NCRA Compute Cluster (Bhima) | User Document

We are using OpenPBS (Portable Batch System) for NCRA Compute Cluster for workload management system, whcih is designed to manage and schedule the execution of jobs on a compute cluster. This guide provides basic instructions for users to efficiently use OpenPBS.

Bhima Cluster Architecture

Total CPUs in cluster are 248 cores (8 cores reserved for cluster management)

Compute Nodes configuration - 3 Numbers

- CPU: Intel(R) Xeon(R) Platinum 8358
- CPU(s): 64
- On-line CPU(s) list: 0-63
- Thread(s) per core: 1
- Core(s) per socket: 32
- No. of Sockets: 2
- RAM: 2 TB

GPU Node configuration - 1 Number

- CPU: Intel(R) Xeon(R) Platinum 8358
- CPU(s): 64
- On-line CPU(s) list: 0-63
- Thread(s) per core: 1
- Core(s) per socket: 32
- No. of Sockets: 2
- GPU: NVIDIA H100 - 80GB
- RAM: 1TB

Storage 600 TB total

- Each user will have a specific quota in the `/home/<your_username>` directory. This directory is where you should store everything, including your processing data, processed data, and codes.
  There is no separate `/data` directory; instead, everything should be saved within your `/home/<your_username>` directory. So, please make sure that all your work, including data and code, is organized in this location.

- /scratch area is a common area writable by everyone and will be cleared in 7 days - backup up your data (we will not send any reminders about this area utilization)

Other Information

- Bhima (bhimacluster) login is different from NCRA dhruva/mail LDAP login, you need to request of this cluster user account and get approval from Chair CFC.
- there will be no backup of any of your data, please make sure you take regular backups.
- System administrators or any computer staff are not responsible for bugs, memory leaks, garbage collector issues, security vulnerabilities, or any other issues in user code. Please review and test your code thoroughly before submitting it to the cluster.

Table of Contents

1. Getting Started

Accessing the Cluster
To access the cluster, use SSH (Secure Shell) to connect to the head node:

```
ssh your_username@cluster_address
```

Replace `your_username` with your actual username and `cluster_address` with the address of the cluster.

```
ssh ratnakumar@bhima.ncra.tifr.res.in
```

2. Submitting Jobs

Basic Job Submission

> **Important Note**: Any jobs executed or run without PBS scheduling will be terminated without any warning. Please ensure that all job submissions go through the PBS scheduler to avoid any disruptions.

To submit a job, use the `qsub` command followed by the job script:

```
qsub my_script.pbs
```

3. Monitoring Jobs

Checking Job Status

To check the status of your jobs, use the `qstat` command:

```
qstat
```

To get detailed information about a specific job:

```
qstat -f job_id
```

Checking Quotas Status

```
[ratnakumar@bhima03 ~]$ /soft/my_quota
Disk quotas for usr ratnakumar (uid 415800004):
    Filesystem   used   quota   limit   grace   files   quota   limit   grace
/home/ratnakumar
             75.02G    95G    100G      -  150556       0       0       -
uid 415800004 is using default file quota setting
```

> ratnakumar@bhima03 ~]$ /soft/my_quota

4. Job Control

Deleting Jobs

To delete a job, use the `qdel` command followed by the job ID:

```
qdel job_id
```

Holding and Releasing Jobs

To hold a job:

```
qhold job_id
```

To release a held job:

```
qrls job_id
```

5. Resource Requests

Specifying Resources for CPU

Specify resources in the job script or on the command line. For example:

```
#PBS -l nodes=2:ppn=4,walltime=01:00:00
```

This requests 2 nodes with 4 processors per node and a wall time of 1 hours.

Specifying Resources for GPU

Sample PBS job submission script:

```
#!/bin/bash
#PBS -N cuda_integral_job
#PBS -l select=1:ngpus=1:host=bhima04
#PBS -l walltime=02:00:00
#PBS -q gpu
#PBS -j oe
#PBS -o cuda_output.log

# Load CUDA module (if required)
module load cuda/12.3
module load python/3.8

# Navigate to the working directory
cd $PBS_O_WORKDIR

# Run the program
python3.8 check_gpu.py --gpu --size 50000
```

```
## With CPUs and memory specification

#!/bin/bash
#PBS -N cuda_integral_mem_job
#PBS -l select=1:ncpus=4:ngpus=1:mem=32gb:host=bhima04
#PBS -l walltime=02:00:00
#PBS -q gpu
#PBS -j oe
#PBS -o cuda_output.log

# Load CUDA module (if required)
module load cuda/12.3
module load python/3.8

# Navigate to the working directory
cd $PBS_O_WORKDIR

# Run the program
python3.8 check_gpu_cpu_mem.py --gpu --size 50000
```

Following ***`-l select`*** and ***`-q gpu`*** are important flags for GPU scheduling your job in **bhima04** GPU Node.

> Only bhima04 Node is having NVIDIA H100 GPU - use python3.8 and pip3.8 for gpu node

> PBS -l select=1:ngpus=1:host=bhima04

> #PBS -q gpu

Common Resource Requests

- **Nodes and Processors:** `nodes=number_of_nodes:ppn=processors_per_node`
- **Memory:** `mem=memory_amount`
- **Wall Time:** `walltime=hh:mm:ss`

6. PBS Scripts

Basic Script Structure

A basic PBS script includes resource requests, environment settings, and job commands:

```
#!/bin/bash
#PBS -N job_name
#PBS -l nodes=1:ppn=4,walltime=01:00:00
#PBS -q workq
#PBS -j oe

cd $PBS_O_WORKDIR
module load some_module

./your_executable
```

Sample CPU and node (either bhima01, bhima02 or bhima04) specific PBS submit script

Usually, all users are running code on default login node bhima/bhima03. It is better to see which nodes are free and set the host flag to either bhima01, bhima02 or bhima04. - We are working to do it automatically and will update this document once it gets enabled.

```
#!/bin/bash
#PBS -N job_name
#PBS -q workq
#PBS -l select=1:host=bhima02:ncpus=10
#PBS -j oe
#PBS -V
cd $PBS_O_WORKDIR

source /home/ratnakumar/.bashrc
python3 run_some_parallel_code.py
```

Sample CPU PBS submit script

```
#!/bin/bash
#PBS -N job_montecarlo
#PBS -q workq
#PBS -l nodes=4:ppn=4
#PBS -j oe
#PBS -V
cd $PBS_O_WORKDIR
mpiexec -np 16 -machinefile $PBS_NODEFILE ./mpi_montecarlo
```

Common Directives

> - `#PBS -N job_name` - Sets the job name.
> - `#PBS -l nodes=1:ppn=4` - Requests 1 node with 4 processors.
> - `#PBS -q workq` - Specifies the queue - we have '*workq*' for cpu based and '***gpu***' queue
> - `#PBS -j oe` - Merges standard output and error files.

7. Common Commands

> - **Submit a job:** `qsub job_script.pbs`
> - **Check job status:** `qstat`
> - **Delete a job:** `qdel job_id`
> - **Hold a job:** `qhold job_id`
> - **Release a job:** `qrls job_id`
> - **Show job details:** `qstat -f job_id`

8. Best Practices

1. **Test Scripts Locally:** Test your scripts on a local machine before submitting to the cluster.
2. **Specify Resources Accurately:** Request only the resources you need to optimize cluster utilization.
3. **Monitor Jobs Regularly:** Keep an eye on your jobs to handle any issues promptly.
4. **Cleanup:** Remove unnecessary files and jobs to keep the cluster environment clean.

9. "module" command

To improve the efficiency and management of software installations on our cluster, we are introducing a new policy regarding software management:

Software Policy

1. **No Individual Software Areas in `/home/username`:**
   Please refrain from installing or maintaining individual copies of software in your `/home/username` directories. This practice leads to redundancy, wasted storage, and

increased maintenance overhead.

2. **Centralized Software Installation in `/soft`:**
   We will maintain a centralized software repository in the `/soft` directory. All commonly used software will be installed here and configured for easy access by all users. Please let us know what all software are required, we will install them as per the requirement - write to **system administrator** with subject as **"*Bhima software installation*"**
3. **Accessing Software via `module load`:**
   To use any software from the centralized `/soft` directory, simply use the `module load <software_name/vesrsion>` command. This ensures that everyone is using the same, up-to-date version of the software, and it helps to optimize our resource utilization, also if required to maintain the same software with multiple versions.

Benefits

- **Reduced Redundancy**
  By avoiding multiple installations of the same software, we can free up storage space and reduce duplication.
- **Easier Management**
  A central repository allows for easier updates, patches, and management of software.
- **Consistent Environment**
  All users will have access to the same versions of software, reducing compatibility issues and ensuring consistency in results.

Implementation

1. **Migration to `/soft`**
   If you have software installed in your `/home/username` directory that you believe is useful to others, please inform the **system administrator.** We will help migrate the software to the `/soft` directory and make it available to everyone.
2. **Requesting New Software**
   If you need new software that is not currently available in `/soft`, please submit a request to the system administrator with subject as **"*Bhima software installation*"**. We will evaluate the request and, if appropriate, install it centrally.
3. **Regular Updates**
   The `/soft` directory will be regularly updated with the latest versions of software, ensuring you have access to the most current tools.

Brief Explanation of Modules

Modules are a system used in Unix-like operating systems, including many Linux distributions, to manage the environment for different software packages. The **Environment Modules** package allows users to dynamically modify their environment (like `PATH`, `LD_LIBRARY_PATH`, etc.) by loading and unloading module files. This is particularly useful on multi-user systems, such as clusters, where multiple versions of software might be needed by different users.

How Modules Work?

- **Module Files**: These are scripts that set or unset environment variables, typically stored in a directory like `/etc/modulefiles` or `/usr/share/modules/modulefiles or /soft/modulefiles`
- **Loading a Module**: When you load a module using `module load <module_name>`, the module file modifies your environment so that the specified software package or version becomes active.
- **Unloading a Module**: Using `module unload <module_name>`, the changes made by the module file are undone, returning your environment to its previous state.

Example of Module file

```
#%Module1.0#####################################################################
##########
##
## use.own modulefile
##
proc ModulesHelp { } {
   puts stderr "\tThis module file will add \$HOME/privatemodules to the"
   puts stderr "\tlist of directories that the module command will search"
   puts stderr "\tfor modules.  Place your own module files here."
   puts stderr "\tThis module, when loaded, will create this directory"
   puts stderr "\tif necessary."
}

module-whatis   "adds your own modulefiles directory to MODULEPATH"

eval set  [ array get env HOME ]
set ownmoddir   $HOME/privatemodules

# create directory if necessary
if [ module-info mode load ] {
   if { ! [ file exists $ownmoddir ] } {
      file mkdir $ownmoddir
      set null [open $ownmoddir/null w]
      puts $null
"#%Module#####################################################################
############"
      puts $null "##"
      puts $null "## null modulefile"
      puts $null "##"
```

```
    puts $null "proc ModulesHelp { } {"
    puts $null "    puts stderr \"\tThis module does absolutely nothing.\""
    puts $null "    puts stderr \"\tIt's meant simply as a place holder in your\""
    puts $null "    puts stderr \"\tdot file initialization.\""
    puts $null "}"
    puts $null ""
    puts $null "module-whatis   \"does absolutely nothing\""
  }
}

module use --append $ownmoddir
```

"module" command usage/example

- **Load a Module**:
  `module load <module_name>`
- **Unload a Module**:
  `module unload <module_name>`
- **List Available Modules**:
  `module avail`
- **Show Currently Loaded Modules**:
  `module list`

Advantages of "module"

- **Easy Software Management**: You can switch between software versions without manually changing environment variables.
- **Centralized Software Installation**: Instead of duplicating software installations, you can load the same software from a shared directory.
- **User-Specific Environments**: You can create and manage their environment without affecting others on the system.

10. How to user PSRSOFT?

To use the PSRSOFT commands (includes tempo2 & presto) please do the following:

- Check the file /soft/psrsoft.bashrc
- contents of /soft/psrsoft.bashrc

```
export PSRSOFT=/soft/psrsoft/usr


export PRESTO=$PSRSOFT/src/presto
export LD_LIBRARY_PATH=$PSRSOFT/lib:$PRESTO/lib/:/usr/local/cuda/lib64:$LD_LIBRARY_PATH
export
```

```
PATH=$PSRSOFT/bin:/soft/plotres/:$PRESTO/bin/:/soft/psrsoft/usr/src/fake_simulation/bin:/soft/psrsoft/usr/src/ffancy:/usr/local/cuda/bin:$PATH

export TEMPO=$PSRSOFT/src/tempo
export TEMPO2=$PSRSOFT/src/tempo2
export SIGPROC=$PSRSOFT/src/sixproc
export PGPLOT_DIR=$PSRSOFT/src/pgplot
export PGPLOT_FONT=$PGPLOT_DIR/grfont.dat
```

- either copy the contents of this file to your .bashrc
- OR you can run as sh +x /soft/psrsoft.bashrc

11. Policy

Policy for allocating resources in the new NCRA compute servers

Version 2.0, Date: 26 July 2024

New Compute Server Specifications:

- Server 1: 64 cores, 2TB RAM
- Server 2: 64 cores, 2TB RAM
- Server 3: 64 cores, 2TB RAM
- Server 4: 64 cores, 1TB RAM, Nvidia H100 80GB GPU
- Storage: 600 TB Usable

User Policy Framework by User Group (No limit on memory):

User Type	**Resource Allocation**	**Usage Limits**	**Storage**	**Retention Period**
Faculty	Maximum of 96 cores per user; Access to GPU resources on Server 4*	Up to 5000 CPU hours per month, following which jobs will be run on lower priority.	10 TB	3 years
Postdoc			5 TB	5 years
Regular students			5 TB	5 years
Group processing	No reserved computing resources			

* Jobs requiring GPU would be automatically allocated in server 4 with "#PBS -q gpu" flag. An appropriate flag needs to be included for such jobs while submission.

This resource allocation policy will be revised upwards in 2-3 months, depending on the usage pattern and occupancy rate of the cores, in order to optimize and make the best use of the available resources.

Common Policies for All User Groups:

1. **User Registration:** Users need to request for an account on these servers, separate from their usual NCRA user account.
2. **Job Submission and Queue Management:** Jobs must be submitted through Open PBS with proper resource requests specified. The queue system will enforce the fair share policy to prevent resource monopolisation.
3. **Resource Monitoring:** Users must monitor their jobs and ensure they do not exceed allocated resources. Jobs found consuming more resources than allocated will be terminated after a warning. Similarly, jobs that are stale/inactive for more than 24 hours would be terminated after a warning.
4. **Maintenance and Downtime:** Regular maintenance windows will be communicated to all users. Plan jobs around these windows to avoid disruptions. Proposing maintenance twice in a year for electrical UPS maintenance.
5. **Administrative Oversight:** The computer centre staff will monitor usage patterns and adjust allocations as necessary to ensure fairness and optimal performance. User feedback will be collected periodically to refine policies and resource allocation strategies.
6. **User Responsibilities:**
   - **Adherence to Policies:** Users must comply with the outlined policies. Violations may result in access suspension.
   - **Resource Optimization:** Users should optimise their code and resource usage to maximise efficiency.
   - **Job Management:** Ensure jobs are terminated upon completion to free up resources.

By establishing these detailed policies, we can ensure that all user groups have fair access to compute resources, enabling productive and balanced use of the infrastructure.

12. Support

For any issues or support, please contact the system administrator

Ratna Kumar N Bollapragada

Email: ratnakumar@ncra.tifr.res.in

Phone: 020-25719251

Revision #45
Created 15 July 2024 10:21:28 by Ratna Kumar Bollapragada
Updated 17 December 2024 06:42:16 by Ratna Kumar Bollapragada