
High Performance Computing @ Sharanga

Frequently Asked Questions

M. Nischay, V.K. Malhotra and N. Anil



BITS Pilani
Hyderabad Campus

Contents

1	Login and Accounts	7
1.1	How to generate an SSH key pair?	7
1.1.1	On Windows: Using PuTTY	7
1.1.2	On Windows: Using Windows 10 OpenSSH Client	8
1.1.3	On macOS	9
1.1.4	On Linux	10
1.2	How to login to the HPC facility?	10
1.2.1	On Windows: Using PuTTY	10
1.2.2	On Windows: Using Windows 10 OpenSSH Client	11
1.2.3	On macOS	11
1.2.4	On Linux	12
1.3	How do I add another SSH key to my account for remote access?	12
1.4	My ssh connection disconnects with “Write failed: Broken pipe”.	12
1.4.1	On Windows: Using PuTTY	12
1.4.2	On Windows: Using Windows 10 OpenSSH Client	13
1.4.3	Linux and macOS	13
2	Storage Space	15
2.1	How much storage space do I have?	15
2.2	How to find the current status of my storage quota?	15
2.3	I am getting the message “Disk Quota Exceeded”. What should I do?	15
2.4	How to find the files and directories that are occupying significant space in home directory?	15
2.5	Is there any backup available for my data?	16
3	Files and Directories	17
3.1	How to find the last modification date of a file?	17
3.2	How to find the deletion date of a file stored in SCRATCH ?	17
3.2.1	Using date	17
3.2.2	Using usertools	18
3.3	How to copy files or directories from my computer to the HPC cluster?	18

3.4	How to copy files or directories from HPC cluster to my computer?	19
3.5	How to access my \$HOME directory?	19
3.6	How to access my \$SCRATCH space?	19
3.7	SCP file transfer is very slow. Is it possible to speed it up?	19
4	Job Scheduling using Slurm	21
4.1	What is Slurm?	21
4.2	What are the frequently used commands for Slurm?	21
4.3	How to schedule a distributed memory CPU parallel job on Slurm?	21
4.3.1	Example of a job script <code>job.sh</code> using Modulefiles	22
4.3.2	Example of a job script <code>job.sh</code> using Spack	24
4.4	How to schedule a shared memory CPU parallel job on Slurm?	26
4.4.1	Example of a job script <code>job.sh</code> using Modulefiles	26
4.4.2	Example of a job script <code>job.sh</code> using Spack	29
4.5	How to schedule a GPU parallel job on Slurm?	31
4.5.1	Example of a job script using Modulefiles	31
4.5.2	Example of a job script using Spack	34
4.6	How to view the status of partitions?	36
4.7	How to view my submitted jobs?	36
4.8	How to cancel my job?	36
4.9	How to control my job?	37
4.10	Can I run my code for a few minutes on the login node?	37
4.11	How can I monitor the output of my jobs that are running?	37
4.12	Can I run interactive jobs?	38
4.13	I have job scripts written for another job scheduler. Can I use them for Slurm?	38
4.14	When is my job going to start to run?	38
4.15	Why is my job not running?	38
5	Package Management with Modulefiles	41
5.1	What are modulefiles?	41
5.2	How to view the list of available packages in the facility?	41
5.3	How to load and unload a package(s)?	42
5.4	How to view the list of loaded packages?	42
5.5	How do I check the version of the package installed?	42
5.6	After submitting a job, I am getting messages like “module: command not found”. Why am I getting this message?	42
6	Package Management with Spack	43
6.1	What is Spack?	43
6.2	What is the difference between Modulefiles and Spack?	43
6.3	How to view the list of available packages installed in the facility?	43

6.4	How to load and unload a package(s)?	44
6.5	How to view the list of loaded packages?	44
6.6	How do I check the version of the package installed?	44
6.7	After submitting a job, I am getting messages like “spack: command not found”. Why am I getting this message?	44
6.8	Between Modulefiles and Spack, which is more user friendly?	44
7	Miscellaneous	45
7.1	How do I acknowledge the high performance computing facility in my presentations or publications?	45
7.2	Is HyperThreading enabled on the compute nodes?	45
7.3	Can I launch HTML files on the cluster?	45
7.4	Can I launch GUI applications such as gnuplot?	45
7.5	While executing a command, I am getting the error “undefined symbol: EVP_KDF_ctrl, version OPENSSL_1_1_1b”.	45
7.6	After launching nano or vi(m), the application throws a segmentation fault.	46

Chapter 1

Login and Accounts

1.1 How to generate an SSH key pair?

An SSH key pair consists of two keys, namely, the public key and the private key. The public key is stored in the SSH directory of the HPC facility, while the private key remains with the user. Users are requested to safeguard the private key carefully. Note that the file name of the public key is same as the private key except that the public key has a file extension `.pub`. For example, if the file name of the SSH private key is `id_rsa`, then the file name of the public key would be `id_rsa.pub`.

1.1.1 On Windows: Using PuTTY

Users are recommended to use PuTTY to access SSH in Windows. The installer for PuTTY can be downloaded from [this](#) website. Within this link, please choose the MSI 64-bit ('Windows Installer for putty') version of the package.

- Once installation is complete, go to **Start Menu -> All Programs -> PuTTYgen** and then launch the application. Figure 1.1 shows the screenshot of the launched application.
- For the type of key to generate, please select RSA and click **Generate**, and start moving the mouse within the Window. Putty uses the mouse movements to collect randomness. Note that on older machines, please select SSH-2 (RSA) if RSA is not available.
- Once the progress bar becomes full, the actual key generation takes place. This process may take few seconds to several minutes. When complete, the public key should appear in the Window. You can specify a passphrase for the key (Optional).
- Under Actions click on the **Save public key** and **Save private key** to save your public and private keys respectively.
- Please share the public key with the system administrator, who will copy it to the SSH directory of the HPC facility.

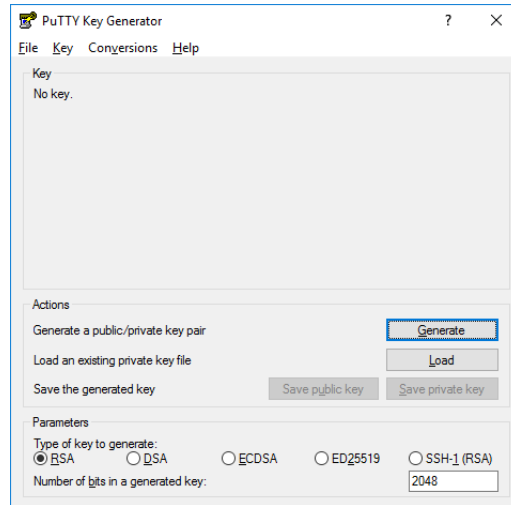


Figure 1.1: PuTTY Key Generator.

1.1.2 On Windows: Using Windows 10 OpenSSH Client

Microsoft, from **Windows 10** build **1803** and onwards have shipped OpenSSH Client on all Windows 10 PCs. Users can use the OpenSSH `ssh-keygen` command to generate their SSH keys. The following steps will help in generating the SSH keys.

- Press **Windows Key + R** to open the Windows Run Prompt.
- Type `cmd` and press **ENTER**.
- Type `ssh-keygen` in the console and press **ENTER**.
- In order to generate the SSH key pair and store them securely, Windows may prompt you to enter a directory where the key pair will be stored. You may press **Enter** to choose the default location provided.
- Next, you'll be prompted to enter your passphrase for the key (Optional).
- Once the process is completed, two files will be generated along with the **SHA256** fingerprint. The key's random art image will also be displayed on-screen. A sample screenshot is shown in Figure 1.2. Note that the generated image and fingerprint will differ from system to system.
- Please share the public key with the system administrator, who will copy it to the SSH directory of the HPC facility.


```

C:\Users\TestSubjector>ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\TestSubjector\.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in C:\Users\TestSubjector\.ssh/id_rsa.
Your public key has been saved in C:\Users\TestSubjector\.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:7Dfv/J1zovE26t6ILsRL55xQrFFlIn+012KhmdMma24 testsubjector@TestSubject-Gamma
The key's randomart image is:
+---[RSA 2048]-----+
|  .  .o |
|  o.o  |
|  oo .  |
|  ..=o= |
|  .S+* + |
|  .**.o  |
|  o+*o.. |
|  oE.+ =+oo|
|  ..oo+X=+++|
+-----[SHA256]-----+
C:\Users\TestSubjector>

```

Figure 1.2: Windows 10 Key Random Art.

1.1.3 On macOS

Users are recommended to use the **Terminal** application to generate ssh key pairs. It is located in the utilities which can be accessed using the Finder.

- Open Terminal and enter `ssh-keygen -t rsa`.
- This starts the key generation process. The utility may also prompt you to indicate where to store the key pair. Press ENTER to accept the default location.
- Next, you'll be prompted to enter your passphrase for the key (Optional).
- Once the process is completed, two files will be generated along with the SHA256 fingerprint. The key's random art image will be displayed on-screen. A sample screenshot generated on a mac system is shown in Figure 1.3. Note that the generated image and fingerprint will differ from system to system.
- Please share the public key with the system administrator, who will copy it to the SSH directory of the HPC facility.

```

Your identification has been saved in /Users/user/.ssh/id_rsa.
Your public key has been saved in /Users/user/.ssh/id_rsa.pub.
The key fingerprint is:
ae:89:72:0b:85:da:5a:f4:7c:1f:c2:43:fd:c6:44:38 user@mymac.local
The key's randomart image is:
+--[ RSA 2048 ]-----+
|      .      |
|      E .    |
|      . o    |
|      o . . S |
| + + o . +   |
| . + o = o + |
| o...o * o   |
| . oo.o .    |
+-----+

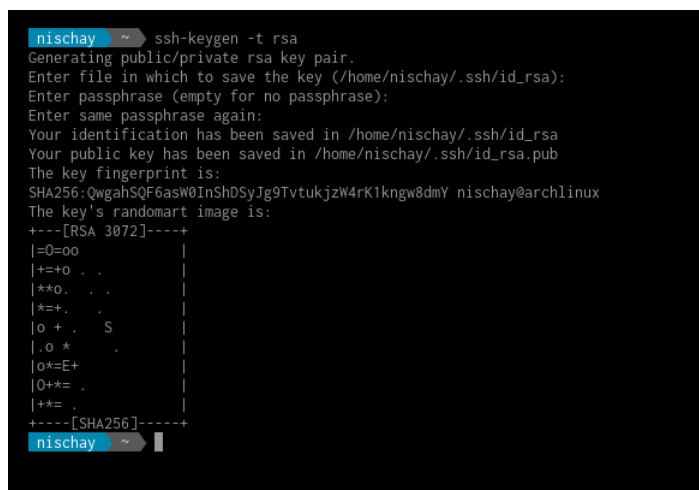
```

Figure 1.3: macOS Key Random Art.

1.1.4 On Linux

To generate `ssh` keys on a Linux based operating system,

- Open a Terminal or Console and enter `ssh-keygen -t rsa`.
- The utility will now prompt you to enter the location where to save the key pair. Press ENTER to accept the default location.
- Next, you'll be prompted to enter your passphrase for the key (Optional).
- Once the process is completed, two files will be generated along with the **SHA256** fingerprint. The key's random art image will also be displayed on-screen. A sample screenshot generated on a linux system is shown in Figure 1.4. Note that the generated image and fingerprint will differ from system to system.
- Please share the public key with the system administrator, who will copy it to the SSH directory of the HPC facility.



```
nischay ~ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/nischay/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/nischay/.ssh/id_rsa
Your public key has been saved in /home/nischay/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:QwgahSQF6asW0InShDSyJg9TvtukjzW4rK1kngw8dmY nischay@archlinux
The key's randomart image is:
+---[RSA 3072]-----+
|  =0=oo |
| +=+0 . . |
| **0. . . |
| *+=. . . |
| o + . S |
| . o * . . |
| o +=E+ |
| o +=. . |
| +=. . |
+---[SHA256]-----+
nischay ~
```

Figure 1.4: Linux Key Random Art.

1.2 How to login to the HPC facility?

Users can securely access the HPC facility using the `ssh` protocol.

1.2.1 On Windows: Using PuTTY

- To launch PuTTY, go to **Start Menu -> All Programs -> PuTTY**.
- When the program starts, a window titled **PuTTY configuration** should open.
- Under **Connection** please expand **SSH**.

- Select **Auth** and a new configuration pane should open on the left.
- Under **Authentication Parameters**, you will find a **Browser** button. Click on it and select the file path of the Private SSH Key you generated.
- Once the above process is completed, users can go back to **Session** Category. This window has a configuration pane on the right containing **Hostname (or IP address)** field.
- Enter `hpc.bits-hyderabad.ac.in` in the **Hostname** field and click **Open**.
- If this is the first time you are trying to login you might get a Security Alert. This is normal, and you should click **Yes**.
- After the security alert, you should get a terminal window asking for username. Type the **username** provided to you by the HPC team to access the facility. If you have entered a **passphrase** for your ssh key pair, please type the password and press ENTER.
- You should now be connected to the login node of the HPC facility.

Alternatively, users can click on the **Save** button in the Session category to save the SSH configuration. Users can then quickly connect to the login node by simply loading their saved configuration.

1.2.2 On Windows: Using Windows 10 OpenSSH Client

If you have already generated an SSH key pair using the OpenSSH Client, you can then directly access the login node by using the **Command Prompt**.

- Press **Windows Key + R** to open the Windows Run Prompt.
- Type `cmd` and press ENTER.
- Type `ssh <username>@hpc.bits-hyderabad.ac.in` in the console and press ENTER. Here, `<username>` is the **username** provided to you by the HPC team to access the facility. If you have entered a **passphrase** for your ssh key pair please type the password and press ENTER.
- You should now be connected to the login node of the HPC facility.

1.2.3 On macOS

Users are recommended to use the **Terminal** application to login using **ssh**. It is located in the utilities, which can be accessed using the Finder.

- Open Terminal.
- Type `ssh <username>@hpc.bits-hyderabad.ac.in` in the console and press ENTER. Here, `<username>` is the **username** provided to you by the HPC team to access the facility. If you have entered a **passphrase** for your ssh key pair please type the password and press ENTER.
- You should now be connected to the login node of the HPC facility.

1.2.4 On Linux

- Open a Terminal or Console.
- Type `ssh <username>@hpc.bits-hyderabad.ac.in` in the console and press ENTER. Here, `<username>` is the `username` provided to you by the HPC team to access the facility. If you have entered a `passphrase` for your ssh key pair please type the password and press ENTER.
- You should now be connected to the login node of the HPC facility.

1.3 How do I add another SSH key to my account for remote access?

Users can add multiple SSH keys into their `authorized_keys` file of the HPC facility.

- Open a terminal in the HPC facility.
- Using `nano` or `vi` open `~/.ssh/authorized_keys`.
- Append the contents of the new public key at the end of the file. Please ensure that the entire content of the public key fits in one single line with no additional spaces in between. Note that, each line in this file represents 1 ssh key pair.
- Save the file and exit. You should now be able to connect from the newly added machine.

Note: You are requested to copy the **public** key of the key pair and not the **private** key. Public key can be identified easily with the **.pub** file extension at the end of the filename. For example `id_rsa.pub` refers to the public key and `id_rsa` refers to the private key.

1.4 My ssh connection disconnects with “Write failed: Broken pipe”.

By default `ssh` enforces a maximum idle time after last input from user. This is a security precaution to protect the account from unauthorized access. A user can override this setting by configuring the `KeepAlive` directive, as explained below.

1.4.1 On Windows: Using PuTTY

- Launch PuTTY from Start Menu -> All Programs -> PuTTY.
- Select **Connection**.
- In the configuration pane on the left you should find an option called **Seconds between keepalive**.
- Enter in **seconds** the amount of time you would want as maximum idle time.

1.4.2 On Windows: Using Windows 10 OpenSSH Client

- Open a file editor, preferably Notepad.
- Type the following text without the \$ symbol. Example shown is for 300 seconds and users can change this value accordingly.

```
$ Host *  
    ServerAliveInterval 300
```

- Save this file as **config** under C:\Users\<<your username>\.ssh\config.

1.4.3 Linux and macOS

- Open a Terminal.
- Using a file editor of your choice, type the following lines without the \$ symbol. Example shown is for 300 seconds and users can change this value accordingly.

```
$ Host *  
    ServerAliveInterval 300
```

- Save this file as **config** under ~/.ssh/config.
- Make sure the file has **644** set as file system permission. If not, users can run `chmod 644 ~/.ssh/config` to set the file permissions.

Chapter 2

Storage Space

2.1 How much storage space do I have?

Users are provided with 40GiB of storage space in their `$HOME` directory and unlimited storage space (subject to the availability) in their `$SCRATCH` directory. Please note that files stored in `$SCRATCH` directory are deleted after 15 days from their last modification date.

2.2 How to find the current status of my storage quota?

Users are provided with 40GiB of storage quota for their home directories. Details on the quota can be obtained using the command `usertools`.

```
$ usertools quota info
```

```
Username: nischay
Quota Exceeded: No
Total Storage Alloted: 40 GiB
Total Storage Available: 37.86 GiB (94.65 %)
Total Storage Consumed: 2.14 GiB (5.35 %)
```

2.3 I am getting the message “Disk Quota Exceeded”. What should I do?

To ensure fair share of storage resources on the HPC facility, users have limited disk quota for their `$HOME` directory. Once the storage space exceeds the allotted quota, users might receive the above message. To resolve this issue, we request the users to either shift files from their `$HOME` directory to `$SCRATCH` or to their personal machine.

2.4 How to find the files and directories that are occupying significant space in home directory?

The `du` command lists the size of files/folders.

```
$ du
```

For example, if we want to view the top 20 files and directories occupying the most space in `$HOME` directory, then type the following command:

```
$ du -ah ~/ 2>/dev/null | sort -n -r | head -n 20
```

However, some softwares and libraries create hidden directories in the home directory. To see their data usage, you can run the following command:

```
$ du -sh ~/.??*
```

2.5 Is there any backup available for my data?

Currently, we maintain daily backups of the `$HOME` directory for a period of 30 days. This is subjected to the HPC policy and also the availability of storage space on the backup drives. In case a user has accidentally lost any of their data, they may contact the system administrator at hpc@hyderabad.bits-pilani.ac.in for a possible restoration.

Please note that `$SCRATCH` directory is **not** backed up by the system and therefore not restorable in case of any loss of data. Users are advised to maintain their **own** backups as the HPC facility is not responsible for the loss of users data.

Chapter 3

Files and Directories

3.1 How to find the last modification date of a file?

Users can use the `stat` command to get the detailed status of a file. For example, if a user has a file called `example.txt`, then the `stat` command can be invoked as

```
$ stat example.txt
```

The output will be:

```
File: example.txt
Size: 0 Blocks: 1 IO Block: 4194304 regular empty file
Device: f96638d6h/4184226006d Inode: 144115339825778942 Links: 1
Access: (0600/-rw-----) Uid: ( 1000/ nischay) Gid: ( 1000/ nischay)
Access: 2020-08-18 23:21:06.000000000 +0530
Modify: 2020-08-18 23:21:06.000000000 +0530
Change: 2020-08-18 23:21:06.000000000 +0530
```

3.2 How to find the deletion date of a file stored in SCRATCH?

3.2.1 Using date

The file deletion date of a file can be obtained using the command `date`. For example, if we want to know the deletion date of the file `example.txt`, please type the following command

```
$ date -d "2020-08-18 23:21:06.000000000 +0530 +15days"
```

Here, `2020-08-18 23:21:06.000000000 +0530` is the access time of the file, obtained using the `stat` command as shown earlier. Since the files stored in `$SCRATCH` are automatically deleted after 15 days from its last modification, we need to add `+15days` to the access time. The output of the above command will be:

3.2.2 Using usertools

Alternatively, users can use the `usertools` command to get the access time and also the time left before deletion with one single command. Please note that `usertools` is **not** a standard linux command but developed by us to obtain the status of files with ease.

```
$ usertools file access example.txt
```

```
Filename: example.txt
Access Time: 2020-08-18 23:21:06.100206
Time Left: 359 hours 59 minutes 52 seconds.
```

3.3 How to copy files or directories from my computer to the HPC cluster?

Users of Linux, Windows 10 (1803 and above) and macOS can copy files using `scp` utility. The following commands illustrate copying a file `example.txt` and a directory `test` from a user's personal machine to the HPC facility.

```
$ scp example.txt <user>@hpc.bits-hyderabad.ac.in:
```

Here `<user>` is the `username` of the account to access the facility. Please note the `:` (colon) after the domain name. By default, if no path is specified after the colon, the file is copied into the `$HOME` directory of the user. Similarly, to copy a directory, `-r` flag is used along with `scp`.

```
$ scp -r test <user>@hpc.bits-hyderabad.ac.in:
```

Alternatively, the above two commands can be merged into a single command.

```
$ scp -r example.txt test <user>@hpc.bits-hyderabad.ac.in:
```

Users of macOS and Linux can use the `rsync` utility, which offers faster resumable transfers.

```
$ rsync -avz --progress test <user>@hpc.bits-hyderabad.ac.in:
```

Here,

- `-a` represents archival mode of transfer, which preserves metadata of the file.
- `-v` represents verbose mode.
- `-z` represents compression mode, which compresses the data during the transfer and thus improves the transfer rate.
- `--progress` shows the progress of the transfer.

3.4 How to copy files or directories from HPC cluster to my computer?

Users can copy a file or directory from the HPC cluster to their machine by using the `scp` command. For example, if a user wishes to transfer a file `example.txt` and a directory `test` stored in `\home\<user>\result` directory, then the following commands can be used.

```
$ scp <user>@hpc.bits-hyderabad.ac.in:\home\<user>\result\example.txt .
```

```
$ scp -r <user>@hpc.bits-hyderabad.ac.in:\home\<user>\result\test .
```

Here `<user>` is the `username` of the account to access the facility. Please note the `.` (dot symbol) at the end of the command. The above commands copy `example.txt` and `test` into the current directory of the user's terminal on their local machine. For more information on `scp` and `rsync`, users can refer to the `manpages` for the utilities. The manual pages of these commands can be accessed using `man scp` and `man rsync`.

3.5 How to access my \$HOME directory?

Users can access their `$HOME` directory by typing `cd $HOME` in a terminal on Sharanga. Note that the home directory of a user is accessible only to that particular user and cannot be viewed or accessed by others in the cluster.

3.6 How to access my \$SCRATCH space?

Users can access their `$SCRATCH` space by typing `cd $SCRATCH` in a terminal on Sharanga. Note that the scratch space of a user is accessible only to that particular user and cannot be viewed or accessed by others in the cluster.

3.7 SCP file transfer is very slow. Is it possible to speed it up?

Users are advised to use `rsync` with `-avz` flags for compressible and resumable file transfers.

Chapter 4

Job Scheduling using Slurm

4.1 What is Slurm?

Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for clusters. It facilitates the execution of parallel jobs on the cluster in an efficient manner. For more information on Slurm, users are requested to visit [Slurm Workload Manager - Documentation](#).

4.2 What are the frequently used commands for Slurm?

Here is the list of frequently used commands. For more information, users are requested to refer [Slurm Workload Manager - Documentation](#).

- **salloc** - To allocate resources to a Slurm job with a possible set of constraints.
- **sbatch** - Submits a Slurm job script.
- **scancel** - Cancels a Slurm job.
- **scontrol** - To query information and manage jobs.
- **sinfo** - To retrieve information about partitions and nodes.
- **squeue** - To query the list of pending and running jobs.
- **srun** - To run a parallel job on the cluster managed by Slurm.

4.3 How to schedule a distributed memory CPU parallel job on Slurm?

Users can use the **sbatch** command provided by Slurm to submit a job script. Note that, for loading the required packages one can use either **Modulefiles** or **Spack**. In the following we have shown example job scripts using **Modulefiles** and **Spack**.

4.3.1 Example of a job script `job.sh` using Modulefiles

```
1  #!/bin/bash
2  #SBATCH -p compute
3  #SBATCH -N 1
4  #SBATCH -n 4
5  #SBATCH --mem 512M
6  #SBATCH -t 4-2:23 # time (D-HH:MM)
7  #SBATCH --job-name="hello_test"
8  #SBATCH -o slurm.%j.out
9  #SBATCH -e slurm.%j.err
10 #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
11 #SBATCH --mail-type=ALL
12 module load openmpi-3.1.6-gcc-9.3.0
13 module load parmetis-4.0.3-gcc-9.3.0
14 module load openblas-0.3.10-gcc-9.3.0
15 module load hdf5-1.10.6-gcc-9.3.0
16 module load petsc-3.13.1-gcc-9.3.0
17 srun ./execname
```

To submit the above job script use the following command.

```
$ sbatch job.sh
```

If you wish to test your job script and want to find when it is estimated to run, please run

```
$ sbatch --test-only job.sh
```

Note that this does not actually submit the job. A detailed explanation for each code snippet of the job script `job.sh` is given below.

```
1  #!/bin/bash
```

This is the standard convention to let the linux shell know what interpreter to run.

```
2  #SBATCH -p compute
3  #SBATCH -N 1
4  #SBATCH -n 4
```

Configuration variables for Slurm start with `SBATCH`.

- `-p` refers to the partition to be used by Slurm. Sharanga provides two partitions, namely, `compute` and `gpu`. For jobs to be executed exclusively on CPUs, we use the `compute` partition.
- `-N` represents the number of nodes to be used. In the present example, we are using 1 node.

- `-n` represents the number of tasks to be executed. For codes employing distributed parallelism, users are requested to specify the number of tasks as the number of compute cores required.

```

5  #SBATCH --mem 512M
6  #SBATCH -t 4-2:23 # time (D-HH:MM)
7  #SBATCH --job-name="hello_test"

```

- `--mem` represents the maximum amount of required memory. Here, we are requesting 512 Megabytes of memory. Note that Slurm prioritises lower memory jobs over higher memory jobs in the queue. This may result in delayed execution of higher memory jobs. Therefore, users are requested to give accurate and desirable memory limits.
- `-t` represents the maximum wall clock time the job requires. Here, we are requesting 4 days, 2 hours and 23 minutes. Slurm prioritises shorter time limit jobs over longer time limit jobs in the queue. This may result in delayed execution of longer time limit jobs. Therefore, users are requested to give accurate and desirable time limits. Note that setting values greater than 168 hours will result in the termination of the job by Slurm automatically.
- `--job-name` represents the name of the job.

```

8  #SBATCH -o slurm.%j.out
9  #SBATCH -e slurm.%j.err

```

- `-o` represents `stdout`.
- `-e` represents `stderr`.

We are instructing Slurm to redirect `stdout` and `stderr` of the executed application to disk. For example, if your `jobid` is 121, then `slurm.121.out` would contain the normal output of the application, while `slurm.121.err` would contain the error output of the application. These files will be stored in the directory, where the jobs were launched from.

```

10 #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
11 #SBATCH --mail-type=ALL

```

- `--mail-user` represents the email address to which job events are to be delivered.
- `--mail-type` represents the type of events to be logged. Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL (equivalent to BEGIN, END, FAIL, REQUEUE, and STAGE_OUT), STAGE_OUT (burst buffer stage out and teardown completed), TIME_LIMIT, TIME_LIMIT_90 (reached 90 percent of time limit), TIME_LIMIT_80 (reached 80 percent of time limit) and TIME_LIMIT_50 (reached 50 percent of time limit). Here, we have specified the event type to be `ALL`.

```

12 module load openmpi-3.1.6-gcc-9.3.0
13 module load parmetis-4.0.3-gcc-9.3.0
14 module load openblas-0.3.10-gcc-9.3.0
15 module load hdf5-1.10.6-gcc-9.3.0
16 module load petsc-3.13.1-gcc-9.3.0

```

We are using Modulefiles to set the environment needed to run our application. The example application depends on PETSc and also has transitive dependencies OpenMPI, ParMETIS, OpenBLAS and HDF5. We are using GCC 9.3.0 compiled libraries for the application.

```

17 srun ./execname

```

Finally, we are using `srun` to start the execution of the application. This is somewhat analogous to `mpirun`. Users are requested **not** to use `mpirun` and instead use `srun`. `srun` takes care of the allocation and efficient management of resources via Slurm automatically.

4.3.2 Example of a job script `job.sh` using Spack

```

1  #!/bin/bash
2  #SBATCH -p compute
3  #SBATCH -N 1
4  #SBATCH -n 4
5  #SBATCH --mem 512M
6  #SBATCH -t 4-2:23 # time (D-HH:MM)
7  #SBATCH --job-name="hello_test"
8  #SBATCH -o slurm.%j.out
9  #SBATCH -e slurm.%j.err
10 #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
11 #SBATCH --mail-type=ALL
12 spack load petsc@3.13.1%gcc@9.3.0
13 srun ./execname

```

To submit the above job script use the following command.

```
$ sbatch job.sh
```

If you wish to test your job script and want to find when it is estimated to run, please run

```
$ sbatch --test-only job.sh
```

Note that this does not actually submit the job. A detailed explanation for each code snippet of the job script `job.sh` is given below.

```

1  #!/bin/bash

```


This is the standard convention to let the linux shell know what interpreter to run.

```
2  #SBATCH -p compute
3  #SBATCH -N 1
4  #SBATCH -n 4
```

Configuration variables for Slurm start with **SBATCH**.

- **-p** refers to the partition to be used by Slurm. Sharanga provides two partitions, namely, **compute** and **gpu**. For jobs to be executed exclusively on CPUs, we use the **compute** partition.
- **-N** represents the number of nodes to be used. In the present example, we are using 1 node.
- **-n** represents the number of tasks to be executed. For codes employing distributed parallelism, users are requested to specify the number of tasks as the number of compute cores required.

```
5  #SBATCH --mem 512M
6  #SBATCH -t 4-2:23 # time (D-HH:MM)
7  #SBATCH --job-name="hello_test"
```

- **--mem** represents the maximum amount of required memory. Here, we are requesting 512 Megabytes of memory. Note that Slurm prioritises lower memory jobs over higher memory jobs in the queue. This may result in delayed execution of higher memory jobs. Therefore, users are requested to give accurate and desirable memory limits.
- **-t** represents the maximum wall clock time the job requires. Here, we are requesting 4 days, 2 hours and 23 minutes. Slurm prioritises shorter time limit jobs over longer time limit jobs in the queue. This may result in delayed execution of longer time limit jobs. Therefore, users are requested to give accurate and desirable time limits. Note that setting values greater than 168 hours will result in the termination of the job by Slurm automatically.
- **--job-name** represents the name of the job.

```
8  #SBATCH -o slurm.%j.out
9  #SBATCH -e slurm.%j.err
```

- **-o** represents **stdout**.
- **-e** represents **stderr**.

We are instructing Slurm to redirect **stdout** and **stderr** of the executed application to disk. For example, if your **jobid** is 121, then **slurm.121.out** would contain the normal output of the application, while **slurm.121.err** would contain the error output of the application. These files will be stored in the directory, where the jobs were launched from.

```
10 #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
11 #SBATCH --mail-type=ALL
```

- `--mail-user` represents the email address to which job events are to be delivered.
- `--mail-type` represents the type of events to be logged. Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL (equivalent to BEGIN, END, FAIL, REQUEUE, and STAGE_OUT), STAGE_OUT (burst buffer stage out and teardown completed), TIME_LIMIT, TIME_LIMIT_90 (reached 90 percent of time limit), TIME_LIMIT_80 (reached 80 percent of time limit) and TIME_LIMIT_50 (reached 50 percent of time limit). Here, we have specified the event type to be ALL.

```
12    spack load petsc@3.13.1%gcc@9.3.0
```

We are using Spack to set the environment needed to run our application. Our example application depends on `PetSc` and has transitive dependencies `OpenMPI`, `ParMETIS`, `OpenBLAS` and `HDF5`. We are using `gcc 9.3.0` compiled libraries for our application. An advantage of Spack over Modulefiles is that Spack handles all the dependencies automatically in one single command.

```
13    srun ./execname
```

Finally, we are using `srun` to start the execution of the application. This is somewhat analogous to `mpirun`. Users are requested **not** to use `mpirun` and instead use `srun`. `srun` takes care of the allocation and efficient management of resources via Slurm automatically.

4.4 How to schedule a shared memory CPU parallel job on Slurm?

Users can use the `sbatch` command provided by Slurm to submit a job script. Note that, for loading the required packages one can use either Modulefiles or Spack. In the following we have shown example job scripts using Modulefiles and Spack.

4.4.1 Example of a job script `job.sh` using Modulefiles

```
1    #!/bin/bash
2    #SBATCH -p compute
3    #SBATCH -N 1
4    #SBATCH -c 4
5    #SBATCH --mem 512M
6    #SBATCH -t 4-2:23 # time (D-HH:MM)
7    #SBATCH --job-name="hello_test"
8    #SBATCH -o slurm.%j.out
9    #SBATCH -e slurm.%j.err
10   #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
11   #SBATCH --mail-type=ALL
12   export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
13   module load openmpi-3.1.6-gcc-9.3.0
```

```

14 module load parmetis-4.0.3-gcc-9.3.0
15 module load openblas-0.3.10-gcc-9.3.0
16 module load hdf5-1.10.6-gcc-9.3.0
17 module load petsc-3.13.1-gcc-9.3.0
18 srun ./execname

```

To submit the above job script use the following command.

```
$ sbatch job.sh
```

If you wish to test your job script and want to find when it is estimated to run, please run

```
$ sbatch --test-only job.sh
```

Note that this does not actually submit the job. A detailed explanation for each code snippet of the job script `job.sh` is given below.

```
1  #!/bin/bash
```

This is the standard convention to let the linux shell know what interpreter to run.

```

2  #SBATCH -p compute
3  #SBATCH -N 1
4  #SBATCH -c 4

```

Configuration variables for Slurm start with `SBATCH`.

- `-p` refers to the partition to be used by Slurm. Sharanga provides two partitions, namely, `compute` and `gpu`. For jobs to be executed exclusively on CPUs, we use the `compute` partition.
- `-N` represents the number of nodes to be used. In the present example, we are using 1 node. Note that in pure `OpenMP` implementations, the number of nodes is always set to 1. However, if you are using hybrid parallel models such as `MPI+OpenMP`, then the number of nodes can be more than one.
- `-c` represents the number of CPUs per task. For codes employing shared memory parallelism, users are requested to specify the number of threads as the number of compute cores required.

```

5  #SBATCH --mem 512M
6  #SBATCH -t 4-2:23 # time (D-HH:MM)
7  #SBATCH --job-name="hello_test"

```

- `--mem` represents the maximum amount of required memory. Here, we are requesting 512 Megabytes of memory. Note that Slurm prioritises lower memory jobs over higher memory jobs in the queue. This may result in delayed execution of higher memory jobs. Therefore, users are requested to give accurate and desirable memory limits.

- `-t` represents the maximum wall clock time the job requires. Here, we are requesting 4 days, 2 hours and 23 minutes. Slurm prioritises shorter time limit jobs over longer time limit jobs in the queue. This may result in delayed execution of longer time limit jobs. Therefore, users are requested to give accurate and desirable time limits. Note that setting values greater than 168 hours will result in the termination of the job by Slurm automatically.
- `--job-name` represents the name of the job.

```
8  #SBATCH -o slurm.%j.out
9  #SBATCH -e slurm.%j.err
```

- `-o` represents `stdout`.
- `-e` represents `stderr`.

We are instructing Slurm to redirect `stdout` and `stderr` of the executed application to disk. For example, if your `jobid` is 121, then `slurm.121.out` would contain the normal output of the application, while `slurm.121.err` would contain the error output of the application. These files will be stored in the directory, where the jobs were launched from.

```
10  #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
11  #SBATCH --mail-type=ALL
```

- `--mail-user` represents the email address to which job events are to be delivered.
- `--mail-type` represents the type of events to be logged. Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL (equivalent to BEGIN, END, FAIL, REQUEUE, and STAGE_OUT), STAGE_OUT (burst buffer stage out and teardown completed), TIME_LIMIT, TIME_LIMIT_90 (reached 90 percent of time limit), TIME_LIMIT_80 (reached 80 percent of time limit) and TIME_LIMIT_50 (reached 50 percent of time limit). Here, we have specified the event type to be ALL.

```
12  export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
```

The above command sets the environment variable `$OMP_NUM_THREADS`, which specifies the number of threads to be used in parallel regions of the OPENMP code. Here, `$SLURM_CPUS_PER_TASK` is a Slurm defined variable that is automatically set to the value of `-c` directive.

```
13  module load openmpi-3.1.6-gcc-9.3.0
14  module load parmetis-4.0.3-gcc-9.3.0
15  module load openblas-0.3.10-gcc-9.3.0
16  module load hdf5-1.10.6-gcc-9.3.0
17  module load petsc-3.13.1-gcc-9.3.0
```

We are using Modulefiles to set the environment needed to run our application. The example application depends on PETSc and also has transitive dependencies OpenMPI, ParMETIS, OpenBLAS and HDF5. We are using gcc 9.3.0 compiled libraries for the application.

```
18      srun ./execname
```

Finally, we are using `srun` to start the execution of the application. This is somewhat analogous to `mpirun`. Users are requested **not** to use `mpirun` and instead use `srun`. `srun` takes care of the allocation and efficient management of resources via Slurm automatically.

4.4.2 Example of a job script `job.sh` using Spack

```
1      #!/bin/bash
2      #SBATCH -p compute
3      #SBATCH -N 1
4      #SBATCH -c 4
5      #SBATCH --mem 512M
6      #SBATCH -t 4-2:23 # time (D-HH:MM)
7      #SBATCH --job-name="hello_test"
8      #SBATCH -o slurm.%j.out
9      #SBATCH -e slurm.%j.err
10     #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
11     #SBATCH --mail-type=ALL
12     export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
13     spack load petsc@3.13.1%gcc@9.3.0
14     srun ./execname
```

To submit the above job script use the following command.

```
$ sbatch job.sh
```

If you wish to test your job script and want to find when it is estimated to run, please run

```
$ sbatch --test-only job.sh
```

Note that this does not actually submit the job. A detailed explanation for each code snippet of the job script `job.sh` is given below.

```
1      #!/bin/bash
```

This is the standard convention to let the linux shell know what interpreter to run.

```
2      #SBATCH -p compute
3      #SBATCH -N 1
4      #SBATCH -c 4
```

Configuration variables for Slurm start with `SBATCH`.

- `-p` refers to the partition to be used by Slurm. Sharanga provides two partitions, namely, `compute` and `gpu`. For jobs to be executed exclusively on CPUs, we use the `compute` partition.

- `-N` represents the number of nodes to be used. In the present example, we are using 1 node. Note that in pure `OpenMP` implementations, the number of nodes is always set to 1. However, if you are using hybrid parallel models such as `MPI+OpenMP`, then the number of nodes can be more than one.
- `-c` represents the number of CPUs per task. For codes employing shared memory parallelism, users are requested to specify the number of threads as the number of compute cores required.

```

5  #SBATCH --mem 512M
6  #SBATCH -t 4-2:23 # time (D-HH:MM)
7  #SBATCH --job-name="hello_test"

```

- `--mem` represents the maximum amount of required memory. Here, we are requesting 512 Megabytes of memory. Note that Slurm prioritises lower memory jobs over higher memory jobs in the queue. This may result in delayed execution of higher memory jobs. Therefore, users are requested to give accurate and desirable memory limits.
- `-t` represents the maximum wall clock time the job requires. Here, we are requesting 4 days, 2 hours and 23 minutes. Slurm prioritises shorter time limit jobs over longer time limit jobs in the queue. This may result in delayed execution of longer time limit jobs. Therefore, users are requested to give accurate and desirable time limits. Note that setting values greater than 168 hours will result in the termination of the job by Slurm automatically.
- `--job-name` represents the name of the job.

```

8  #SBATCH -o slurm.%j.out
9  #SBATCH -e slurm.%j.err

```

- `-o` represents `stdout`.
- `-e` represents `stderr`.

We are instructing Slurm to redirect `stdout` and `stderr` of the executed application to disk. For example, if your `jobid` is 121, then `slurm.121.out` would contain the normal output of the application, while `slurm.121.err` would contain the error output of the application. These files will be stored in the directory, where the jobs were launched from.

```

10 #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
11 #SBATCH --mail-type=ALL

```

- `--mail-user` represents the email address to which job events are to be delivered.
- `--mail-type` represents the type of events to be logged. Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL (equivalent to BEGIN, END, FAIL, REQUEUE, and STAGE_OUT), STAGE_OUT (burst buffer stage out and teardown completed), TIME_LIMIT,

TIME_LIMIT_90 (reached 90 percent of time limit), TIME_LIMIT_80 (reached 80 percent of time limit) and TIME_LIMIT_50 (reached 50 percent of time limit). Here, we have specified the event type to be ALL.

```
12 export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
```

The above command sets the environment variable `OMP_NUM_THREADS`, which specifies the number of threads to be used in parallel regions of the `OPENMP` code. Here, `SLURM_CPUS_PER_TASK` is a Slurm defined variable that is automatically set to the value of `-c` directive.

```
13 spack load petsc@3.13.1%gcc@9.3.0
```

We are using Spack to set the environment needed to run our application. Our example application depends on `PetSc` and has transitive dependencies `OpenMPI`, `ParMETIS`, `OpenBLAS` and `HDF5`. We are using `gcc 9.3.0` compiled libraries for our application. An advantage of Spack over `Modulefiles` is that Spack handles all the dependencies automatically in one single command.

```
14 srun ./execname
```

Finally, we are using `srun` to start the execution of the application. This is somewhat analogous to `mpirun`. Users are requested **not** to use `mpirun` and instead use `srun`. `srun` takes care of the allocation and efficient management of resources via Slurm automatically.

4.5 How to schedule a GPU parallel job on Slurm?

Users can use the `sbatch` command provided by Slurm to submit a job script. Note that, for loading the required packages one can use either `Modulefiles` or `Spack`. In the following we have shown example job scripts using `Modulefiles` and `Spack`.

4.5.1 Example of a job script using Modulefiles

```
1  #!/bin/bash
2  #SBATCH -p gpu
3  #SBATCH -N 1
4  #SBATCH -n 1
5  #SBATCH --mem 512M
6  #SBATCH -t 0-4:51 # time (D-HH:MM)
7  #SBATCH --job-name="hello_test"
8  #SBATCH -o slurm.%j.out
9  #SBATCH -e slurm.%j.err
10 #SBATCH --gres=gpu:1
11 #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
12 #SBATCH --mail-type=ALL
```

```

13 module load cuda-11.0.2
14 srun ./execname

```

To submit the above job script use the following command.

```
$ sbatch job.sh
```

If you wish to test your job script and want to find when it is estimated to run, please run

```
$ sbatch --test-only job.sh
```

Note that this does not actually submit the job. A detailed explanation for each code snippet of the job script `job.sh` is given below.

```
1  #!/bin/bash
```

This is the standard convention to let the linux shell know what interpreter to run.

```

2  #SBATCH -p gpu
3  #SBATCH -N 1
4  #SBATCH -n 1

```

Configuration variables for Slurm start with **SBATCH**.

- **-p** refers to the partition to be used by Slurm. Sharanga provides two partitions, namely, **compute** and **gpu**. For jobs to be executed exclusively on GPUs, we use the **gpu** partition.
- **-N** represents the number of nodes to be used. In the present example, we are using 1 node. Note that, at present, Sharanga has provision for only one GPU node.
- **-n** represents the number of tasks to be executed. For codes employing distributed parallelism such as GPGPUs or hybrid parallel models based on CPUs and GPUs, users are requested to specify the number of tasks as the number of compute cores required. In the present example, *n* is set to 1.

```

5  #SBATCH --mem 512M
6  #SBATCH -t 0-4:51 # time (D-HH:MM)
7  #SBATCH --job-name="hello_test"

```

- **--mem** represents the maximum amount of required memory. Here, we are requesting 512 Megabytes of memory. Note that Slurm prioritises lower memory jobs over higher memory jobs in the queue. This may result in delayed execution of higher memory jobs. Therefore, users are requested to give accurate and desirable memory limits.
- **-t** represents the maximum wall clock time the job requires. Here, we are requesting 0 days, 4 hours and 51 minutes. Slurm prioritises shorter time limit jobs over longer time limit jobs in the queue. This may result in delayed execution of longer time limit jobs. Therefore, users are requested to give accurate and desirable time limits. Note that setting values greater than 24 hours will result in the termination of the job by Slurm automatically.

- `--job-name` represents the name of the job.

```
8  #SBATCH -o slurm.%j.out
9  #SBATCH -e slurm.%j.err
```

- `-o` represents `stdout`.
- `-e` represents `stderr`.

We are instructing Slurm to redirect `stdout` and `stderr` of the executed application to disk. For example, if your `jobid` is 121, then `slurm.121.out` would contain the normal output of the application, while `slurm.121.err` would contain the error output of the application. These files will be stored in the directory, where the jobs were launched from.

```
10  #SBATCH --gres=gpu:1
```

- `-gres` represents **generic resource**. Here, we are informing Slurm that our job requires 1 GPU card of any type.

We are instructing Slurm to redirect `stdout` and `stderr` of the executed application to disk. For example, if your `jobid` is 121, then `slurm.121.out` would contain the normal output of the application, while `slurm.121.err` would contain the error output of the application. These files will be stored in the directory, where the jobs were launched from.

```
11  #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
12  #SBATCH --mail-type=ALL
```

- `--mail-user` represents the email address to which job events are to be delivered.
- `--mail-type` represents the type of events to be logged. Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL (equivalent to BEGIN, END, FAIL, REQUEUE, and STAGE_OUT), STAGE_OUT (burst buffer stage out and teardown completed), TIME_LIMIT, TIME_LIMIT_90 (reached 90 percent of time limit), TIME_LIMIT_80 (reached 80 percent of time limit) and TIME_LIMIT_50 (reached 50 percent of time limit). Here, we have specified the event type to be ALL.

```
13  module load cuda-11.0.2
```

We are using Modulefiles to set the environment needed to run our application. The example application depends on CUDA, which is an Nvidia framework for allowing users to utilise GPUs.

```
14  srun ./execname
```

Finally, we are using `srun` to start the execution of the application. This is somewhat analogous to `mpirun`. Users are requested **not** to use `mpirun` and instead use `srun`. `srun` takes care of the allocation and efficient management of resources via Slurm automatically.

4.5.2 Example of a job script using Spack

```
1  #!/bin/bash
2  #SBATCH -p gpu
3  #SBATCH -N 1
4  #SBATCH -n 1
5  #SBATCH --mem 512M
6  #SBATCH -t 0-4:51 # time (D-HH:MM)
7  #SBATCH --job-name="hello_test"
8  #SBATCH -o slurm.%j.out
9  #SBATCH -e slurm.%j.err
10 #SBATCH --gres=gpu:1
11 #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
12 #SBATCH --mail-type=ALL
13 spack load cuda@11.0.2
14 srunk ./execname
```

To submit the above job script use the following command.

```
$ sbatch job.sh
```

If you wish to test your job script and want to find when it is estimated to run, please run

```
$ sbatch --test-only job.sh
```

Note that this does not actually submit the job. A detailed explanation for each code snippet of the job script `job.sh` is given below.

```
1  #!/bin/bash
```

This is the standard convention to let the linux shell know what interpreter to run.

```
2  #SBATCH -p gpu
3  #SBATCH -N 1
4  #SBATCH -n 1
```

Configuration variables for Slurm start with `SBATCH`.

- `-p` refers to the partition to be used by Slurm. Sharanga provides two partitions, namely, `compute` and `gpu`. For jobs to be executed exclusively on GPUs, we use the `gpu` partition.
- `-N` represents the number of nodes to be used. In the present example, we are using 1 node. Note that, at present, Sharanga has provision for only one GPU node.
- `-n` represents the number of tasks to be executed. For codes employing distributed parallelism such as GPGPUs or hybrid parallel models based on CPUs and GPUs, users are requested to specify the number of tasks as the number of compute cores required. In the present example, `n` is set to 1.

```

5  #SBATCH --mem 512M
6  #SBATCH -t 0-4:51 # time (D-HH:MM)
7  #SBATCH --job-name="hello_test"

```

- `--mem` represents the maximum amount of required memory. Here, we are requesting 512 Megabytes of memory. Note that Slurm prioritises lower memory jobs over higher memory jobs in the queue. This may result in delayed execution of higher memory jobs. Therefore, users are requested to give accurate and desirable memory limits.
- `-t` represents the maximum wall clock time the job requires. Here, we are requesting 0 days, 4 hours and 51 minutes. Slurm prioritises shorter time limit jobs over longer time limit jobs in the queue. This may result in delayed execution of longer time limit jobs. Therefore, users are requested to give accurate and desirable time limits. Note that setting values greater than 24 hours will result in the termination of the job by Slurm automatically.
- `--job-name` represents the name of the job.

```

8  #SBATCH -o slurm.%j.out
9  #SBATCH -e slurm.%j.err

```

- `-o` represents `stdout`.
- `-e` represents `stderr`.

```

10 #SBATCH --gres=gpu:1

```

- `-gres` represents **generic resource**. Here, we are informing Slurm that our job requires 1 GPU card of any type.

We are instructing Slurm to redirect `stdout` and `stderr` of the executed application to disk. For example, if your `jobid` is 121, then `slurm.121.out` would contain the normal output of the application, while `slurm.121.err` would contain the error output of the application. These files will be stored in the directory, where the jobs were launched from.

```

11 #SBATCH --mail-user=<username>@hyderabad.bits-pilani.ac.in
12 #SBATCH --mail-type=ALL

```

- `--mail-user` represents the email address to which job events are to be delivered.
- `--mail-type` represents the type of events to be logged. Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL (equivalent to BEGIN, END, FAIL, REQUEUE, and STAGE_OUT), STAGE_OUT (burst buffer stage out and teardown completed), TIME_LIMIT, TIME_LIMIT_90 (reached 90 percent of time limit), TIME_LIMIT_80 (reached 80 percent of time limit) and TIME_LIMIT_50 (reached 50 percent of time limit). Here, we have specified the event type to be `ALL`.

13 `spack load cuda@11.0.2`

We are using Spack to set the environment needed to run our application. The example application depends on `CUDA`, which is an Nvidia framework for allowing users to utilise GPUs.

14 `srun ./execname`

Finally, we are using `srun` to start the execution of the application. This is somewhat analogous to `mpirun`. Users are requested **not** to use `mpirun` and instead use `srun`. `srun` takes care of the allocation and efficient management of resources via Slurm automatically.

4.6 How to view the status of partitions?

Users can use the `sinfo` command to view the status of partitions and nodes.

```
$ sinfo
```

Alternatively, users can run `scontrol show partition` to know about partitions and their limits.

```
$ scontrol show partition
```

4.7 How to view my submitted jobs?

To view all the current jobs of a user, please type the following command.

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

To view all the running jobs of a user, please type

```
$ squeue -u <username> -t RUNNING
```

4.8 How to cancel my job?

To cancel a particular job by its `jobid`, use the following command.

```
$ scancel <jobid>
```

To cancel a job by its name, please type

```
$ scancel --name <jobname>
```

To cancel all the jobs of a user

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

To cancel all the `PENDING` jobs of a user

```
$ scancel -t PENDING -u <username>
```

4.9 How to control my job?

To hold a job from being scheduled

```
$ scontrol hold <jobid>
```

To release a job to be scheduled

```
$ scontrol release <jobid>
```

To requeue (cancel or rerun) a job

```
$ scontrol requeue <jobid>
```

For more information, users can use `man scontrol`, `man squeue`, `man scancel`, etc.

4.10 Can I run my code for a few minutes on the login node?

Users are not authorized to run their codes on the login node. Codes running on login nodes will be terminated automatically. Jobs **have** to be submitted through the scheduler.

4.11 How can I monitor the output of my jobs that are running?

Users can access the output and error logs generated by Slurm while running their codes using the `tail` command. Users are strongly recommended to use the example scripts given above as a base for their job scripts. Job logs are written to the current directory, from which the Slurm job was executed by `sbatch`.

Please note that by default `sbatch` does not write the output logs. The example provided above instructs Slurm to dump the output in the format of `slurm.<jobid>.out`, where `<jobid>` is the job id of the slurm job. Users can use the `tail` command with `-f` flag to view the output file being written continuously by Slurm.

For example, if a user has a job with the job id 121, and would like to view the job output, then use the following syntax.

```
$ tail -f slurm.121.out
```

Alternatively, users can use `vi` or `nano` to view the output file after the job is completed.

```
$ vi slurm.121.out
```

```
$ nano slurm.121.out
```

4.12 Can I run interactive jobs?

Yes, the facility provides users access to interactive job scheduling. Users can use the `srun` command to setup an interactive session on the nodes. Consider an example where a user wants to request 2 nodes with 64 cores per node.

```
$ srun -N 2 -n 1 -c 128 -p compute --pty bash -i
```

Here, `-N` is the number of nodes to be used. `-n` is the number of tasks (instances) to be executed. `-c` is the number of cores required for the task. `-p` is the partition to be used. In the present example, we chose the `compute` partition. `--pty bash` instructs Slurm to setup a pseudo terminal in bash. `-i` tells Slurm to run this command in interactive mode.

4.13 I have job scripts written for another job scheduler. Can I use them for Slurm?

No, you cannot execute job scripts written for other job schedulers using Slurm. But, the developers of Slurm have provided users with a documentation containing the correspondences between the options of several job schedulers. Users can access the documentation at this link. [Rosetta stone of Workload manager](#).

4.14 When is my job going to start to run?

To get an estimate of when your job is going to start, users can use the `squeue` command.

```
$ squeue --start -j <jobid>
```

Please note that your job might run before the scheduled start time as jobs that have finished earlier than their requested walltime might free up the queue and resources needed for your job.

4.15 Why is my job not running?

There are several reasons why your job is not running. Users can run the `squeue` command to get the status and reason.

```
$ squeue -j <jobid> -l
```

The `NODELIST(REASON)` section in the output of the above command will have the reason, why Slurm is unable to run the job. Here are the most common reasons.

BadConstraints

The job's constraints can not be satisfied.

Cleaning

The job is being requeued and still cleaning up from its previous execution.

Dependency

This job is waiting for a dependent job to complete.

JobHeldAdmin

The job is held by a system administrator. Please contact the system administrator for more information.

JobHeldUser

The job is held by the user.

NonZeroExitCode

The job terminated with a non-zero exit code.

PartitionDown

The partition required by this job is in a DOWN state.

Priority

One or more higher priority jobs exist for this partition or advanced reservation.

QOSResourceLimit

The job's Quality of Service (QOS) has reached some resource limit.

ReqNodeNotAvail

Some node specifically required by the job is not currently available.

Resources

The job is waiting for resources to become available.

TimeLimit

The job exhausted its time limit.

Chapter 5

Package Management with Modulefiles

5.1 What are modulefiles?

Modulefiles are files that are used to setup the local environment to configure various software packages.

5.2 How to view the list of available packages in the facility?

On Sharanga, users can use the following command to view the list of installed packages (with the versions).

```
$ module avail
----- /share/module-----
2brad_denovo/2019-01-22_giteec5016      meshclust2/2.1.0
2brad_gatk/2019-01-22_git1fcc9e8      metal/2010-02-08
admixture/1.3.0                        metal/2011-03-25
angsd/0.923                           metalge/2010-02-08
annovar/2018apr                       minimac2/2014-09-15
artemis/18.0.3                        minimac3/2.0.1
augustus/3.3.2                       minimac4/1.0.0
bamtools/2.5.1                       mirdeep2/0.1.0
bamutil/1.0.14                       mixcr/3.0.3
basemount/0.15.103.3011              mmap/2018-04-07
basespace-cli/0.8.12.590             morgan/3.2
basespace-cli/0.9.17                 morgan/3.4
basespace-cli/0.10.8                mosdepth/0.2.6
bayescan/2.1                        mothur/1.35.0
bbmap/38.16                         multiqc/1.6
bcbio/1.1.1                         mummer/3.23
```

5.3 How to load and unload a package(s)?

Users can use the `module load` command to load a package into their current environment. For example, a user who wishes to use PETSc library can use the following syntax.

```
$ module load petsc
```

If a user wishes to unload the PETSc library from their user environment, they can use the `module rm` command.

```
$ module rm petsc
```

If a user wishes to unload all the loaded libraries from their user environment, then they can use the following command.

```
$ module purge
```

5.4 How to view the list of loaded packages?

Users can use the following command to find the list of currently loaded modules.

```
$ module list
```

5.5 How do I check the version of the package installed?

Users can type the following command to find the version of the installed package.

```
$ module spider <package name>
```

Note that the user may get additional information about the package by using the following command

```
$ module help <package name>
```

5.6 After submitting a job, I am getting messages like “module: command not found”. Why am I getting this message?

Modulefiles are by default sourced by the default shell `/bin/bash`. You may get this message if you have changed the shell to something other than `bash`. We request you to change the shell back to `bash` for optimal functioning. Users can use the `chsh` shell command to change their user shell environment back to `bash`. Another reason could be due to the unavailability of the desired package.

Chapter 6

Package Management with Spack

6.1 What is Spack?

The HPC facility provides users access to Spack, a package management tool designed to support multiple versions and configurations of a software on a wide variety of platforms and environments.

6.2 What is the difference between Modulefiles and Spack?

Modulefiles are either generated by the package or by the administrator. On the other hand, Spack is a superset of modulefiles and generates the appropriate modulefiles for the installed package. An advantage of Spack is that it offers dependency management for packages.

6.3 How to view the list of available packages installed in the facility?

Users can use the `spack` command to view the list of installed packages.

```
$ spack find
==> 45 installed packages

-- linux-centos8-broadwell / gcc@7.4.0 -----
armadillo@9.800.3  cmake@3.17.3  libpciaccess@0.13.5  m4@1.4.18      openmpi@3.1.6
arpack-ng@3.7.0   gdbm@1.18.1    libsigsegv@2.12      ncurses@6.2    openssl@1.1.1g
autoconf@2.69     hwloc@1.11.11  libtool@2.4.6        numactl@2.0.12 perl@5.30.3
automake@1.16.2   libiconv@1.16  libxml2@2.9.10       openblas@0.3.10 pkgconf@1.7.3

-- linux-centos8-broadwell / gcc@8.3.1 -----
autoconf@2.69     expat@2.2.9    isl@0.20              libsigsegv@2.12 mpfr@3.1.6
automake@1.16.2   gcc@9.3.0      libbsd@0.10.0         libtool@2.4.6   mpfr@3.1.6
bzip2@1.0.8       gdbm@1.18.1    libedit@3.1-20170329  m4@1.4.18      ncurses@6.1
cmake@3.16.2      gettext@0.20.1 libffi@3.2.1          mpc@1.1.0       ncurses@6.2
diffutils@3.7     gmp@6.1.2      libiconv@1.16         mpc@1.1.0       openssl@1.1.1d
```

Note that the above list is for representational purpose and differs from the actual output on Sharanga.

6.4 How to load and unload a package(s)?

Users can use the `spack load` command to load the package into their current environment. For example a user who wishes to use PETSc library can use the following syntax.

```
$ spack load petsc
```

If a user wishes to unload the PETSc library from their environment, they can use the `spack unload` command.

```
$ spack unload petsc
```

If a user wishes to unload all the loaded libraries from their user environment, please use the following command.

```
$ spack unload
```

6.5 How to view the list of loaded packages?

Users can use the following command to find the list of currently loaded packages.

```
$ spack find --loaded
```

6.6 How do I check the version of the package installed?

Users can type `spack find <package name>` to find the version of the package installed. Note that the package version is shown after `@` in the output.

6.7 After submitting a job, I am getting messages like “spack: command not found”. Why am I getting this message?

Spack is sourced by the default shell `/bin/bash`. You may get this message if you have changed the shell to something other than `bash`. We request you to change the shell back to `bash` for optimal functioning. Users can use the `chsh` shell command to change their user shell environment back to `bash`. Another reason could be due to the unavailability of the desired package.

6.8 Between Modulefiles and Spack, which is more user friendly?

Spack is more user friendly and also offers `bash` auto-completion for its commands. It simplifies package management and version control for the system administrator.

Chapter 7

Miscellaneous

7.1 How do I acknowledge the high performance computing facility in my presentations or publications?

We request the users to kindly use the following text to acknowledge the HPC resources.

“The authors gratefully acknowledge the computing time provided on the high performance computing facility, Sharanga, at the Birla Institute of Technology and Science - Pilani, Hyderabad Campus.”

7.2 Is HyperThreading enabled on the compute nodes?

HyperThreading, also known as `Simultaneous multithreading` by AMD is disabled on the compute nodes as we observed several performance regressions and compatibility issues.

7.3 Can I launch HTML files on the cluster?

It is well-known that the browser applications consume a lot of system memory and may hamper the performance of applications. Due to this, we have disabled the ability to forward X Server on the nodes. Therefore, it is not possible to launch HTML files on the cluster.

7.4 Can I launch GUI applications such as gnuplot?

Similar to the browser applications, GUI applications (like gnuplot) too consume a lot of memory. Under heavy load, these applications may result in out of memory and disconnection logs. Therefore, we have disabled GUI on the cluster.

7.5 While executing a command, I am getting the error “undefined symbol: EVP_KDF_ctrl, version OPENSSL_1.1.1b”.

To resolve this issue, please type, `spack unload openssl` in the terminal.

7.6 After launching `nano` or `vi(m)`, the application throws a segmentation fault.

To resolve this error, please type `spack unload openssl` in the terminal.