storage via Storage Space Extension Request. Utilize /scratch/<username> for temporary or intermediate files. Then, move files from /scratch to /work for persistent storage (i.e., the recommended work flow).

Note: Please be mindful of the /scratch purge policy, which can be found on the Research Computing Policy Page. See *Data Storage Options* for information on /work and /scratch.

24.3 Cleaning Directories

24.3.1 Conda

Note: Conda environments are part of your research and should preferably be stored in your PI's `/work` directory.

Here are some suggestions to reduce the storage size of the environments for those using the `/home/<username>/.conda` directory.

Remove unused packages and clear caches of conda by loading an anaconda module and running the following:

```
source activate <your environment>
conda clean --all
```

This will only delete unused packages in your `~/.conda/pkgs` directory.

To remove any unused conda environments, run:

```
conda env list
conda env remove --name <your environment>
```

24.3.2 Singularity

If you have pulled any containers to the HPC using Singularity, you can clean your container cache in your `/home/<username>` directory by running the following command from a compute node:

```
module load singularity/3.5.3
singularity cache clean all
```

To avoid your `~/.singularity` directory filling up, you can set a temporary directory for when you pull containers to store the cache in that location; an example of this procedure, (where `<project>` is your PI's `/work` directory) is the following:

```
mkdir /work/<project>/singularity_tmp
export SINGULARITY_TMPDIR=/work/<project>/singularity_tmp
```

Then, pull the container using singularity as usual.

24.3.3 Cache

The `~/.cache` directory can become large over time with general use of the HPC and Open OnDemand. Make sure you are not running any processes or jobs at the time by running the following:

```
squeue -u <username>
```

which prints a table with `JOBID`, `PARTITION`, `NAME`, `USER ST`, `TIME`, `NODES`, and `NODELIST (REASON)` which is empty when no jobs are running (i.e., it is safe to remove `~/.cache` when no jobs are running).

24.4 Storing research environments

24.4.1 Conda environments

Use conda environments for python on HPC. To create an environment in `/work`, use the `--prefix` flag as follows: (where `<project>` is your PI's `/work` directory and `<my conda env>` is an empty directory to store your conda environment):

```
conda create --prefix=/work/<project>/<my conda env>
```

More information about creating custom conda environments can be found here *Using Conda*.

Utilize the same conda environment to save storage space and time (i.e., avoiding duplicate conda environments). Hence, shared environments can be easily done for a project accessing the same `/work` directory. For more information about creating custom conda environments, see *Using Conda*.

24.4.2 Singularity containers

Containers that are pulled, built and maintained for research work should be stored in your PI's `/work` directory, not in your `/home/<username>` directory.

CHAPTER

TWENTYFIVE

INTRODUCTION TO OOD

Open OnDemand (OOD) is a web portal to the Discovery cluster. A Discovery account is necessary for you to access OOD. If you need an account, see *Request an account*. If you already have an account, in a web browser, go to http://ood.discovery.neu.edu and sign in with your Northeastern username and password.

OOD provides you with a number of resources for interacting with the Discovery cluster:

- Launch a terminal that runs within your web browser without needing a separate terminal program. This is an advantage if you use Windows, as otherwise you need to download and use a separately installed program, such as MobaXterm.
- Use software applications, such as SAS Studio, that run in your browser without needing any further configuration. See *Using OOD's Interactive Apps* for more information.
- View, download, copy, and delete files using the File Explorer feature. See *Using OOD's FIle Explorer* for more information.

Note: OOD is a web-based application. You access it by using a web browser. Like many web-based applications, it has compatibility issues with certain web browsers. For optimal results, use OOD with newer versions of Chrome, Firefox, or Internet Explorer. OOD does not currently have support for Safari or mobile devices (phones and tablets).

CHAPTER

TWENTYSIX

OOD FILE EXPLORER

When working with the resources in OOD, your files are stored in your home directory on the storage space on the Discovery cluster. Similar to any file navigation system, you can work with your files and directories through the OOD Files feature as detailed below. For example, you can download a Jupyter Notebook file in OOD that you've been working on to your local hard drive, rename a file, or delete a file that you no longer need.

Note: Your home directory has a file size limit of 75GB. Be sure to check your home directory regularly, and remove any files you do not need to ensure you do not run out of space.

1. In a web browser, go to ood.discovery.neu.edu. If prompted, enter your myNortheastern username and password.
2. Select **Files > Home Directory**. The contents of your home directory display in a new tab.
3. To download a file to your hard drive, navigate to the file you want to download, select the file, and click **Download**. If prompted by your browser, click **OK** to save your file to your hard drive.
4. To navigate to another folder on the Discovery file system, click **Go To**, enter the path to the folder you want to access, and click **OK**.

Note: From the **Files > Home Directory** view, the **Edit** button will not launch your .ipynb file in a Jupyter Notebook. It will open the file in a text editor. You have to be in Jupyter Notebook in order to launch a .ipynb file from your /home directory. See *Using OOD's Interactive Apps* for how to access a Jupyter Notebook through OOD.

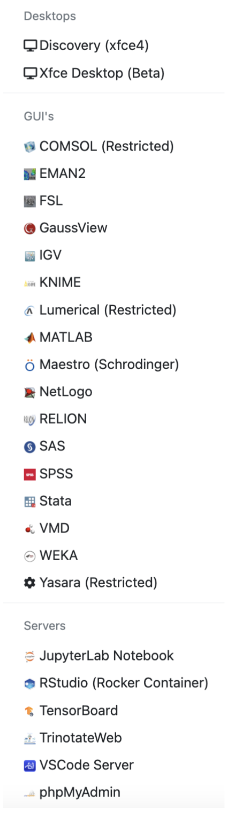
CHAPTER

TWENTYSEVEN

OOD INTERACTIVE APPS

The OOD web portal provides a range of applications. Upon clicking *launch*, the Slurm scheduler assigns a compute node with a specified number of cores and memory. By default, applications run for one hour. If you require more than an hour, you may have to wait for Slurm to allocate resources for the duration of your request.

27.1 Applications on OOD



The Open OnDemand interface offers several applications, which as of June 2023, include:

- JupyterLab
- RStudio (Rocker)
- Matlab
- Schrodinger (Maestro)

- Desktop
- Gaussian (Gaussview)
- KNIME
- Tensorboard
- SAS

These applications can be accessed from the OOD web interface's **Interactive Apps** dropdown menu.

Please note that specific applications in the **Interactive Apps** section, particularly those with graphical user interfaces (GUIs), may require X11 forwarding and the setup of passwordless SSH. For tips and troubleshooting information on X11 forwarding setup and usage, please refer to the Using X11 section of our documentation.

Additionally, we offer a selection of modified standard applications intended to support specific coursework. These applications are under the **Courses** menu on the OOD web interface. Please note that these course-specific applications are only accessible to students enrolled in the respective courses.

Note: Certain apps are reserved for specific research groups and are not publicly accessible, as indicated by the "Restricted" label next to the application name. If you receive an access error when attempting to open a restricted app, and you believe you should have access to it, please email rchelp@northeastern.edu with the following information: your username, research group, the app you are trying to access, and a screenshot of the error message. We will investigate and address the issue.

1. Go to Open On Demand in a web browser. If prompted, enter your myNortheastern username and password.
2. Select **Interactive Apps**, then select the application you want to use.
3. Keep the default options for most apps, then click **Launch**. You might have to wait a minute or two for a compute node to be available for your requested time and resource.

27.2 JupyterLab Notebook

JupyterLab Notebook is one of the interactive apps on OOD. This section will provide a walkthrough of setting up and using this app. The general workflow is to create a virtual Python environment, ensure that JupyterLab Notebook uses your virtual environment, and reference this environment when you start the JupyterLab Notebook OOD interactive app.

To find the JupyterLab Notebook on OOD, follow these steps:

1. Go to Open On Demand.
2. Click on Interactive Apps.
3. Select JupyterLab Notebook from the dropdown list.

The OOD form for launching JupyterLab Notebook will appear.

Conda virtual environment

You can import Python packages in your JupyterLab Notebook session by creating a conda virtual environment and activating that environment when starting a JupyterLab Notebook instance.

1. First, set up a virtual Python environment. See *Creating an Environment* for how to set up a virtual Python environment on the HPC using the terminal.

2. Type `source activate <yourenvironmentname>` where `<yourenvironmentname>` is the name of your custom environment.
3. Type `conda install jupyterlab -y` to install jupyterlab in your environment.

27.2.1 Using OOD to launch JupyterLab Notebook

1. Go to Open On Demand.
2. Click **Interactive Apps**, then select **JupyterLab Notebook**.
3. Enter your **Working Directory** (e.g., `/home/<username>` or `/work/<project>`) that you want JupyterLab Notebook to launch in.
4. Select from the **Partition** dropdown menu the partition you want to use for your session. Refer to *Partitions* for the resource restrictions for the different partitions. If you need a GPU, select the `gpu` partition.
5. Select the compute node features for the job:
   - In the **Time** field, enter the number of hour(s) needed for the job.
   - Enter the memory you need for the job in the **Memory (in Gb)** field.
   - If you selected the `gpu` partition from the dropdown menu, select the GPU you would like to use and the version of CUDA that you would like to use for your session under the respective dropdown menus.
6. Select the Anaconda version you used to create your virtual Python environment in the **System-wide Conda Module** field.
7. Check the **Custom Anaconda Environment** box, and enter the name of your custom virtual Python environment in the **Name of Custom Conda Environment** field.
8. Click **Launch** to join the queue for a compute node. This might take a few minutes, depending on your requested resources.
9. When you have been allocated a compute node, click **Connect to Jupyter**.

When your JupyterLab Notebook is running and open, type `conda list` in a cell and run the cell to confirm that the environment is your custom conda environment (you should see this on the first line). This command will also list all of your available packages.

27.3 Xfce Desktop (Beta)

This Open OnDemand application is a containerized desktop running on the HPC cluster. It has access to these tools/programs:

- Slurm (for running Slurm commands via the terminal in the desktop and interacting on compute nodes)
- Module command (for loading and running HPC-ready modules)
- File explorer (able to traverse and view files that you have access to on the HPC)
- Firefox web browser
- VLC media player
- Office Libre suite of applications (word processing, spreadsheets, presentation applications)

Note: The desktop application is a Singularity container: a Singularity container cannot run inside. It will fail if a user tries to run a module or program that runs a container inside this application.

CHAPTER

TWENTYEIGHT

CLASSROOM HPC: FAQ

There are several use cases for teaching and learning with the Discovery cluster in a classroom. Discovery offers both a command line and web interface, allowing flexibility in access and instruction level. Using Discovery gives you and your students access to many popular scientific applications, and also allows you and your students to install other packages as needed. The easy-to-use Open onDemand web portal also offers a built-in visual file explorer for file viewing and transfer.

The following Frequently Asked Questions section should help to answer most of your questions about how you can use the Discovery cluster for classroom use. You can also reach out to us at rchelp@northeastern.edu or submit a ServiceNow ticket if you need further information.

How can I use Discovery with my class?

There are several ways you can use Discovery in your classroom. Your class can use Discovery for accessing specific software packages and working environments, as well as learning how to utilize high performance computing (HPC) resources for large and complex data processing, such as machine learning; AI and molecular simulations; and more.

How do I get my class access to Discovery?

You fill out a Discovery Classroom Use Request ticket. The form asks you to submit a list of your students' names and emails. We will create accounts on Discovery for them. Follow the steps in the article How to Download a List of Student Email Addresses from Canvas to get a list of your students' emails. If your class roster is not complete, you can attach a preliminary list to the ticket, and then follow up with us on the same ticket with an updated roster when the add/drop period is over.

Is there any training on Discovery for my class?

Yes, we currently provide online training for your class on using Discovery. In the past, we have also given in-person presentations during a class period on the Boston campus. We hope to provide in-person training again in 2021. We can customize the training to focus on the resources you will be using with them for the class. Email us at rchelp@northeastern.edu and provide us with some details about your class and what training you would like us to provide.

Do my students have to learn Linux to work with Discovery?

Depending on your class assignments, many students can work with the Open onDemand web portal, which does not require any knowledge of Linux to use. In cases where you want them to work on the command line, they should have a basic understanding of Linux commands. Please see the question "Is there any training on Discovery for my class" above if you would like us to provide your class with a basic training course prior to using Discovery.

What software is available to use with my class on Discovery?

There are many software packages available, including popular software apps such as Jupyter Notebook, RStudio, and MATLAB. If you have an account on Discovery, you can see the list of available software by using the `module avail` command. See *Using Module* for more information. Students have access to all the modules on Discovery. They can also use the interactive apps available on Open onDemand. See *Introduction to Open OnDemand (OOD)* for more information. We can also install additional modules and libraries specifically for your class as needed.

My class needs access to a specific software application that I do not see installed on Discovery or Open onDemand (OOD). What should I do?

If you class requires software not currently installed on Discovery or OOD, follow the procedure below to request that software be installed on Discovery. You must be a professor or instructor to initiate this request. Students in your class should not make this request. If your students need a specific software application, you must complete the form for them. This is to ensure that we only get one request for the software. Students in one class often make multiple requests for the same software, so having all requests go through the professor or instructor reduces this overlap.

To request additional software (instructors only):

1. Go to the Discovery Cluster Software Request. If prompted, sign in to ServiceNow with your Northeastern username and password to access the form.
2. In the Sponsor's Name field, enter your name.
3. Make sure to follow the instructions on the form regarding either providing the URL of the open source software library or uploading the installation package in your home directory if it requires you to register it first. The request will not be completed without this information.
4. Select the acknowledgement checkbox, and click **Submit**.

Note: Software requests can take up to 24 hours to verify, and then additional time is needed to complete the installation. We might not be able to install every software application requested. If this is the case we will notify you and try to provide alternative software to meet your needs.

Tip: You and your students can install software locally to your PATH on Discovery, which may be a better option in some cases, such as installing multiple conda environments. See *Software Overview* for more information.

I just need my class to access Open onDemand. How do I request that?

Open onDemand is a web portal that lets you access the resources on Discovery through an easy-to-navigate web browser interface. You request course access the same way you would to get access to Discovery, as outlined above. Please specify that you'd like your course to be on Open onDemand for your students, in addition to the information we request above.

I'd like my class to use specific resources on Discovery. Can you create a reservation on Discovery for my class?

In many cases, we will create a reservation on Discovery for your class for specific hardware resources. For example, if you course assignments require the use of GPUs or nodes with a large amount of memory, we can create a reservation on specific nodes that only your class can access for the duration of the course. However, Discovery is a shared resource, so we ask that you test out your assignments on Discovery so that you are requesting an appropriate amount of resources for your class. If a class needs an increase in resources due to higher than expected use, we can always increase your reservation. But, if a reservation is not being used to capacity, we will ask you to review the need for the requested resources, and adjust the reservation accordingly. We understand that sometimes it is difficult to determine exactly what your resource needs will be, but we do need to keep a reservation to a reasonable limit so that we keep the shared resources available to all users.

How long do my students have access to Discovery?

Students will have access to Discovery for the full duration of the class. If they want to continue to have access to Discovery after that time period, they'll need to request an individual account.

How do I get an account on Discovery?

If you are a professor or instructor at Northeastern, you can request an account on Discovery. See *Sponsor Approval Process* for more information.

How do my students get help with Discovery?

You and/or your students can either submit a Get Assistance with Research Computing ticket or email rchelp@northeastern.edu.

CHAPTER

TWENTYNINE

CPS CLASS INSTRUCTIONS

Caution: Note - The following instructions will only work for CPS classes.

These instructions describe the process of opening a CPS JupyterLab environment on the Open OnDemand (OOD) Discovery web portal and accessing class work directories.

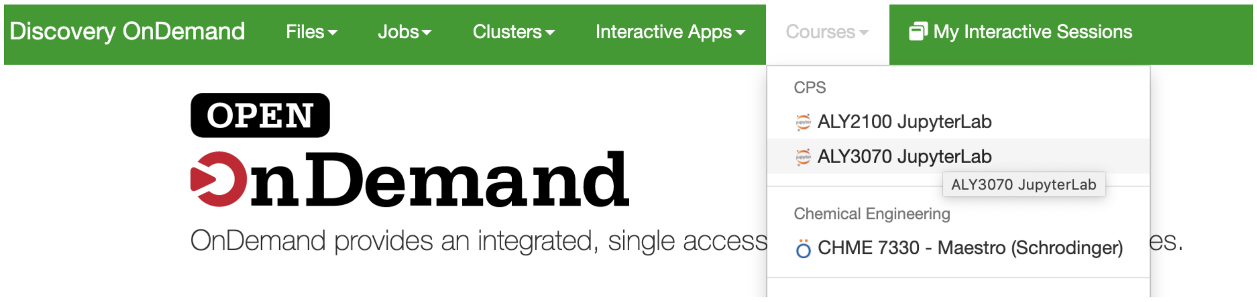
Note: Due to problems with launching OOD on **Safari**, we recommend using **Google Chrome**, **Mozilla Firefox** or **Microsoft Edge** browsers instead for best experience.

29.1 Open the CPS JupyterLab environment

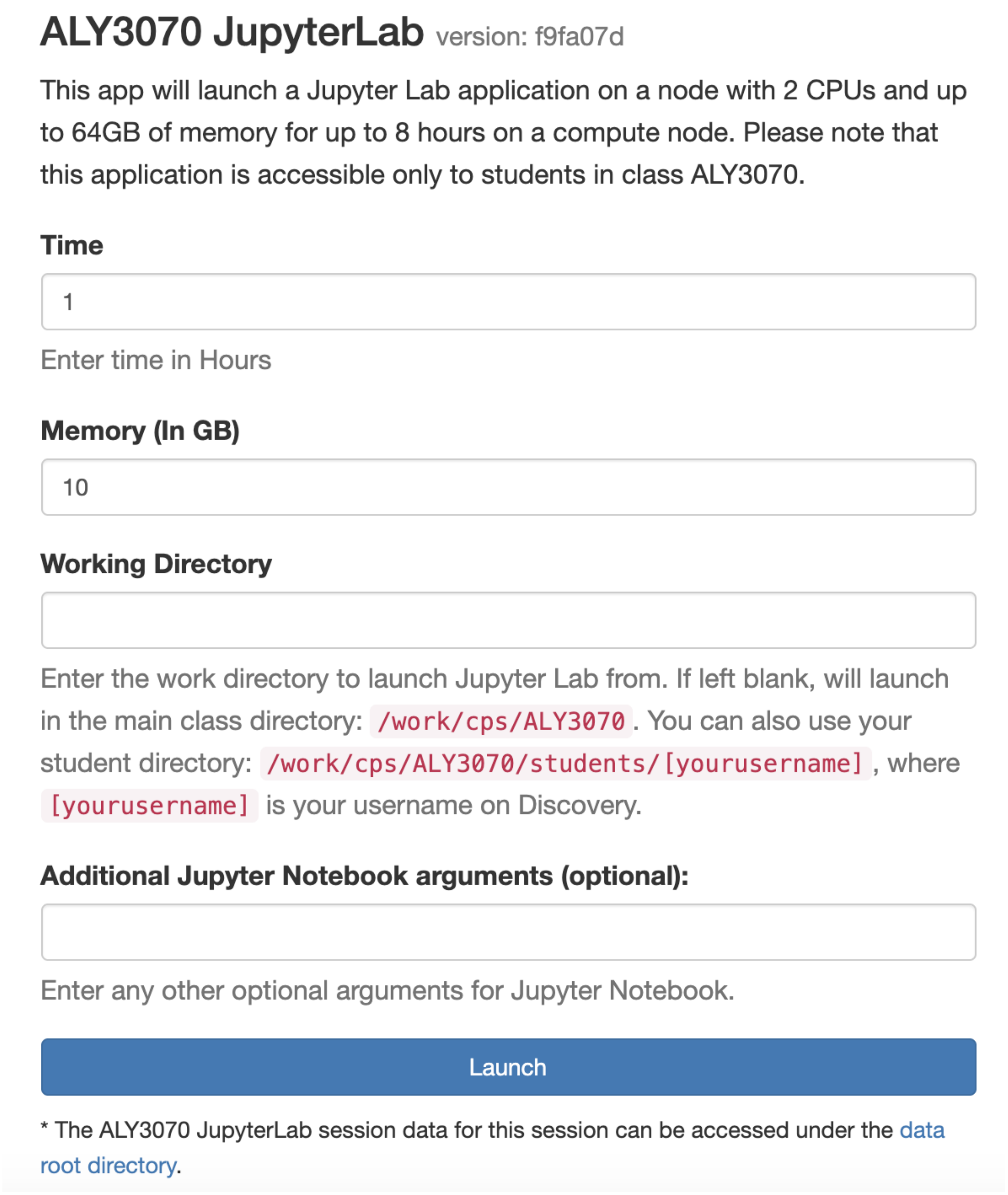
Important: The class instructor needs to fill in the: Discovery Classroom Use Request You will only be able to find your class resources if a request was already made.

In a web browser, go to http://ood.discovery.neu.edu. Login with your NU credentials.

Under the **Courses** menu, select your Class Name (For example: **ALY3070 JupyterLab**):



Select the default options and click **Launch**. Wait until the session is successfully created and ready to be launched (turns green).



ALY3070 JupyterLab version: f9fa07d

This app will launch a Jupyter Lab application on a node with 2 CPUs and up to 64GB of memory for up to 8 hours on a compute node. Please note that this application is accessible only to students in class ALY3070.

Time

1

Enter time in Hours

Memory (In GB)

10

Working Directory

Enter the work directory to launch Jupyter Lab from. If left blank, will launch in the main class directory: `/work/cps/ALY3070`. You can also use your student directory: `/work/cps/ALY3070/students/[yourusername]`, where `[yourusername]` is your username on Discovery.

Additional Jupyter Notebook arguments (optional):

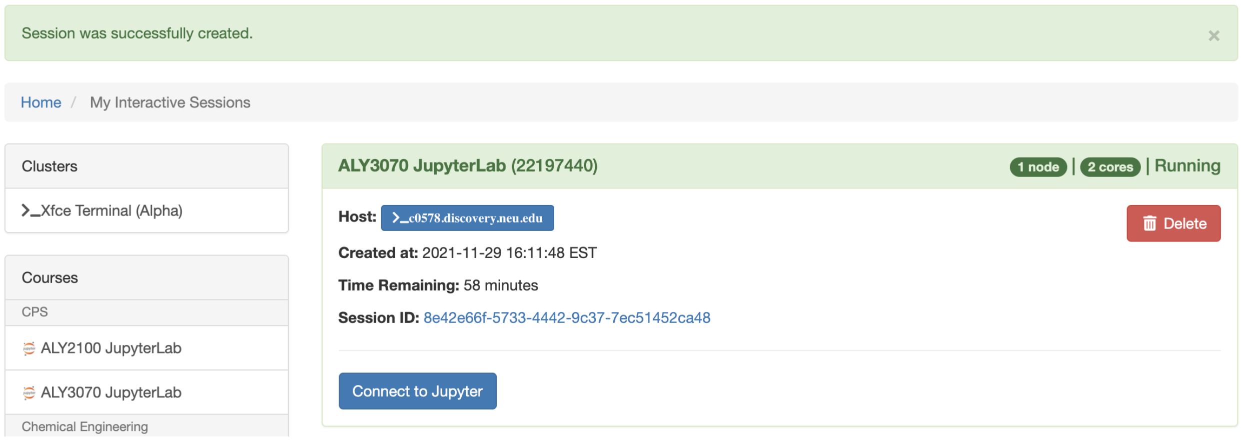
Enter any other optional arguments for Jupyter Notebook.

* The ALY3070 JupyterLab session data for this session can be accessed under the data root directory.

For more control of the session, modify **Time** for the session time (in hours), **Memory** to get more memory in GB, and the **Working Directory** where JupyterLab will launch.

Note: If **Working Directory** is left blank, the session will launch in the main class folder (in this example `/work/cps/ALY3070`). Alternatively, start the session directly from your personal working directory by entering: `/work/cps/ALY3070/students/[username]`, where `[username]` is your username on Discovery. The instructions below assume the field is left blank.

Click **Connect to Jupyter** to open JupyterLab:



Session was successfully created.

Home / My Interactive Sessions

Clusters

>_Xfce Terminal (Alpha)

Courses

CPS

ALY2100 JupyterLab

ALY3070 JupyterLab

Chemical Engineering

ALY3070 JupyterLab (22197440) 1 node | 2 cores | Running

Host: >_c0578.discovery.neu.edu

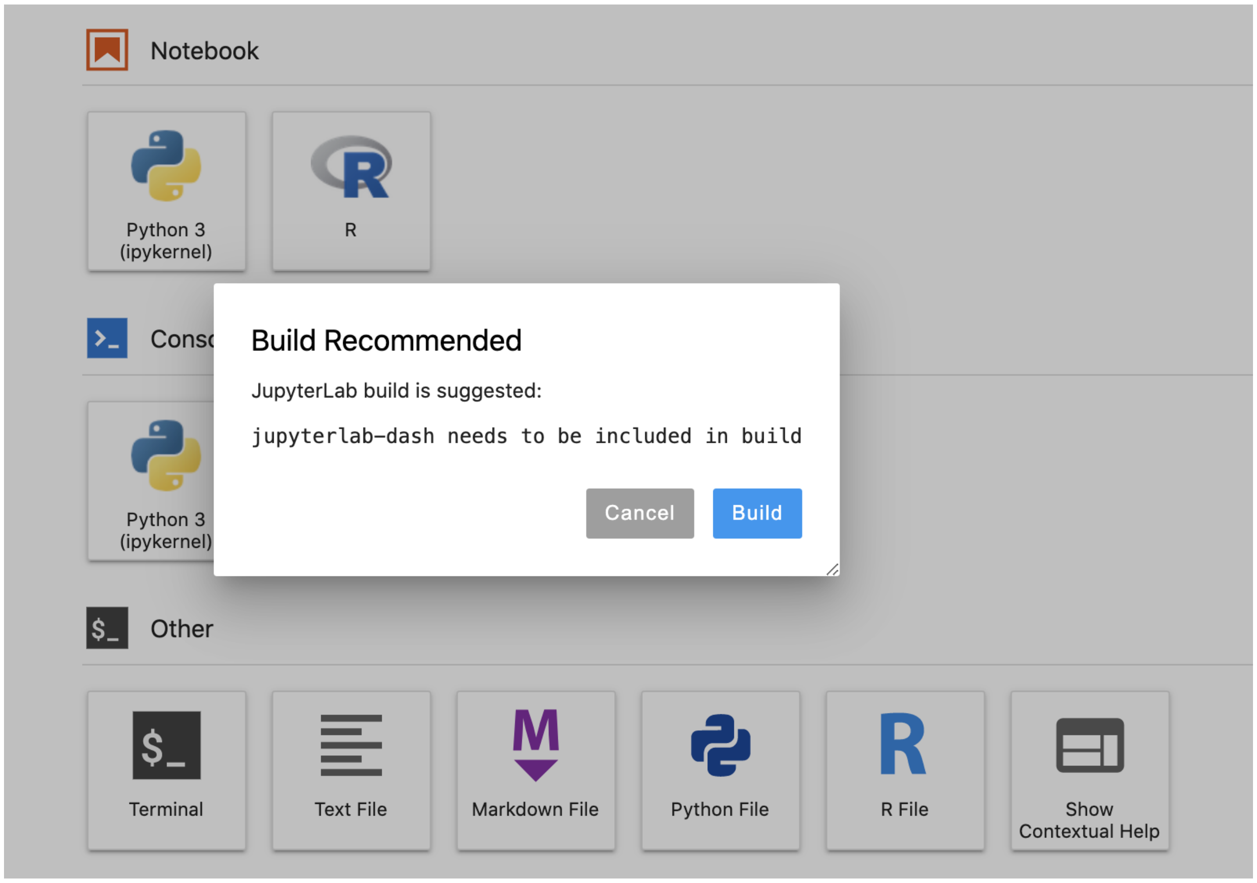
Created at: 2021-11-29 16:11:48 EST

Time Remaining: 58 minutes

Session ID: 8e42e66f-5733-4442-9c37-7ec51452ca48

This will open a JupyterLab interface in another tab.

Select **Cancel** when prompted with the **Build Recommended** option:



Build Recommended

JupyterLab build is suggested:

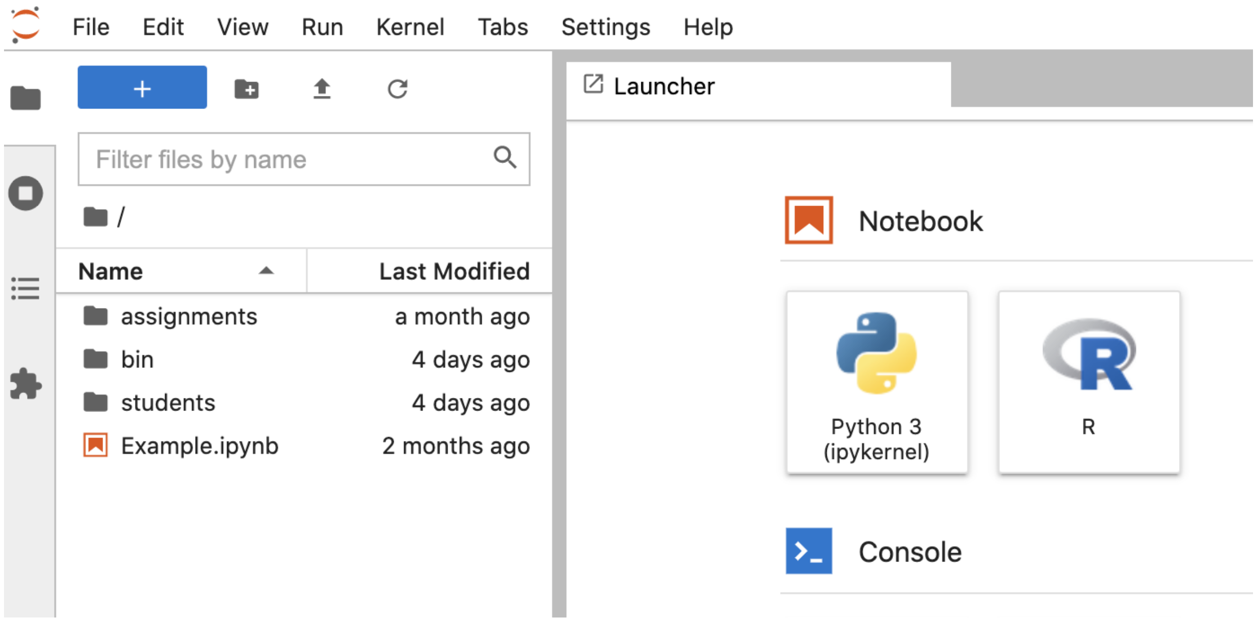
```
jupyterlab-dash needs to be included in build
```

The package jupyterlab-dash does not require a build, and will not work when build is enabled.

29.2 Access CPS class directories

After you are connected to a CPS JupyterLab session on OOD, you can access any shared class directories and your private class directory.

You can navigate between the class folders using the left menu. Your instructor may share files in this directory:

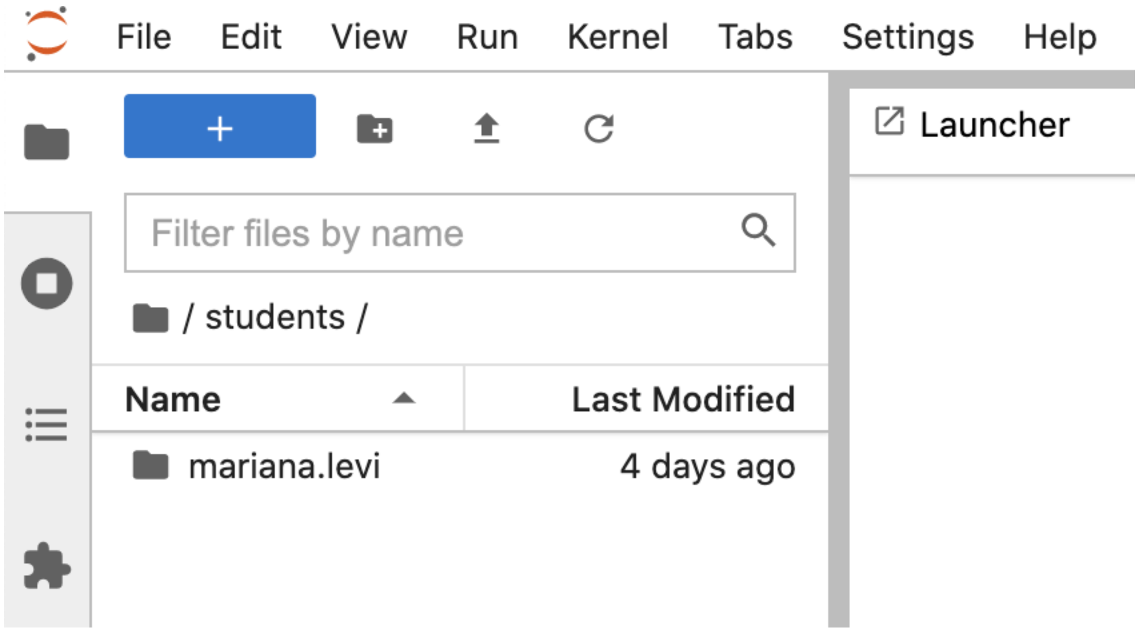


/

Name	Last Modified
assignments	a month ago
bin	4 days ago
students	4 days ago
Example.ipynb	2 months ago

For instance, file **Example.ipynb** can be viewed using Python Jupyter Notebook (but not edited or removed).

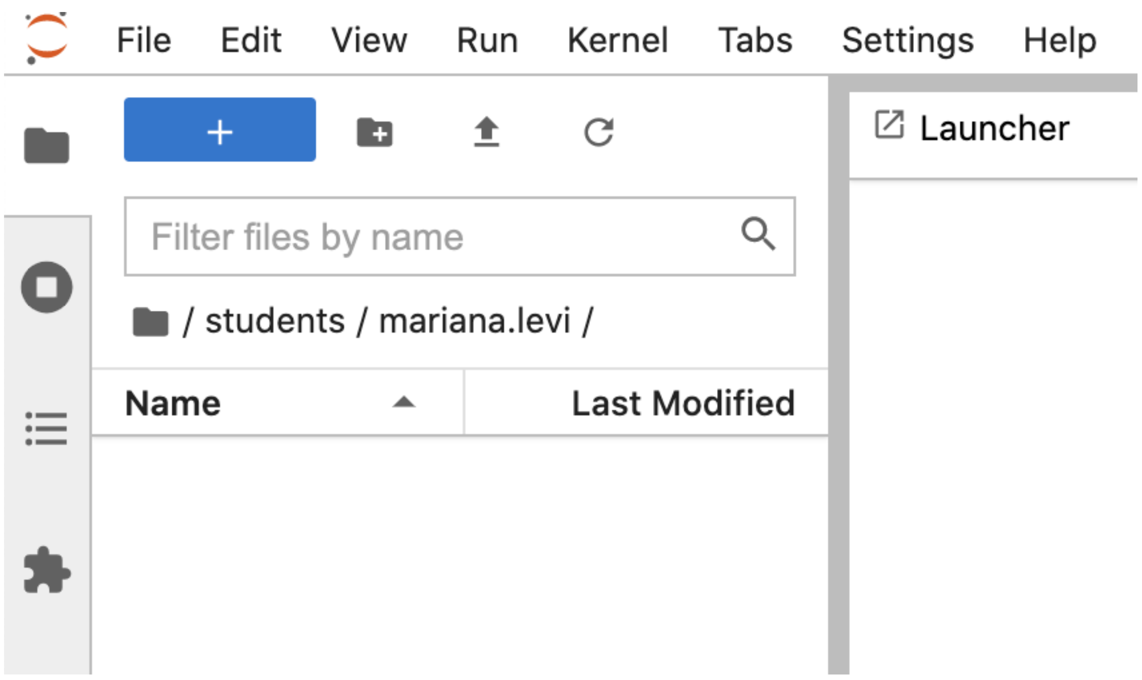
Navigate to the **students** directory, where you will see another directory under your username:



/ students /

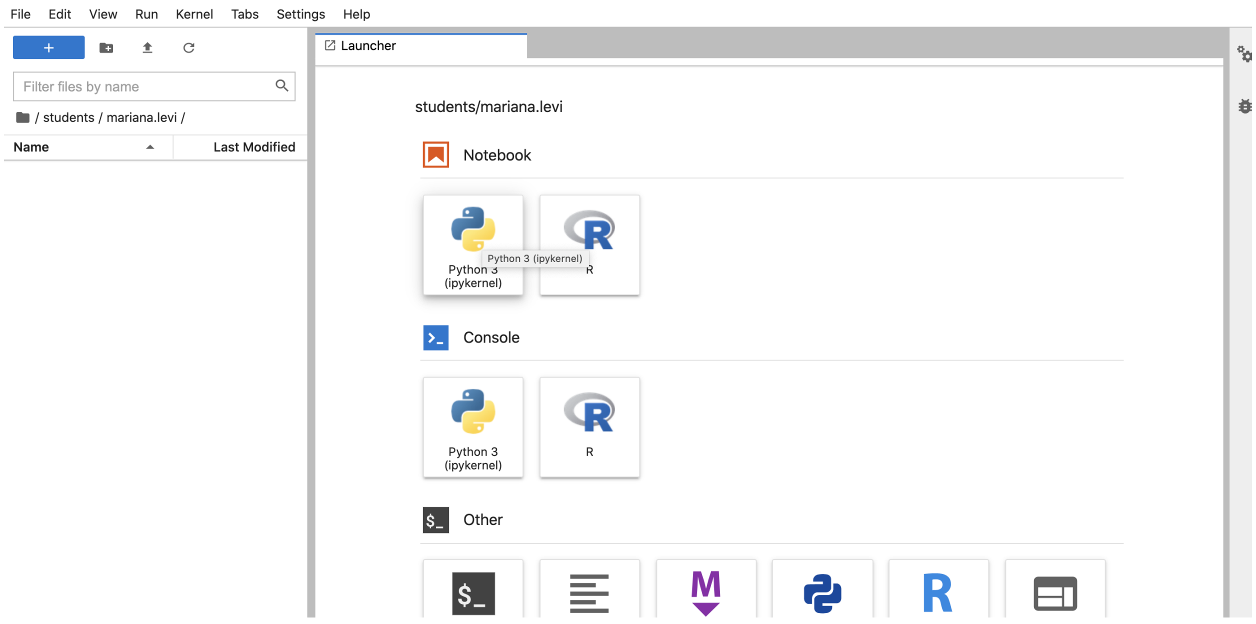
Name	Last Modified
mariana.levi	4 days ago

Enter your personal class directory (here, username `mariana.levi` is shown):

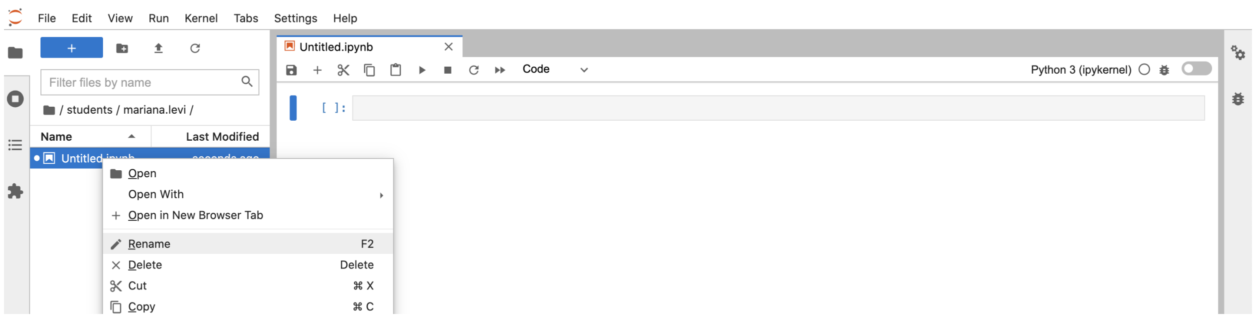


Now you can create and edit Jupyter Notebook files.

Open a new Python Notebook session from the Launcher menu by clicking the **Python 3 (ipykernel)**:



A new file will be created inside your directory called **Untitled.ipynb**. You can rename it by right-clicking on it and using the Rename option:



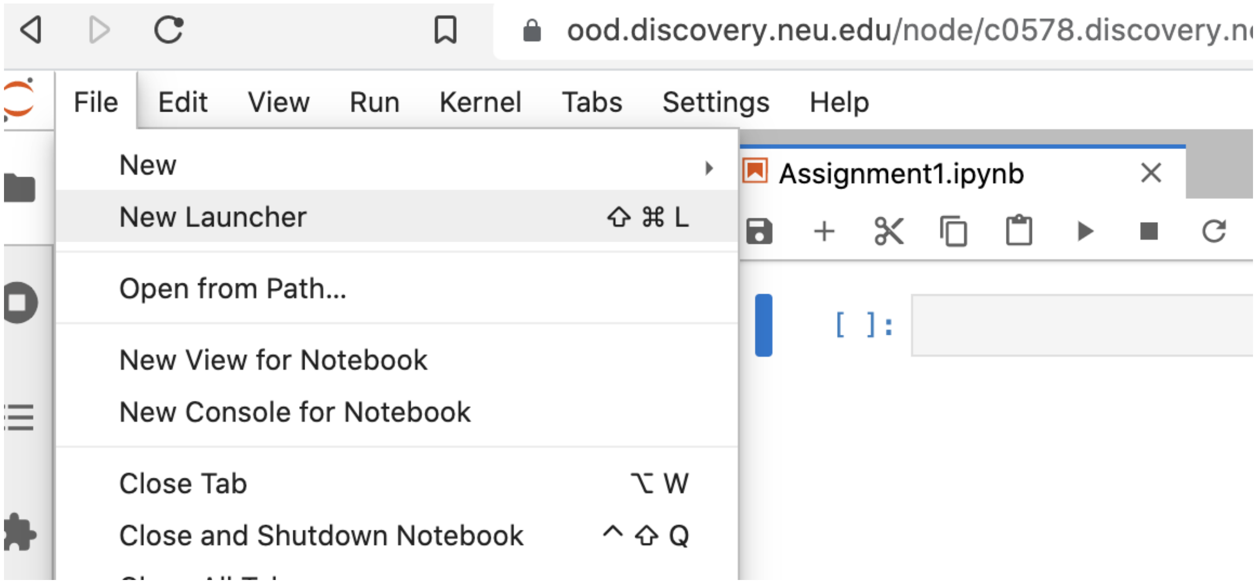
This Python notebook has ready-to-use Python packages needed for your class.

Note: Permission Denied errors: Do not attempt to create, edit or write files that are outside of your personal student directory. Most "Permission Denied" errors are due to directories or files having read-only access permissions.

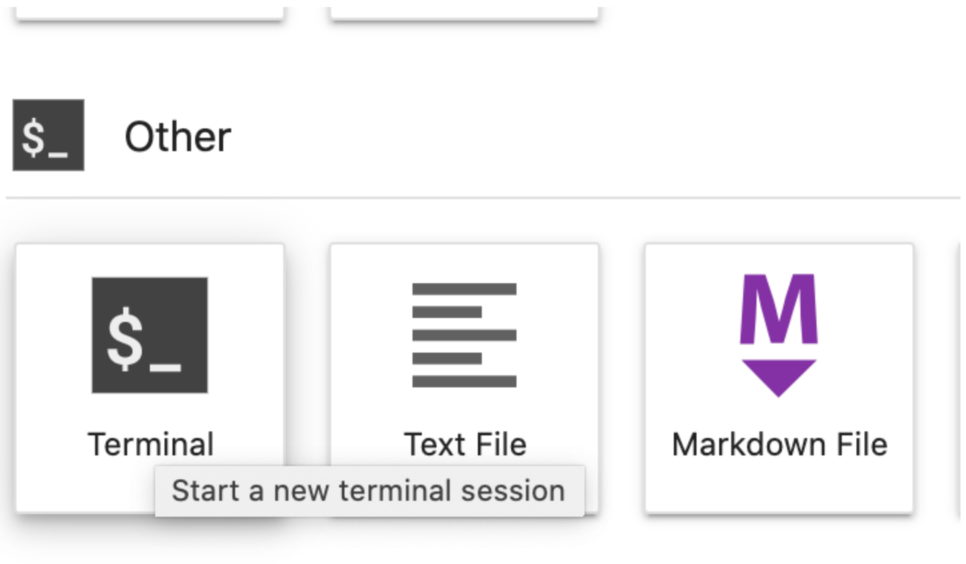
29.3 Submit CPS class assignments

Important: Due to the write-only access permissions on the **assignments** directory, it is required to use the command line interface (Linux Terminal) to submit assignments. **Using other methods, such as the JupyterLab interface or OOD File Explorer, currently does not work**.

To submit your assignment (for example, named: **Assignment1.ipynb**) to the **assignments** directory, open the Juypter-Lab New Launcher by clicking the **File** top menu option, and then selecting **New Launcher**:



Click on the **Terminal** option under **Other** to open a Linux terminal:



Navigate to your personal directory by typing the following command (change the class name from ALY3070 to your class name accordingly):

```
cd /work/cps/ALY3070/students/$USER
```

Where $USER is a saved shell variable for your username. You can optionally also replace it with your username.

Check that your assignment file is visible in the command line by typing `ls`. Then, Copy the assignment file to the **assignments** directory with this command (replace **Assignment1.ipynb** with your file name):

```
cp Assignment1.ipynb ../../assignments
```

To remove an existing assignment, type (replace **Assignment1.ipynb** with your file name):

```
rm ../../assignments/Assignment1.ipynb
```

Close the Terminal tab when done.

CHAPTER THIRTY

DOCUMENTATION PRIMER

Important: Remember that this style guide is a reference for team members working on documentation. It is essential to keep it up-to-date and easily accessible for all contributors.

This style guide outlines the best practices and conventions for creating clear, consistent, high-quality documentation for our project. By following these guidelines, we can ensure a cohesive writing style and tone across all documentation.

The purpose of the guide is to provide guidelines for writing clear, concise, and consistent documentation. It covers language, tone, formatting, and other aspects of documentation.

See also:

Our RTD is sphinx-based using a superset of Markdown called MyST.

The following are RTD-worthy bookmarks:

- MyST Syntax
- MyST Spec
- MyST-Parser Documentation
- MyST-Typography
- Markdown cheatsheet
- MyST How-To Guide
- MyST Example Pages

Important: As specified below, using colons (i.e., `:::`) in place of the typical hyphens when opening and closing blocks.

30.1 Writing Style

Dates and Time

Use EST and specify this when date or time are used (e.g., `9 September at 9-5 pm EST`).

The writing style in documentation is critical for ensuring consistency and understanding. Here are some best practices:

Clarity: Be as clear and straightforward as possible. Avoid ambiguity, along with complex sentences and unusual words. **KEEP IT SIMPLE.**

Do Not	Do
The initiation of the replication process…	To start the replication process…

Conciseness: Keep sentences and paragraphs short. This makes text easier to read and understand.

Do Not	Do
Once the process has been completed successfully, the next step is…	After completing the process, next…

Active Voice: Use the active voice whenever possible as it is more straightforward and engaging.

Do Not	Do
The command can be executed by the user.	The user can execute the command.

Address the Reader: Use the second person ("you") to address the reader directly.

Do Not	Do
One can see the results by…	You can see the results by…

- If you need to explain complex concepts, break them into smaller, more manageable sections to help your reader understand your message.

30.2 Formatting

MyST allows for markdown to be used for our Sphinx project (i.e., RTD).

Bookmark and Review-worth Reference

Check out this Markdown cheatsheet.

Commonly used elements follow the MyST-Typography.

Directives allow us to insert and format various elements (e.g., images, callouts, etc.). We use `colon_fence` (i.e., Colon Fence Directives) to replace the typical hyphens.

Important: We use the `colon_fence` extension rendered by MyST parses three colons (i.e., :) to open and close blocks.

Optional args specified for a directive are listed in MyST Spec.

Note: MyST-Parser Documentation includes several constructs we can use to enhance our pages. **Use elements with purpose, whether to call attention, reference related content, or warn.

Nesting blocks, along with the ability set parameters inline.

Note: There is no need to indent source text, as nested directives are explicitly clear by the number of `:` characters are used to open and close.

Attention: **Do not add line-breaks unless a line break in the rendered view is to need a line break.** Each paragraph remains on the same line, with line-breaks using to clearly show transitions between paragraphs, sections, and directives.

Tip: Set *Soft-Wrap* to be set to 100 characters by default when viewing `*.md`

Note: Add line-breaks before and after the opening and closing of directives (i.e., outside the struct). Only add a single line-break after opening, which is the case for most directives; for those that do not require any inputs but titles (e.g., `:::{directive} Title`), omit line-breaks after the opening.

Important: These nested admonitions were done, mostly, to demonstrate how it can be done. There are exceptions, but this is quite nested: be sure to consider if the flow of information is done most effectively when nests deeper, or even as deep as this example (i.e., five colons `:::::` deep).

Consistent formatting makes your documentation look professional and easy to read. Use consistent formatting for headings, lists, tables, and other elements, and adequately nest subheadings and lists. Here are some guidelines:

- **Headings**: Use headings and subheadings to structure your content. In MyST, use # for level 1 headings, ## for level 2, and so on.
- **Lists:** Use bulleted lists for unordered items and numbered lists for ordered items. In MyST, use bullets `-` for unordered lists and numbers `1.` for ordered lists.
- **Bold and Italics**: Use bold to highlight important concepts and italics to indicate new terms. In MyST, use bold text** for bold and italic text* for italics.
- **Code Blocks**: Use code blocks to present code snippets. In MyST, use triple colons `:::` to create code blocks and specify the language for syntax highlighting.
- **Admonitions**: Use MyST syntax to create admonitions highlighting notes, tips, warnings, and other important information. For example, use `:::{note}` or `:::{warning}` to create admonitions.

See also:

List admonitions types (i.e., callouts) available.

30.2.1 Admonitions

Admonitions highlight a particular block of text that exists slightly apart from the narrative of your page, like so:

This is the title

This is the message.

There are several types of admonitions:

Attention: Use to put emphasis on a specific point.

Caution: Use when extra care is warranted for a specific task.

Danger: Use when there is a potential problem that can occur.

Error: Use to highlight a bug.

Hint: Use to make a suggestion.

Important: Use to put emphasis on a sub-topic or sub-point.

Note: Use to add a side-note or footnote.

See also:

Use to reference another section, page, or reference that extends, provides details on subtopic or step, or serves as supplemental.

Tip: Use to provide pro-tips or tips & tricks.

Warning: Use to warn user of expected behaviors or outputs.

30.2.2 Tabs

MyST Tabs are a great way to present material (see {ref} GPUs for Deep Learning). It is essential for these to be used properly: situations where the same task or topic spans multiple setups or interfaces. For instance, installing deep learning to run on the GPUs is the task in the aforementioned example, where the tab chose is either PyTorch or Tensorflow.

30.3 Code Example

Code examples can help readers understand how to use a feature or solve a problem. Here are some guidelines to follow:

- **Clarity**: Ensure that your code is easy to understand. Include comments to explain complex or essential parts of the code.
- **Syntax** Highlighting: Use syntax highlighting to make your code easier to read. In MyST, you can specify the language for syntax highlighting in code blocks (see MyST Source code and APIs).
- **Error Handling**: If relevant, show how to handle errors and exceptions.
- **Consistency**: Ensure your code examples follow the same coding style, including indentation and naming conventions.
- **GitHub**: Reference the repo if snippets from another RC GitHub project is referred.

Note: Commented code, along with well-written self-documenting code, is a good way to avoid being too verbose. In other words, consider the purpose of the steps being presented, and if the purpose is using OpenMPI, as an example, there is no need to have a single code block for every step (e.g., put the loading of modules, creating of conda, changing directories, etc., in the same block). If you believe it is useful to add comments or provide a `{seealso}` to point to a page on Conda or modules, go ahead, but let's not repeat information and saturate our readers: **BE COMPLETE, YET CONCISE, WITH CODE.**

Tip: Long snippets of code tend to be easier to read when presented on several lines, with new-lines added strategically. For instance, if command-line, use \ to separate key-value when there are many arguments passed.

30.4 Language and Terminology

Clear and consistent language and terminology are crucial for effective communication. Here are some guidelines:

- **Consistency**: Use terminology consistently. Don't use different terms for the same concept.
- **Acronyms**: Always define acronyms the first time you use them.
- **Jargon**: Avoid using jargon unless necessary. If you have to use it, define it first.
- **Language**: Use either American or British English consistently throughout your documentation.

30.5 Documentation Structure

- Organize documentation into folders that represent sections, with each file as an item of its section.
- Use a clear hierarchy for headings and subheadings.
  - A single level-one heading (i.e., `# HEADING`) at the top of the page: `HEADING` will appear in the table of contents.
  - Add file path to index.md as follows:

~~~
```{toctree}
:hidden:
:caption: section

directory/filename
```
~~~

Where *section* is the top-level index listed in the table of contents and *filename* is the markdown file. Note that `.md` can be omitted. Furthermore, when naming a file or directory, use all lowercase with no spaces. For example, *Home Directory Storage Quota* is simply named `homequota.md`.

- Ensure each page has a clear purpose and covers a single topic.

30.6 Links and References

Links and references are crucial for helping users navigate your content and providing additional context. Here's how you can effectively use them in your documentation:

- Use descriptive link text instead of generic phrases like "click here" or "learn more."
- Regularly check for broken links.
- Reference external sources when necessary and provide proper attribution.
- Use cross-references to link to other sections within the documentation. *Text can also be modified inline.* (i.e., `[Text can also be modified inline.](linksreferences)`).
- Descriptive Links: Avoid using phrases like "click here" for hyperlink text. Instead, use descriptive text that informs the reader about the content they'll find when they follow the link.

Attention: Section labels should be all lowercase, with `-` placed between the words (e.g., `(section-one)`. ::

Do Not	Do
Click here for more information about the installation process.	Read our comprehensive installation guide.

- Attribution: If you're citing external sources or using someone else's work (e.g., images, code snippets), attribute the source correctly.

30.7 Images and Diagrams

Visual content like images and diagrams can significantly enhance your documentation. Here are some guidelines:

- **High-Quality Images**: Always use high-resolution images that are clear and easy to see. If you're using screenshots, ensure they accurately represent the current version of your product or interface.
- **Descriptive Alt Text**: Always provide alt text for your images. The alt text describes the image for people who can't see it, whether due to visual impairments, technical issues, or because they're using a screen reader.
- **Simple Diagrams**: Keep your diagrams simple. Avoid including unnecessary details that could confuse your readers.
- **Consistent Style**: If you use multiple diagrams, ensure they have a consistent style and use the same symbols and conventions.

30.8 Markdown and MyST Syntax

Markdown and MyST provide a simple way to format your content. Here are some tips for using these syntax effectively:

- **Basic Markdown**: Familiarize yourself with basic Markdown syntax, including headers, lists, links, images, and code blocks.
- **MyST Features**: MyST extends the standard Markdown syntax with additional features such as cross-references, footnotes, directives, warnings, and other particular elements. Make sure you understand how to use these features correctly.
- **Code Blocks**: Use triple backticks to create code blocks and specify the language for syntax highlighting.
- **Admonitions**: Use MyST's admonition syntax to highlight important information. For example, `:::{note}` creates a note, and `:::{warning}` creates a warning.

Note: There are many admonitions to choose from. Use your best judgement, look at other examples, and be sure to use the appropriate tone.

- **Consistency**: Be consistent with your use of MyST syntax. For example, always use the same header level for section titles, the same style for lists, etc.
- Follow the MyST syntax for advanced features such as directives and roles.

30.9 Review Process

Finally, ensure your documentation undergoes a thorough review process before it's published. We do this through PRs on GitHub.

- **Peer Review**: Encourage team members to review each other's work. This can help catch errors, ambiguities, and inconsistencies the original writer may have missed.
- **Proofreading**: Check for spelling, grammar, and punctuation errors. Tools like Grammarly or language-tool can help with this.

Tip: When adding a substantial amount of content, whether a new page or several sections, consider using Notion to draft. Then, update the RTD pages (i.e., this project) for a technical review. This way, development can be more interactive, time is not wasted adjusting formatting and other syntax, and the history is much simpler (and more promising) than that of our own managed via Git.

- **Technical Accuracy**: Ensure all code snippets, commands, and technical information are accurate and up-to-date.
- **Feedback**: Encourage users to provide feedback on your documentation. This can help you identify areas that need improvement.
- **Continuous Updates**: Keep your documentation up-to-date. As your product evolves, so should your documentation.

Attention: All members of the RC are welcome to contribute on the documentation. In fact, it is encouraged and much appreciated. Do not hesitate to ask for tasks or make suggestions to improve our *User Facing Documentation*.

Remember, good documentation is an ongoing effort. Always strive to improve and adapt to your users' needs.