




Master in High Performance Computing

# Modernizing a High Performance Cluster - A Case for ICMS at Temple University

*Supervisor(s)*:
Fernando Posada, PhD Supervisor,
Axel Kohlmeyer, PhD Supervisor

*Candidate*:
Herbert Kudakwashe Nguruwe

5th edition
2018–2019

**Abstract**

The first part of the paper will look into upgrading the operating system of a cluster from CentOS 7 to CentOS 8 in a bid to test the stability and production readiness of the CentOS 8.0. The second part of the paper focuses on transiting from Torque and Maui to Slurm (batch system) porting the same configuration to achieve the same or improved scheduling from Slurm with shorter waiting time for jobs as well as maintaining the same load or obtaining higher capacity on the cluster.

# Acknowledgement

I would want to thank the International Centre for Theoretical Physics(ICTP) for giving me the opportunity to study for my Master in High-Performance computing.

My highest appreciation goes to Ivan Girotto for extra-ordinary support, Dr Fernando Posada (my supervisor) for supporting me with my Project. Not forgetting the HPC team from ICMS (Temple) for their tremendous help during the period.

# Contents

# List of Figures

# Introduction

Most scientific institutions require high computational power to process complex algorithms with high accuracy. This need for complex calculation with high precision has made a large number of them to host their own local High-Performance Computing centre (HPC).

The Institute of Computational Molecular Science (ICMS) at Temple University is not immune to this problem. They opted to have their local HPC facility, which is funded by multiple sources including the National Science Foundation (NSF). This HPC facility is powered by Linux clusters housed in the Temple's Joint Data Center (JDC) which is also home to other University IT services. ICMS has its own dedicated HPC team that manages HPC cluster, this includes but not limited to build, maintain software (install, update or delete), maintain (upgrade, replace or remove hardware) as well as managing permissions to users. In addition to the above tasks, they are also responsible for engaging Temple faculty researchers by providing training and support where needed[10].

ICMS's HPC has three hardware components which are compute, machine learning and Owls'Nest. Compute is a collection of multi-socket and memory sharing servers that handles interactive calculations, applications that do not scale well and long-running computations, that cannot be interrupted and restarted. In other words, it runs projects with requirements that are not permitted or not possible to run on the Owl's Nest (HPC cluster). The main work station in Compute has 64 CPU cores and 512GB of RAM. Compute includes a particular dedicated server for R-studio that enables users to run jobs interactively via the web browser.

The Machine learning component consists of two GPU servers for intensive GPU computing. One server has 4 GPUs (GPU compute), and the other has 8 GPUs (DGX-1), and both share interactive jobs. Shared interactive means that users are can use resources at the same time. GPU compute houses 16 CPU cores and 512GB of RAM and 4x NVIDIA Tesla V100 GPUs interconnected by NVlink2 and the DGX-1 provides 40 CPU cores and 512GB of RAM, and 8x NVIDIA Tesla Volta V100 GPUs with

NVlink2.

Owl's Nest is the latest and the most substantial addition which was assembled in 2017 and further upgraded in 2018. It features 180 new dual-socket compute nodes with 28 cores and 128GB of RAM each. Jobs that have extensive memory requirements will benefit from 6x 512GB, 6x 1.5TB and 2x 3TB RAM machines. Additional, each 512GB box hosts two NVIDIA P100 GPUs.

In 2018 the cluster was further extended with 48 more dual-socket compute nodes with 16 cores and 96GB of RAM each interconnected by an EDR InfiniBand (100Gb/s). In total, Owl's Nest currently hosts 6,464 CPU cores, providing about 57 million service units (CPU core hours) per year. This computation is backed by a 1.5PB parallel storage which hosts all user's data across the entire cluster. For large public (read-only) data sets, there is an additional 0.5PB storage.

The current Owl's Nest is running on CentOS Linux version 7 which is an open-source operating system derived from the sources of Red Hat Enterprise Linux (RHEL). It has grown popularity among HPC clusters because of its easy to manage, stable, systematic, easily reproducible, secure and well-documented platform moreover because of it's extensive usage its easy to get technical support or advice from the CentOS community[4].REF.

In an HPC environment jobs are mostly submitted in batches, and a batch system to put them in a queue. The scheduler will select the job to run based on their priority. Owl's Nest use torque as the batch system and Maui as the scheduler. Maui is no longer maintained and supported meaning that they will be no more updates and community of people using it is decreasing. The current configuration of Maui Owl's Nest has an average load of 80% to 85%. An essential aspect of the setup is the use of "job priority" policy which implements an algorithm that will try to achieve a fair chance to run their jobs as well as maintain a high throughput. They are many factors that impact job priority and most are presented in section 2.

Upgrading HPC technology (software and hardware) gives an added competitive advantage in delivering solutions in terms of reduced computational time, thus compute big and complex computations for the same time and high quality of solutions. This prompted the HPC team to decide that the next version of Owl's Nest, will be using CentOS 8 to take advantage of the improvements explained in detail in part 1. The first part of this paper investigates the production readiness of CentOS 8, looking into security, stability, adaptability and reliability of the software to run in the cluster.

The idea of upgrading the operating system prompted questions about the stability and sustainability of the HPC cluster how well will other software integrates with the new operating system. In this case, Maui and Torque are

no longer supported and making plans with unsupported software is a risk. Also, although they are working well with the current setup at ICMS, they present two problems:

1. Owl's Nest is a medium cluster, users that require more resources than what it provides must apply to the national HPC clusters. However, National clusters use Slurm for scheduling which is different from what most users from ICMS are used to (Torque and Maui). The change in the batch system might lead to users consulting the HPC team to change scripts from Maui to Slurm, increasing more work to the HPC team. Hence there is a need for a move to a more adapted environment on the local ICMS cluster.

2. HPC team develops any patches/updates on Torque and Maui since its no longer supported. Besides, this will mean that it will only be getting harder to get help since the community base is declining.Additionally we do not know what to expect if we move to the new operating system.

With the above challenges, the decision to move to a more stable, well maintained and supported batch system is unavoidable. The HPC team decided to investigate if the same setup/configuration that Maui uses could be imported to Slurm and maintain the same load to the cluster or even improve the capacity with better scheduling[10].

# Part I

# Operating System

# Chapter 1

# Cluster Installation (CentOS 8)

## 1.1 Introduction and Motivation

The operating system is the most important component in building an HPC cluster, as mentioned earlier Owl's Nest is running on a stable version of CentOS 7,6. CentOS is the preferred open-source Linux distribution because of its build from the source code of Red Hat Enterprise Linux (RHEL) in other words an enterprise service for free. The only difference, however, is that Redhat doesn't have dedicated commercial support, but because of the large community of users, it is easier to get help from other users. Additionally, updates are frequently released especially the most important security updates plus other reliable, extensively tested packages with a higher life cycle version support of 10 years[3].

To remain on top of the game while providing bleeding edge technology to users. ICMS HPC team would want to roll out the next HPC cycle of hardware on the latest CentOS 8. This section is going to test CentOS 8 whether it is mature enough to be used in production. The tests will seek to answer the following questions:

- Is it easy to install, complete ( most the mandatory packages included in the package manager) and portable (on different hardware)?
- Is the system easy to update packages without breaking the underlying platform?
- is it reliable, stable and secure?

## 1.2 CentOS 8 change overview

CentOS 8 provides significant improvements from its predecessors, and this section focuses on updates that were used directly in building the cluster.

- Distribution
  Packages are divided into parts, BaseOS and Appstream repositories which are both required for a basic RHEL installation.

  1. BaseOS - is set to provide the functionality to the operating system and forms the bases of all installations.
  2. AppStream - contents in this repository consists of additional user-space applications, run time languages and databases. The repository is Red Hat Package Manager available in RPM format and modules. Modules are collections of related packages that form a logical unit which built, tested and released together[3]. In other words Module, streams are different versions of the same software, and only one module can be installed[4].

- Kernel
  It is distributed with kernel version 4.18, provides support for to the following architectures:

  – AMD and Intel 64-bit architectures
  – 64-bit ARM architecture
  – IBM Power Systems, little Endian
  – IBM Z

- Software management
  Dandified Yum (DNF) is the next version of Yellowdog Updater (YUM) package manager software that installs, updates, and removes packages on RPM-based Linux distributions. DNF supports modules, and it also increases performance and well-designed stable API.

- Dynamic programming

  1. There is no Python by default; Python 3.6 is the default implementation and limited support for Python 2.7
  2. Nginx 1.14 is the default

- File systems and storage
  CentOS 8 no longer allows creation, mounting or installation of Btrfs.

XFS allows copy-on-write (COW) functionality which enables two or more files to share a common set of data blocks. This improvement boosts the file system by making it faster(no disk I/O utilization in creating shared copies), space-efficient (doesn't consume additional disk) and transparent. The `/etc/sysconfig/nfs` file has been moved to `/etc/nfs.conf`[4].

- Security
  CentOS 8 made security upgrades components of HPC cluster.The following are cluster related security improvements:
  - Insecure cipher suites and protocols are removed, for instance, SSLv3 and SSLv2.
  - Network Security Services (NSS) now use SQL format for a database by default.
  - OpenSSH has been upgraded to 7.8 with notable changes to set minimal accepted RSA key size to 1024 bits
  - libssh2 is not available, but libssh implements Secure Shell(SSH) protocol as a core cryptographic component.
  - Rsyslog does not support legacy format by default.
- Networking

  Network scripts are deprecated and not provided by default. In the basic installation, the `ifup` and `down` scripts call running NetworkManager using the nmcli tool. Also NetworkManager supports internal and dhclient, but by default, in RHEL 8, it uses the internal plugin. Also below are the notable changes:
  - nftables framework replaces iptables as default packet filtering framework
  - firewalld daemon uses nftables as default backend
- Virtualization
  - PCI Express-based machine type (Q35) is supported and configured automatically in virtual machines (VM), providing improved features and compatibility of virtual devices.
  - Cockpit can be used as a web console to create and manage VMs
  - QEMU emulator has the sandboxing feature that provides limitations to system QEMU calls can perform making VM's secure[4].

- compilers and development tools
  - gcc compiler version 8.2, newer standard versions of C++
  - variety of tools to generate, manipulate and debug code can be experiment with DWARF5 debugging format
  - glibc library, supports Unicode 11, new Linux system calls and additional security hardening with improved performance.
- Kickstart

  Some commands have been removed or deprecated and new others have added. Most commonly used deprecated commands are `auth, install, device, partition`. Two commands added are `authselect` module‘ [4].

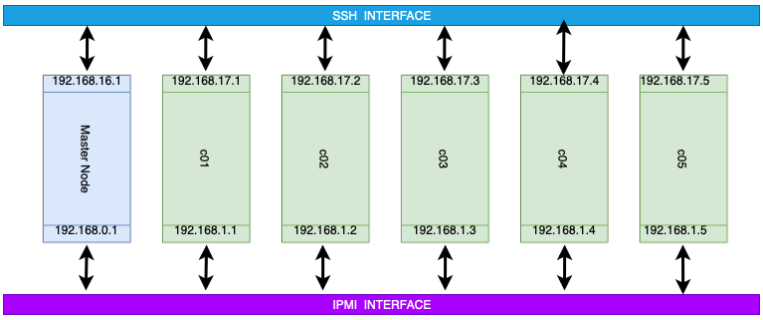


Figure 1.1: Training cluster architecture

Figure 1.1 Shows the layout of our training cluster. We have a training cluster with one master node and five compute nodes. The master node acts as the login node as well. We executed three different types of testing(installation), Manual installation, installation with Cobbler and Ansible playbooks.

## 1.3 Master node installation

Installation of CentOS 8 in the master node was not without hurdles as we had to deal with the curse of technology, latest software and in old hardware. CentOS 8 removed support for most of the old hardware drivers, including mptsas drivers required by the kernel to locate our hard drive impacting our installation. The kernel did not include the drivers for the local disk, making the kernel unable to locate the disk during installation.

The solution was to manually download the drivers from Red Hat servers and save them on a local flash drive/disk. During installation, we provided additional options to the grub that loads the local drivers to the kernel using the following command:

```
inst.dd=location
```

where the location is the path to the driver update, in this case, a flash drive path for instance `/dev/sdb`

The command loads the disk drivers, making it possible for the kernel to find the drive/disk to install the operating system during the installation. Upon completion, we updated the newly installed operating system using YUM or DNF package manager.

The system updates without issues and everything works perfectly until the system reboots. When we reboot the server, it crushes into the dracut shell because it will not be able to find the disk again since the latest kernel becomes the default during the boot time. To solve this problem, we reboot the server again and manually selecting the older kernel during booting. When the operating system has loaded, we download patched RPMs for dracut to our local drive and update dracut to the latest patched version. The patched version was not available in package manager YUM/DNF at the time of writing of this paper[3]. Red Hat Package Manager(RPM) is a package management system that makes it easier to distribute, manage and update software created for Red hat[4].

After dracut was updated, we created the drivers for the latest kernel using the following commands.

- Create drivers for the latest kernel

```
    dracut -f /boot/initramfs- 4.18.0-80.7.1.el8_0.
x86_64.img  4.18.0-80.7.1.el8_0.x86_64

```

- Check if the drivers are created

```
      lsinitrd -k 4.18.0-80.7.1.el8_0.x86_64 | grep
mpt3sas

```

A restart of the system successfully boots to the latest kernel.

## 1.4 Compute nodes installation

As seen in figure 1.1, our HPC architecture has one master node with five compute nodes that are connected. In this section will focus on installing CentOS 8 in compute nodes and setting up the environment for running the computation.

### 1.4.1 Configure Hardware and Network

ICMS HPC training cluster computers connect using two types of network communication protocols which are Secure shell (SSH) and Intelligent Platform Management Interface (IPMI). SSH is a secure interface and protocol that is used to connect to remote server [12]. IPMI is a platform that provides management and monitoring capabilities independently of the host system's CPU, firmware (BIOS or UEFI) and/or operating system. This protocol enables system administrators to achieve tasks such as switch on or off, get mac addresses of nodes, get power consumption etc.

In our setup, the master node will be the only node that has access to the outside world (global network) for security reasons. All compute nodes will get the required software either during installation/updates from a repository located in the master node. In addition to the security step communication between nodes is controlled by iptables, which is a firewall daemon that controls access to packets between computers.

#### DHCP Configuration

Compute nodes are connected to the network by providing media access control (MAC) addresses to DHCP server, which allocates them IP addresses. Media access control (MAC) addresses are a unique identifier on every network device on a network. To get MAC addresses of compute nodes, we use tcpdump. A tool that traces packets in a network and gets the MAC address of where the packet is going using the verbose option of tcpdump we can obtain MAC address of compute node using the following command:

```
1 tcpdump -i ens4 -vv port bootps
```

Table 1.1 contains the list of compute nodes' details relevant to the DHCP and cluster management.

DHCP assigns static IP addresses to compute nodes using the information in the table 1.1. The network requires two zones(IPMI and SSH) where the IPMI is for serial access of compute nodes and SSH used to access remote server to execute all the commands using the SSH daemon. According to

| ipmi ip | ipmi mac | name | hostname | mgmtclass |
|---|---|---|---|---|
| 192.168.0.1 | f0:4d:a2:06:90:c7 | c01 | c01 | compute |
| 192.168.0.2 | f0:4d:a2:08:53:27 | c02 | c02 | compute |
| 192.168.0.3 | f0:4d:a2:06:80:77 | c03 | c03 | compute |
| 192.168.0.4 | f0:4d:a2:06:96:79 | c04 | c04 | compute |
| 192.168.0.5 | f0:4d:a2:08:50:f9 | c05 | c05 | compute |

Table 1.1: Compute node details for DHCP

our architecture 1.1, the DHCP is in the master node, and below shows, a snapshot configuration file for compute node c01.

### IPMI Configuration

The snapshot below creates one IPMI zone and adds compute node c01 with IP address `192.168.1.1/20` to this subnet and the routing server is the master node (`192.168.0.1/20`).

```
allow booting;
allow bootp;
 subnet 192.168.0.0  netmask 255.255.240.0 {
  option routers 192.168.0.1;
  option domain-name "ipmi";
  option domain-name-servers 192.168.0.1;
  option subnet-mask 255.255.240.0;
  default-lease-time 600;
  max-lease-time 7200;
 }

host c01-ipmi{
 hardware ethernet f0:4d:a2:06:90:c7;
 fixed-address 192.168.1.1;
 option subnet-mask 255.255.240.0;
}
```

### SSH Configuration

The configuration below creates a different subnet for compute nodes to connect remotely using SSH and also the setup of one compute node c01 assigned fixed IP address of 192.168.17.1/20.

```
 subnet 192.168.16.0  netmask 255.255.240.0 {
  option routers 192.168.16.1;
  option domain-name "hpc";
```

```
   option domain-name-servers 192.168.16.1;
   option subnet-mask 255.255.240.0;
   default-lease-time 600;
   max-lease-time 7200;
  }

 host c01-hpc{
  hardware Ethernet 00:21:9b:9f:7c:e1;
  fixed-address 192.168.17.1;
  option subnet-mask 255.255.240.0;
  option host-name "c01";
 }
```

### DNS Configuration

The Domain Name System (DNS) is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network [1].In the above DHCP configuration, we use IP addresses to locate and access compute nodes. DNS is used to locate nodes using human-readable hostnames instead of IP addresses either by forward lookup or Reverse lookup. A forward lookup is when the DNS is used to look for an IP address using the name of the compute node in a network configuration, for example in our case `c01` will resolve `192.168.17.1`.Reverse Lookup is when DNS use an IP address of a compute node, and the DNS can find the human-readable hostname in our case from `192.168.17.1` will to resolve to `c01`.

DNS server called bind server is installed and enabled in the master node and configured to allow nodes in our network to access it. Below is our SSH configurations in the DNS server for the zone `hpc`. This configuration means that if you SSH to master using `ssh master` it will resolve to the master node with the IP address of 192.168.16.1, which is the master node. The same applies to compute nodes; bash command `ssh c01` will resolve to compute node one.

```
$TTL 300
@                       IN      SOA     master.hpc. master.
   hpc. (
                                        2018102904   ; Serial
                                        600         ; Refresh
                                        1800         ; Retry
                                        604800       ; Expire
                                        300          ; TTL
                                        )

```

```
                            IN      NS      master.hpc.
master                      IN      A       192.168.16.1

c01                         IN      A       192.168.17.1
c02                         IN      A       192.168.17.2
c03                         IN      A       192.168.17.3
c04                         IN      A       192.168.17.4
```

The below configuration is a reverse lookup which allows us to use the IP address to resolve to the human-readable names. In the case of the master node, we simply use `hostname 192.168.16.1`. So with both configurations, we can use either use IP address or hostname to connect compute nodes with each other or with the master node.

```
$TTL 300
@                           IN      SOA     master.hpc. master.hpc. (
                                              2018102904   ; Serial
                                              600          ; Refresh
                                              1800         ; Retry
                                              604800       ; Expire
                                              300          ; TTL
                                              )

                            IN      NS      master.hpc.
master                      IN      A       192.168.16.1

1     IN  PTR  c01.hpc.;
2     IN  PTR  c02.hpc.;
3     IN  PTR  c03.hpc.;
4     IN  PTR  c04.hpc.;
```

### 1.4.2 Network booting

When the master and compute nodes are connected to the network, then network booting is possible. To enable network booting, we add the "next server" setting in the DHCP server configuration to point to master node which hosts the boot/iso image to be used for installation for the compute nodes. The master node must have TFTP, HTTPD servers to host kernel and initial ramdisks that will be used to boot from, and it must have the PXELinux which is part of the `syslinux` package. Httpd is an Apache server implemented as a standalone daemon that uses HyperText Transfer Protocol (HTTP), but because its a daemon it becomes HTTPD. A bootable ISO image is saved in the HTTPD server. Trivial File Transfer Protocol (TFTP) is a file transfer protocol used to transfer files from remote servers or to send

one. Network booting configuration enables different types of installation media such as serial console, SSH, TMUX, VNC, and Kickstart. Serial Console installation is when you install using the command line terminal using the IPMI network. SSH installation is when we install using the SSH channel to the compute nodes. TMUX is when we install using the TMUX console for installations. Kickstart is an installation using a kickstart file from the server; a kickstart file is a file that defines the pre-booting configurations.

### 1.4.3 Netboot environment with Cobbler

After successfully doing all manual installations, we added a semi-automated setup using Cobbler. The master node in our architecture is also the provisioning server, so all the necessary software to install in compute nodes are hosted in the master node.

#### Cobbler installation

Cobbler is a software that speeds up network installation by provisioning, managing services such as the DNS and DHCP, package updates, power management, configuration management orchestration. At the time of writing of this paper, Cobbler was not available in the YUM or DNF Package for CentOS 8, so it was installed using source files. Cobbler version 3.0 which is supposed to work using Python 3 had issues that we were forced to downgrade to use Cobbler 2.8 which uses Python2. We used the following procedure to install Cobbler:

- Install Python 2 and Python 3
- configure that Python 2 be linked to `/usr/bin/python` using the alternative command
- Test which Python version installed
- download source from git and checkout to `release28` branch
- Make install
- Enable and start Cobbler

Below is a full list of commands:

```
1   yum install Python 2  pythton3
2   alternative --config Python # select Python2
3   Python --version
```

```
4     git clone https://github.com/cobbler/cobbler.git
5     git checkout release28 # the stable version but uses
      Python 2
6     make install
```

We must set Cobbler to manage DHCP and DNS servers using templates. Rendering these templates will produce the same configurations as the above section. Cobbler imports an iso image and creates a profile by default. We add systems (compute nodes) to Cobbler database so that it will have a list of systems (compute nodes) to install. When adding compute nodes, additional options are required, such as both (IPMI and SSH) interfaces and kopts to systems. kopts are kernel options such as a location for other drivers or the latest drivers.

At the time of the writing of this paper, the latest Redhat kernel (version 4.8) does not support the old hard drives (mptsas) and required the patched dracut software to be installed to enable the updated kernel to create the drivers for updated kernel[4]. In our kick start file, we need to add kopts and additional repository that contains latest disk drivers during the installation.

#### RPM with Cobbler

As mentioned above, CentOS 8 contains BaseOS and Appstream, which Cobbler imports and create into repositories by default when the image is imported. Dracut software from the iso package at the time of testing software had a bug that it could not create init-ramfs files for the latest kernel. To solve this problem, we add the repository to the Cobbler that contains the latest dracut software so that during the boot process Cobbler will install the latest dracut version. Additionally, to the dracut repository, compute nodes needs to install minimal software for them to execute submitted jobs. We create an additional repository called the minimum repository that will enable all applications required by all to compute nodes is available after installation.

### 1.4.4 Cluster Environment

User management is one of the important aspects to implement; this is to ensure that we protect user jobs accessible to unauthorised users. For ease of use, users must be able to SSH to compute nodes that have their jobs without using passwords. This is achieved by having two SSH keys that synchronize to all the compute nodes. From a system admin point of view when you reinstall a node, it will be easier to save the ssh keys of all compute nodes

in the management node. So any new installation will be given an already existing key to avoiding known hosts error.

### Chrony server

There is a need to synchronise time between compute nodes and the master node. In the CentOS 7 NTP server was used to synchronise time between the master node and compute nodes, but CentOS 8 uses the chrony server as the default. The NTP and chrony server can be host and clients to each other; this means connecting to the existing chrony, or NTP server is possible. Master node with chrony will synchronize to an already existing external NTP server.

### Software Modules

A cluster environment requires different versions of the same software; for instance, some packages would require Python three and others require Python 2. Lmod is a Lua based module (LMOD) system is responsible for managing the same software with different versions by dynamically modifying the user's environment under Unix systems [5]. We installed LMOD in a shared file system such that the environment is available on all compute nodes, and installation is done using a standard user while additional post installations like link creation require root user permissions.

Installing software versions:

In our case, our shared path is `/shared/opt` after downloading and extracting archive we need to configure all Python software as follows

```
../Python-2.7.16/configure --prefix=/shared/opt/tools/Python-2.7.16
```

Upon completion of the configure command, we execute the make command and export the relevant environment variables:

```
# PATH - used to show where the executable are located.
export PATH=/shared/opt/tools/Python-2.7.16/bin:\$PATH
#LD_LIBRARY_PATH - to find dynamic linker
export LD_LIBRARY_PATH=/shared/opt/tools/Python-2.7.16/lib:\
    $LD_LIBRARY_PATH
#MANPATH - path for package documentation
export MANPATH =/shared/opt/tools/Python-2.7.16/share/man:\
    $MANPATH
#CPATH - global include for compilers such as gcc
export CPATH=/shared/opt/tools/Python-2.7.2/include/Python2
    .7:\$CPATH
```

Lua files are added in the modules directory which contains software specification, and in other cases, we can set the default version for the software. We repeat the same procedure for other software such as gcc, mpi and clang.

### 1.4.5 Torque and Maui

Torque is a batch system that is used in clusters to manage jobs as they are submitted, and Maui is responsible for the scheduling of jobs that are in a batch system. The two software is no longer supported and but at ICMS custom build their version by modifying and adding additional features. Installation was done from source code (Torque version 6.1.3) to create a software distribution and make a tarball, and with the distribution package, we create RPMs using rpmbuild. Pam security, cgroups and syslog must be added as options to the rpmbuild.

```
rpmbuild --with pam --without spool --with
    cgroups --with syslog -tb torque-6.1.3.tar.gz
```

RPMs are created and copied to a batch system folder to create a local repository that is later added to a Cobbler. We install the torque server in the master node and torque client in the compute nodes. The torque server will install trquathd which controls access to the server from clients and the pbs_server. A further detailed explanation can be found in the next section 2.2.

## 1.5 Ansible playbooks

All sections above were installed manually or semi-automated for both master node and compute node. This process works for small cluster but can be tiresome for large cluster deployment and maintenance. The HPC team at ICMS uses Ansible to automate, configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates [2]. Ansible has playbooks that combine all logical tasks into one group, and for you to run a playbook, you require an inventory. Inventory file consists of target machines and also roles that execute on target machines in the order which they appear.

We group roles using tags which a logical classification, for example, "auth" would combine all roles that provide authentication to the target hosts. For our training scenario, we only have types of inventory the master node and compute node.

Below is a list and brief description of the roles:

- auth - responsible for Authentication type of roles
- Cobblerd - install, configure and add systems to Cobbler
- Cobbler_bootstrap - responsible for setting the installation environment on compute nodes.
- update - is used to update the software packages
- storage - responsible for setting up and configuration to storage devices
- user software - responsible for ensuring that using software is available
- batch system - responsible for the batch system
- IPMI - responsible for setting up IPMI network
- mail - responsible for sending emails to users in case an error happened during job execution
- Infiniband - used for setting up the Infiniband connection
- network - used to setup networking within compute nodes.
- repos - manage repositories suing Cobbler
- ntpd_server- Install and manage munge server
- Slurm_server - install and manage Slurm server

A couple of changes had to be made to the old Ansible scripts to accommodate patches that were encountered during the manual installation. Playbook changes:

- VAR - global variable
  - change `system_ip` in the`master.yml`
- Cobblerd :-Cobbler is not available on yum, so a manual installation had to be done.
  - TASKS
    - download Cobbler task checkout to release28 which requires Python 2

    * install Cobbler dependencies and add a path to Python 2 in the Cobbler script or create a virtual environment but in our case, we use alternatives, so we use system Python 2
    * `make install` to install Cobbler
    * install mod_wsgi for Python2 task is used to configure mod_wsgi for Python 2
    * revert to Python 3 after installation
  - FILES
    * edit the zones files and add only 2 zones 192.168.0, 192.168.16
    * delete files 172.16.0, 172.16.1, 172.16.2, 192.168.1, 192.168.2
    * delete DHCP and named management template
    * rename master templates to DHCP.template and named.template
    * rename settings master to settings
    * replace the `add_root_pubkey, id_rsa, id_rsa.pub` with the correct public key
    * dhcp.template
      · edit the subnets from "255.255.255.255 to 255.255.240.0"
      · edit the names of the subnets to "ipmi and hpc"
      · add NetworkManager to packages to enable network after booting
      · install openssh - to allow SSH authentication after completing
      · edit disk partition in kickstart in order to use the whole disk.
    * named.teplate
      · edit the acl servicenets to 192.168.0.0/20, 192.168.16.0/20
      · settings file
        1. edit the zones
        2. edit reverse lookup
        3. edit dhcp and named servers

- repos- repository
  - TASKS
    * execute script that downloads the dracut RPMs
    * add dracut repo to a Cobbler and execute `a repo` sync to
  - FILES

        * add `download_dracut.sh` repo rpms script

- Cobbler_bootstrap
    - TASKS
        * Change the task names to CentOS 8 from CentOS 7
        * Change the download repo from 7 to 8
        * change profile names to reference CentOS 8
- create_systems
    - DEFAULTS
        * change all ip subnets from "255.255.255.0" to "255.255.240.0"
        * rename default profile to CentOS8-1905-minimal
    - TASKS
        * configure system SSH change the regex expression to from 1 to 17 for SSH network
        * add the kopts so that we can locate an additional driver
    - VARS
        * edit the database to use with the cluster 5 compute nodes
- ntpd_server
    - TASKS
        * install the latest chrony
        * configure master to use management node synchronise to Temple(university) ntp server
        * copy file to file_path = `/etc/chrony.conf`
        * allow compute nodes in the subnet to accees the master by inserting allow 192.168.16.0/20 and restart chronyd daemon.
- munge_server
    - TASKS
        * execute the script that creates munge user
        * copy munge file to compute nodes and ensure munge owns the file.
        * enable, restart and test munge
    - FILES

        * add create_munge_and_slurm_users.sh
        * copy munge.key to compute nodes

- slurm_server
    - TASKS
        * download and install Slurm dependencies
        * install slurmctld, slurmdbd and Slurm
        * enable, configure, restart both services

Cobbler changes that affect compute nodes:

- Ntpd_client (Chrony has replaced NTPD cleint)
    - TASKS
        * install latest Chrony
        * copy file_path = /etc/chrony.conf
        * configure compute nodes to use the already existing master node as ntpd server.
        * restart chrony demaon

- munge_client
    - TASKS
        * install munge dependencies
        * copy the munge key to all compute nodes and ensure that it has the correct permissions
        * enable, restart and test munge
    - FILES
        * copy munge.key to compute nodes

- munge_rpm_build
    - TASKS
        * download munge from source code
        * install unzipping software
        * create RPMs using rpmbuild
        * save RPMs to provisioning folder and sync them to Cobbler as repo
        * associate batch to profile

- slurm_client
  - TASKS
    - install Slurm dependencies and Slurm and slurmd
    - copy `slurm.conf`
    - create /var/spool/slurm/d directory
    - give ownership to Slurm user and Slurm group
    - enable and restart Slurm
  - FILES
    - add slurm.conf file
- slurm_rpm_build
  - TASKS
    - download tar from source code
    - copy RPMs to provisioning batch directory (batch system directory)
    - create a repo and execute `cobbler sync`

## 1.6 Conclusion

CentOS 8 offers quite a significant improvement from its predecessor CentOS 7. Still, I do not think that at the time of writing this paper, it is stable enough to be implemented for production. During the installation, we had to patch dracut for us to be able to use the drive. After updating it, the latest kernel would not boot because it could not find the drive. The kernel update can be a significant drawback if deployed in production as we have we might have to patch all the nodes. In case the issue is resolved by Red hat, and we update the system our nodes, we do not know how this will impact the already existing system.

At the time of the experiment, most applications were not compatible yet to work well with CentOS 8. For example, Cobbler had issues in importing CentOS 8 image and was also not ported in YUM. YUM did not have as many packages as we anticipated during the time of installation. Such that we had to install some of the essential applications from scratch.

# Part II

# Batch System and Scheduler

# Chapter 2

# Transit from Maui to Slurm

## 2.1 Motivation

As mentioned in the earlier chapter, Torque and Maui are no longer supported, and its popularity is declining with time. This cause a lot of inconveniences since they are no updates or bug fixes implemented to integrate with the latest technology. In the case of ICMS, the HPC team implements any changes and fixes the bugs, which might mean more work for them.

Owl's Nest is a medium-sized cluster offering a maximum of 12 nodes with 28 processes each for a period of 48hours (wall time) per job, but in some cases, users require more HPC resources than those provided by it[10]. In this case, they will apply to a national super-computing centres which most likely have adopted slum cluster. Using the Maui cluster means that they have to create a script that runs on Slurm. Frequently users request for assistance scripting assistance from the HPC team because they are not using Slurm. These requests increase the workload for the HPC team.

These reasons have led to the decision of moving from Torque and Maui to Slurm. The current configuration in Maui was gathered by scavenging different sources such as mail lists and blogs added with years of experience. The settings have been serving the cluster to the satisfactory of HPC team hence the need to transfer configuration to Slurm.

## 2.2 Maui

Terascale Open-source Resource and Queue Manager(TORQUE) is a resources manager that is responsible for controlling batch jobs, compute nodes and additionally comes with its scheduler. Maui is a scheduler that is responsible for scheduling jobs that are submitted in TORQUE. Many HPC

administrators prefer Maui and Torque combination in job handling rather than using Torque's default scheduler.

## 2.3 Slurm

Slurm is an open-source resource manager used in HPC environments that is highly configurable in heterogeneous resources as well as a job scheduler. It is a fault-tolerant and operates in a multi-thread and multi-daemon environment. Slurm controller daemon (slurmctld) is the main daemon that is responsible for managing resources and allocating workload on compute nodes. At the same time, slurmd is a communication node that runs on compute nodes.It is responsible for starting and finishing user's jobs on nodes and execution of prolog and epilog. Figure 2.1 illustrates the main components of Slurm.

Configuring Slurm and testing how the changes impact slum's job scheduling performance is not easy because it is usually done on the production system with actual job submissions to measure the impact of the given parameters. Many of the configuration changes of Slurm affect its performance in different ways, and various combinations might have either positive or negative implications, therefore, performing a parametric optimisation on Slurm on a live production system might have undesirable consequences. Therefore testing the impact of these changes in a simulation mode is the most preferred option[9].

### Scheduling Algorithms

Slurm provides a few scheduling algorithms, but for the scope of this paper, we are going to look at only two algorithms. In this paper, we will look at the FIFO and Backfill Algorithm.

#### FIFO

Jobs submitted in a batch system can be executed using first come first save basis,(FIFO way). In figure 2.2, FIFO scheduler executes jobs as they come, but when one job gets stuck, that means the jobs that are behind that stuck job will not run. Another problem with FIFO is underutilisation where if there are spaces in the cluster that can fit jobs that are in the queue, they cannot be executed. In other words, FIFO works with time-based fairness, In Slurm configuration file the option is set by using "sched/builtin" in file implements FIFO.

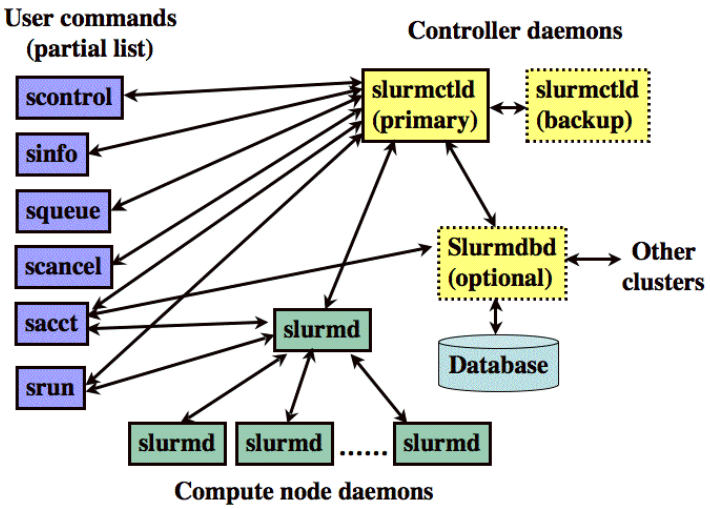


Figure 2.1: Slurm components [9]

### Backfill Algorithm

To improve throughput, Slurm has other scheduling algorithms implemented, but in this paper, we are only going to look at the backfill algorithm. Backfill algorithm is when Slurm performs smooth and periodic scheduling which happens, after a job has been submitted, a job has completed in the allocated nodes and also if the configuration changes. In figure 2.2, backfill algorithm nicely packs and fits all the jobs to fit in the available space. This scheduling strategy increases throughput and makes small jobs run faster than in the FIFO. To activate this in the configuration file, we use "sched/backfill" option. Backfill algorithm uses job priorities to schedule jobs.

### Slurm job prioritization

A job priority is a number/factor that is assigned to a job by the scheduler and used to determine when the job will run in the queue. Slurm has a multi-factor plugin that is responsible for calculating jobs priority. Factors that affect job priorities are listed below:

- Age - the amount of time a job is waiting in a queue
- Association - factor influenced by your association

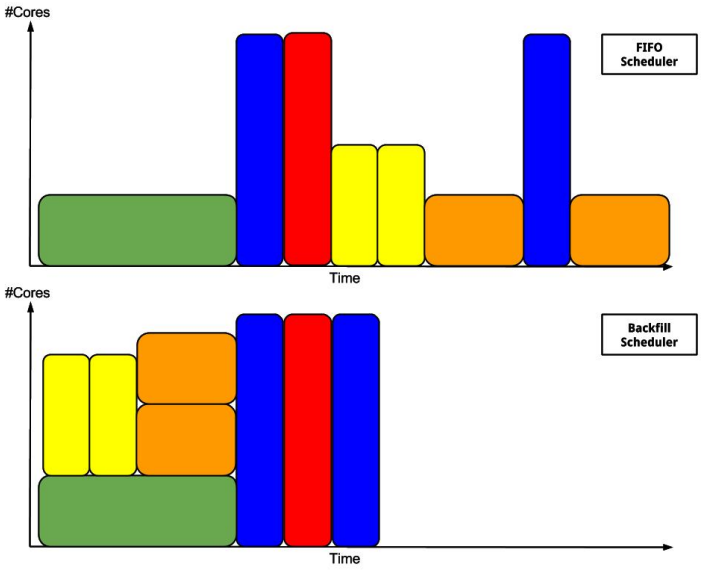


Figure 2.2: FIFO and Blackfill [6]

- Fair-Share - A factor that uses resource-based fairness, it will be a user's historical usage versus the available resources.
- Job size - allocated nodes to the job
- Nice - contribution that can be adjusted by a user
- Partition - contribution linked to the partition in which the job is running
- QOS - factor contributed by the quality of service
- TRES - Each tres, e.g. CPU, memory requested has its contribution factor

$$
\begin{aligned}
Job\_priority = site\_factor &+ (PriorityWeightAge) * (age\_factor) \\
&+ (PriorityWeightAssoc) * (assoc\_factor) \\
&+ (PriorityWeightFairshare) * (fair\text{-}share\_factor) \\
&+ (PriorityWeightJobSize) * (job\_size\_factor) \\
&+ (PriorityWeightPartition) * (partition\_factor) \\
&+ (PriorityWeightQOS) * (QOS\_factor) \\
&+ SUM(TRES\_weight\_cpu * TRES\_factor\_cpu, \\
&TRES\_weight\_ < type > * TRES\_factor\_ < type >, ...) \quad (2.1)
\end{aligned}
$$

The equation 2.1 is used to calculate job priorities that are later used by the backfill algorithm to schedule jobs. When choosing priority weights, we need to use high values as the factors are a number between 0 and 1. So to have a significant impact, we use the high-value names start from 1000[9].

### 2.3.1 Slurm simulator

As mention in 2.3 ,that we do not know the effects of changing parameters on a production system gave birth to slum simulator.If the effects are in production are adverse, we might encounter an inefficient job scheduling , that jobs might have extended waiting periods or inability to allocate resources and in worst cases a possible downtime.A Slurm simulator is a software that uses advanced algorithms to mimic real live Slurm to schedule jobs.Depending on the number of jobs and queues the Slurm simulator, can schedule a year's jobs in one hour (REF).As such, it used to identify areas of priority boost that will influence the scheduler to align according to the organization's policies. Additionally, parameter optimization between the main scheduler and backfill scheduler can be investigated in detail. In another context,the simulator is used to develop Slurm scheduling components[7].

The Slurm simulator altered several components from the real Slurm simulator. They replaced two primary methods; time() and gettimeofday() to point to the current simulation time and they return a shifted time value. For instance, jobs submitted in 2017 will be simulated, just like in 2017. Slurm has a multi-threaded design which makes it fault tolerant, but this makes it slow as it needs to synchronise. On the other hand, the simulator executes mostly in a serial mode and avoids thread lock hence increasing scheduling efficiency. Performance can also be boosted by compiling optimisation flags and disable functions such as assertions. The real slurm scheduler a single

backfill might take minutes to schedule jobs while in simulation, it can lead to jobs starting faster[8].

## 2.4 Current HPC analysis at ICMS

For us to have a better understanding of the cluster we have to analyse the job submission trend using Log files from the Owl's Nest production log files. The log files were from October 2017 to October 2019, but to narrow the scope we only concentrated from January to June 2018. The log files from Maui had many attributes that we striped some information to leave only the ones that are relevant to the project. Additional attributes had to be created, such as the waiting time and execution time. Below is a list of attributes that were relevant to the project and their respective meaning.

- JobID - The unique Id of the Job
- UserName - Unique username to a user
- GroupName - Class group the user belongs to
- WCLimit (Wall clock limit) - The maximum time the user is requesting on the cluster
- JobState - One of Completed, Removed, Not Run
- class - Queue type normal/large
- Queuetime - Epoch time when a job was submitted
- DispatheT - Epoch time when scheduler requested job begin executing
- StartTime - Epoch time when a job began executing
- CompleteT - Epoch time when job completed execution
- Byp - Number of time job was bypassed by lower priority jobs via backfill or '-1' if not specified
- Start Date - Epoch time indicating the earliest time job can start
- End Date - Epoch time indicating the latest time by which job must complete

### 2.4.1 Maui Configuration at ICMS

Our scope for this paper is to focus on the normal queue, which has 180 nodes, and each node has 28 cores. Owl's Nest's configuration favours big jobs over small jobs. This setup prevents small jobs from filling the whole cluster, and big jobs will not be able to run at all. So to balance off things we prioritise big jobs and small jobs are added by the scheduler using the backfill algorithm. Backfill algorithm calculates the estimated time high priority job expected to start and checks if there are available resources. If there are free resources and the queue has, job/jobs that can use the free resources without interfering with the start of the next high priority job, then this job/jobs will run. If this condition is not met then they will be no back filling at all [11].

#### Job prioritisation in Maui

Job prioritisation is the process of assigning jobs, a number that will determine the position in the queue relation to which job runs next. In Maui, this number is contributed by five factors which are listed below with their respective equations:

**CRED (job credentials)**

Equation 2.2 calculates the contribution made by the CRED component to the overall priority. This priority is based on political views and is based on who you are. which account you belong to and which groups do you belong.

$$\begin{aligned} Priority+ = CREDWEIGHT * (&userweight * J- > U- > Priority \\ &+ groupweight * J- > G- > Priority \\ &+ accountweight * J- > A- > Priority \\ &+ qosweight * J- > Q- > Priority \\ &+ classweight * J- > C- > Priority) \end{aligned} \quad (2.2)$$

In equation 2.2, J represents User, G will be a group that J belongs to, A is the account, Q is for quality of service and C is for the class associated.

**FS (fair-share usage)**

$$
\begin{aligned}
Priority+ = FSWEIGHT * MIN(fscap,\\
(fsuserweight * DeltaUserFSUsage\\
+ fsgroupweight * DeltaGroupFSUsage\\
+ fsaccountweight * DeltaAccountFSUsage\\
+ fsqosweight * DeltaQOSFSUsage\\
+ fsclassweight * DeltaClassFSUsage)) \quad (2.3)
\end{aligned}
$$

Equation 2.3 calculates the fair-share contribution to the overall priority. Fairshare component uses short term historical usage to favour jobs. It alters jobs priority based on a user's historical system utilisation billed against a user, group, account, or QoS. The duration in which a historical job usage affects job prioritisation is configured in a Slurm configuration file using the decay factor. Decay factor is the exponential decay in which historical usage will affect user's job prioritization.

**RES (requested job resources)**

Resources (Res) component allows a site to prefer certain jobs based on requested resources to meet either their mission objective, improve fairness and overall system utilization. In the case of ICMS they would love to favour big/large jobs over small jobs. The 2.4 refers to help.

$$
\begin{aligned}
Priority+ = RESWEIGHT * MIN(rescap,\\
(nodewieght * TotalNodesRequested\\
+ procweight * TotalProcessorsRequested\\
+ memweight * TotalMemoryRequested\\
+ swapweight * TotalSwapRequested\\
+ diskweight * TotalDiskRequested\\
+ peweight * TotalPERequested)) \quad (2.4)
\end{aligned}
$$

**SERV (current service levels)**

Service components (serv) selects which service metrics have greater value on the site. This means selecting the way Maui execute the service has a greater effect to service delivery. Below are subcomponent options that influence the way service delivery factor.

- Queue time: Selecting this component will favour first in first out (FIFO). This will favour time-based fairness only.
- Xfactor

$$xfactor \quad = \quad \frac{1+ < effqueuetime >}{\max(< xfminwclimit >, < wallclocklimit >)} \tag{2.5}$$

  This component favours short jobs (with less execution time), short running jobs see their xfactor grow faster than long running jobs as demonstrated in 2.5. The weight contribution is added using XFACTORWEIGHT.

- Bypass (BYPASS) Subcomponent: This component prevents backfill starvation by increasing bypass count whenever a job id passed by another job with lower priority.

**TARGET (target service levels)**

$$Priority+ = TARGWEIGHT* \\ (QueueTimeComponent + XFactorComponent) \tag{2.6}$$

The 2.6 is used to meet certain job scheduling targets exponentially. This is not implemented in the ICMS's cluster

**USAGE (consumed resources – active jobs only)**

$$Priority+ = USAGEWEIGHT* \\ (usageconsumedweight * ProcSecondsConsumed \\ + usageremainingweight * ProcSecRemaining \\ + usageexecutiontimeweight * SecondsSinceStart \\ + usagepercentweight * WalltimePercent) \tag{2.7}$$

The 2.7 applies to execute jobs already running only, and not implemented at ICMS.

### 2.4.2 ICMS log/ Maui log analysis

For an in depth understanding we investigate the log files and discover job submission trends in Owl's Nest. Our log analysis is going to focus on waiting time for jobs, job bypass factor and cluster capacity. In the end, we will use

these factors to measure the success of our parametric transfer between Maui and Slurm. Job waiting time is the time between when a job is submitted to when the job starts to run. According to Maui job trace files, waiting time is the difference between the submission time (QueueTime), and the job begins running (StartTime). Bypass factor, on the other hand, is the number of times lower priority jobs passed a job with high priority via the backfill algorithm. Cluster occupancy is how many cpu-time is being used against the total cpu-time available in the cluster.

To better understand the usage of the cluster, we decided to focus on a specific time range which is from 1 January 2018 to 30 June 2018. Our main area of interests from the log files occupancy and waiting time. Still, other metrics such as node request distribution and Bypass factor help gives us a clearer understanding of the cluster.

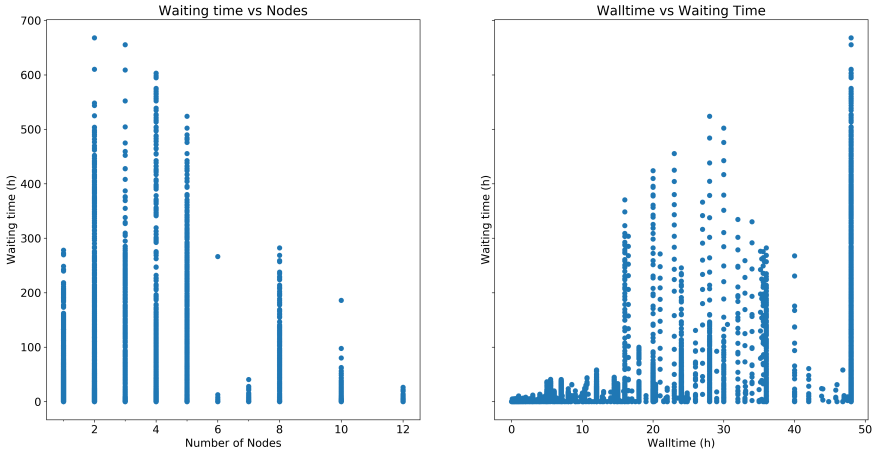


Figure 2.3: Wall time analysis

Owl's Nest's configuration favours jobs with high number of nodes over jobs with lower nodes. Also jobs with short wall time get started faster as compared to the ones with longer wall time. This setup is evident in the graph 2.3 that the higher wall time you request, the more waiting time you will endure. This is shown that jobs that have asked for a time closer to maximum wall time will have to wait longer than the ones that requested for a few hours. The concept is that the main scheduler will schedule the bigger jobs (more nodes) and backfill scheduler will fill in the small jobs (less number of nodes).

### Node Frequency

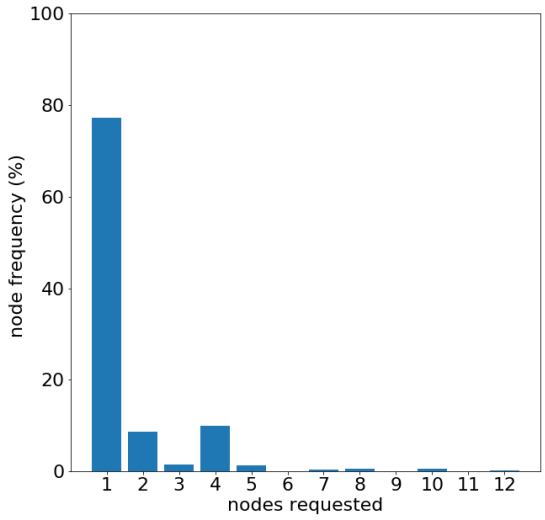


Figure 2.4: Node Frequency (pre-analysis)

The node frequency graph (Figure 2.4), shows statistical representation of nodes. Node frequency refers to the number of times a certain number of nodes are requested. One can also note that almost 80% of jobs in this queue are one node jobs. Jobs with twelve nodes are very few that their impact might not be that significant concerning their contribution to job priority. Two and four-node jobs are significantly high as compared to the other numbers, excluding one node jobs.

As evident in the wall time graphs 2.3, it can be noted that in the graph 2.5 the bypass factor is higher for smaller jobs in terms of nodes and smaller for big node jobs. The same principle applies when bypass factor is measured against wall time. The more wall time you request the higher the by pass factor.

### Cluster

The occupancy graph (Figure 2.6) shows how much load has the cluster been subjected to over some time. In this regard, Owl's nest maintains a very high occupancy, which is more than 80%. We created the graph by extracting intervals of fifteen days for six months. We calculated the occupancy in the following manner:

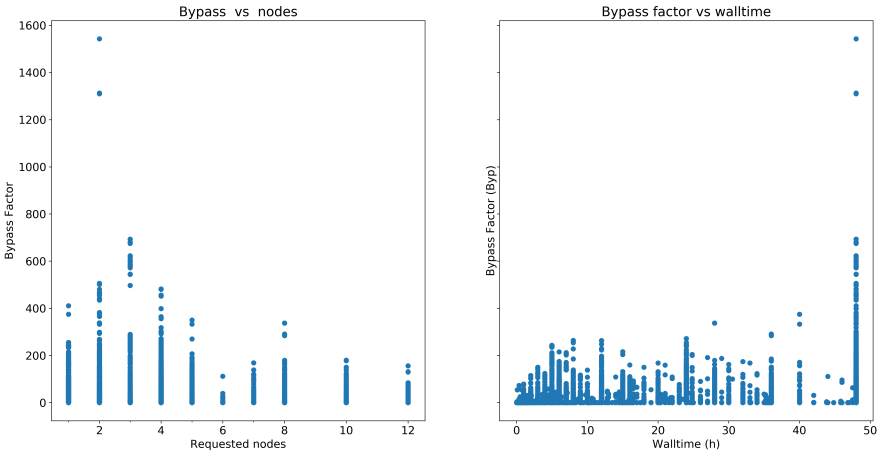


Figure 2.5: Bypass Factor

$$CPU\text{-}hour_{max} = 15day * 24hour * 180node * 28CPU \tag{2.8}$$

and the

$$CPU\text{-}hour_{used} = 28CPU * \left( \sum_{i}^{n} t_i * nodes_i \right) \tag{2.9}$$

for the $n$ jobs with starting date within the time interval chosen. Each job $i$, has a duration time $t_i$ and a number of nodes allocated $nodes_i$. Then, our occupancy for fifteen days will be:

$$Occupancy(\%) = \frac{CPU\text{-}hour_{used}}{CPU\text{-}hour_{max}} * 100 \tag{2.10}$$

## 2.5 Simulation in Slurm

Maui log files are the main ingredients of our slurm simulation. According to the graph 2.7 simulation process circle. We extract all users from the data and create a new `users.sim` file that has unique user names. The file `users.sim` is of the format: `USER : UID`, where UID is the Linux user identifier. This user information enables us to create a script that will add users to the Slurm database using the account manager sacctmgr, the code used to add a user is the following :

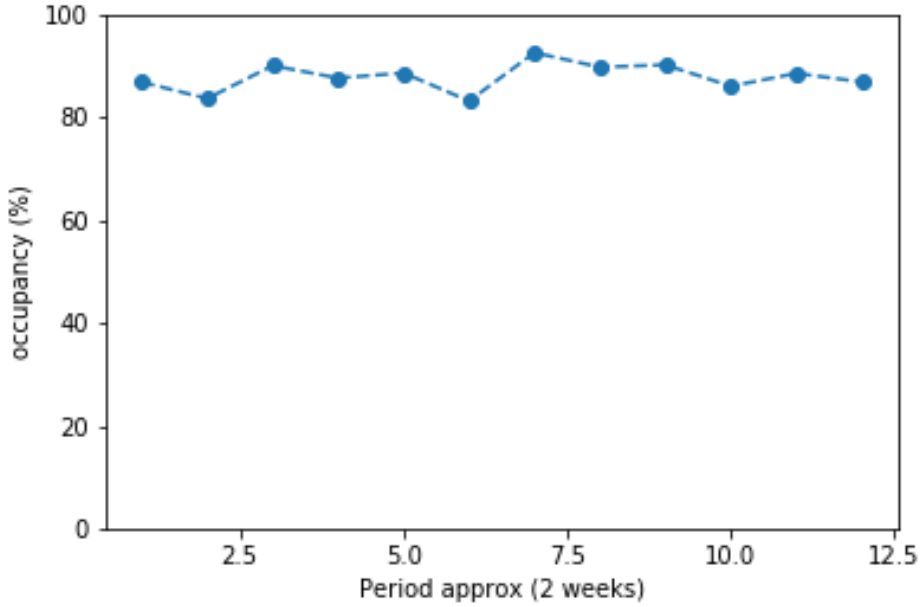


Figure 2.6: Utilization of normal queue for the period of time analysed.

```
sacctmgr -i add user name=user1 DefaultAccount=account1
  MaxSubmitJobs=1000
```

Using the user's MAXPS from Maui configuration file, we create another script that adds this limit to users. The equivalent of Maui's MAXPS is GrpTRESRunMin in Slurm. We modify the user using the following code:

```
sacctmgr -i modify user user1 GrpTRESRunMin=cpu=40000000
```

At the same time, we can install and configure slurm simulator. The main components in the Slurm are the slurmdbd and slurmctld. Once the infrastructure is working, we can execute the user scripts into the database.

After dealing with user scripts, we extract jobs from Maui and change to job trace file that will be used by Slurm simulator as a job trace. Then transfer settings from Maui to Slurm using our intelligent guess and our online community recommendation. After reconfiguring Slurm, we export the `SLURM_CONF` file to point to where the configuration is. We can now mimic our HPC cluster, Slurm writes log files that we later use to analyse how it was performing. When Slurm has completed simulating, we now perform statistical calculations and perform some data visualisation.

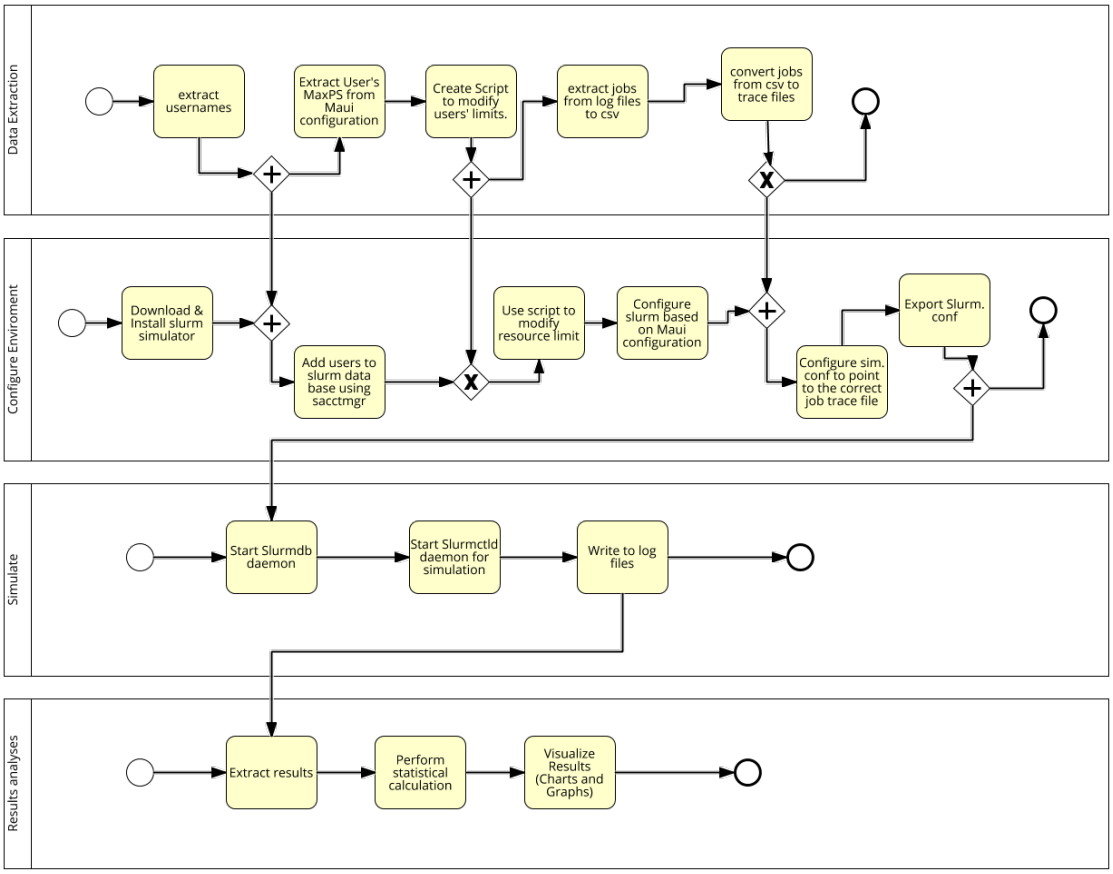


Figure 2.7: Simulation process

**Settings transfer**

The initial transfer of settings from Maui to Slurm was based on a educated guess and guidance from resources online articles. The table 2.1 shows the initial transfer form.

| Maui Parameter | Slurm |
|---|---|
| MAXPS | GrpTRES and GrpTRESRunMin |
| FSWEIGHT | PriorityWeightFairShare |
| SERVICE WEIGHT(Xfactor) | PriorityWeightJobSize |
| CRED WEIGHT | Association |
| RES WEIGHT | PriorityWeightTRES |

Table 2.1: Comparison between Maui and Slurm

Table 2.1, shows more or less the equivalence of parameters between Maui and Slurm. Maui's MAXPS corresponds to Slurm's GrpTRES and GrpTRESRunMin. These settings are using to enforce resources restrictions to users. FSWEIGHT is the fair share weight in Maui and PriorityWeightFairShare does the same thing, the difference is that in Slurm we do not have subcomponents.The component of SERVICE WEIGHT, which has a subcomponent of Xfactor implements more concept of favouring Big jobs over small jobs, is equivalent to PriorityWeightJobSize. CRED WEIGHT has to with social politics is matched to Association in slurm. RES WEIGHT in slurm is mapped to PriorityWeightTRES in Slurm

## 2.6 Results and Analysis

**First simulation (no job priority and no GrpTRESRunMin)**

Our first approach to testing the effect of altering parameters was to do a simulation with no restrictions applied. In this test, we did not use any job priority and restrictions (MAXPS/GrpTRESRunMin) to users and we used fewer jobs to test this effect than the actual submitted jobs. The idea is to mimic a FIFO algorithm were jobs are executed as they are submitted. Based on the mean value of requested nodes, we would expect less to no waiting time. This observation is evident in the graph 2.8. The maximum waiting time for a job is 4 hours because the only attribute stopping jobs is the maximum number of nodes 12 applied to the queue in Slurm configuration file.

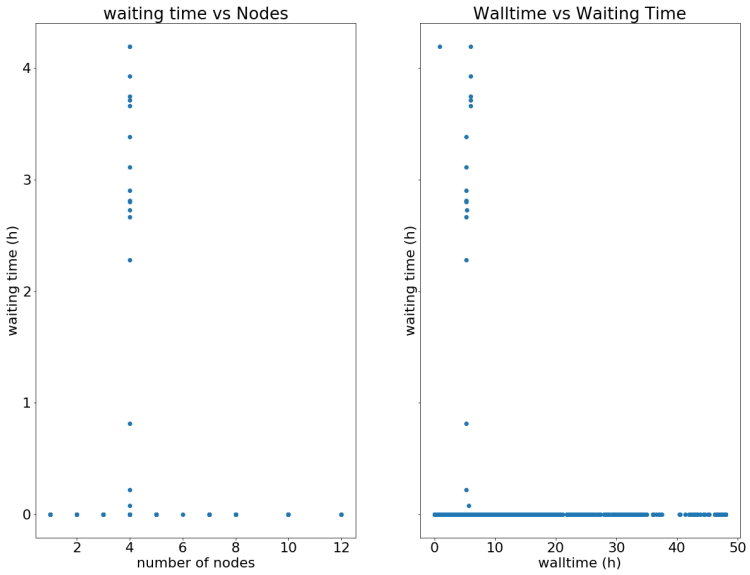


Figure 2.8: Wall time analysis:no GrpTRESRunMin

**Second simulation (with GrpTRESRunMin but no job priority)**

This exercise is to see the impact of MAXPS; in our first simulation, there were no restrictions that implemented to users except the one from the queue for 12 nodes and 48 hours wall time. In the same configuration, we added MAXPS to users, and there was an increase in waiting time from a maximum of 4 hours to 70 hours (see Figure 2.9). These results are expected because even though resources are available, you are not allowed to exceed an as certain threshold. This restriction will cause jobs to wait longer until the jobs of the same user have finished running.

### 2.6.1 Third simulation with fair share weight

In this simulation we are going to test the effect of fair-share to the scheduler. We are going to simulate all jobs submitted from 1 January 2018 to 30 June 2018. This configuration will have GrpTRESRunMin, queue restrictions and fair-share implemented.(TO BE SIMULATED)

The graph 2.10 shows a simulation of jobs for two months from the period of 1 January 2018 to 28 February. We added the fairshare and GrpTRES-RunMin. Based on the configuration we have the wall time is a bit higher for

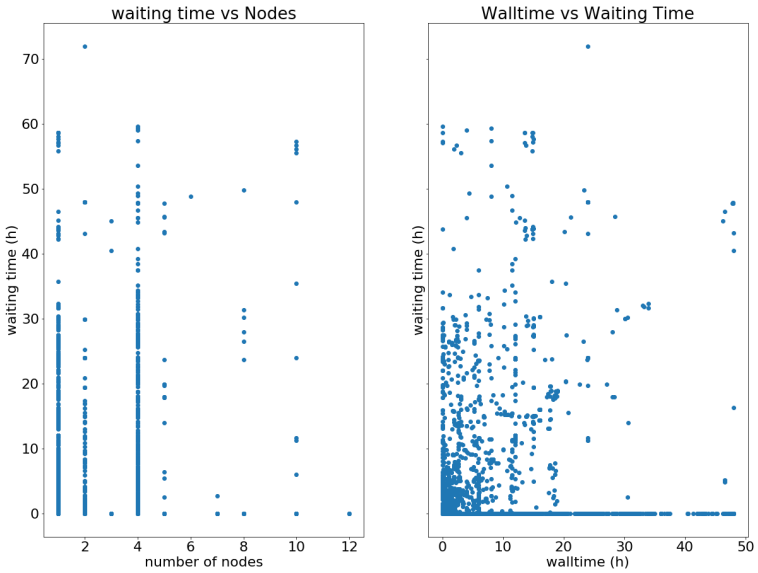


Figure 2.9: Wall time analysis: with GrpTRESRunMin but no job priority

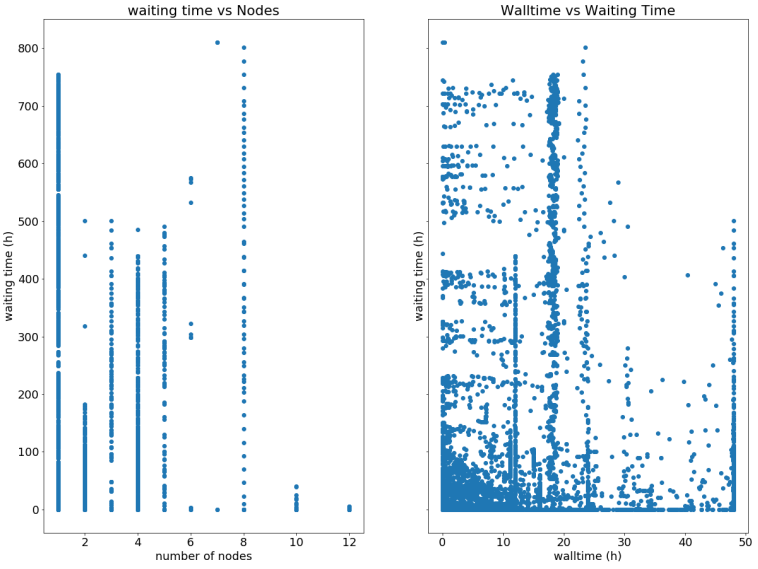


Figure 2.10: Wall time analysis: with fair share weight

jobs with lower wall time. For a shorter period like it might be not very easy to point the reason. Based on the simulation log files, GrpTRESRunMin would stop some users because they have submitted several jobs exceeding the limit.

For this simulation, node frequency 2.11 is as expected its around 75%. However, the occupancy is 20% lower than the expected or acceptable.

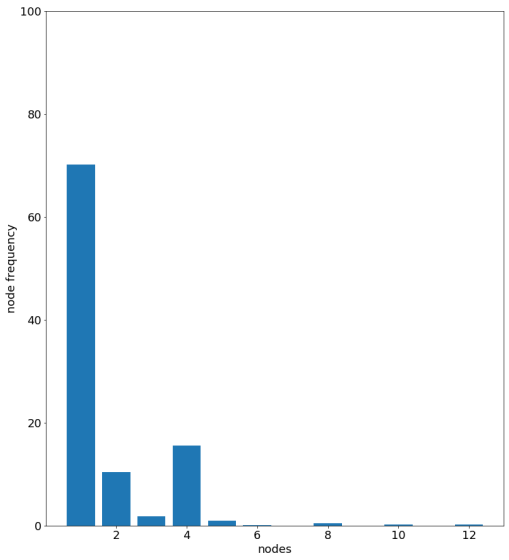


Figure 2.11: Node frequency: with fair share weight

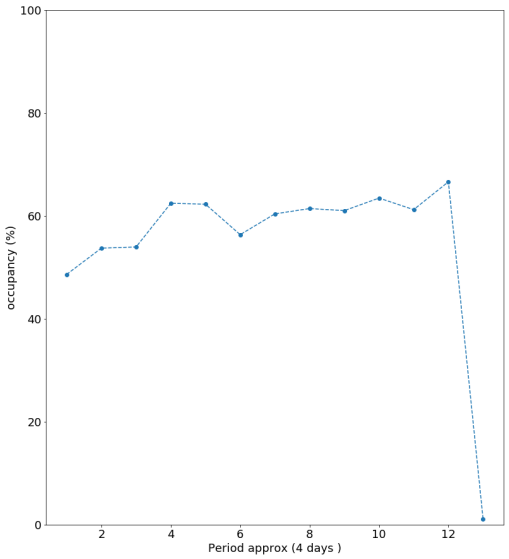


Figure 2.12: Occupancy: with fair share weight

We would have wanted to simulate this for at least a period of 6 months and fine-tune the results with better and more stable results. It is unfortunate that based on the time constraints, we have to use two months. It is also crucial to a note that the period in which to consider for occupancy should be when jobs are still submitted. This is because the queue might be only left with jobs that are being stopped because they do not meet specific criteria this will reduce the occupancy, but the simulator will be running. This is shown on the last point of the graph 2.12

### Conclusion

The project was to transfer configuration settings that are working in Maui to Slurm by testing the impact of individual parameter changes. However, with time constraint, we have managed to check user association rules, job size priority and fair share priority factors. So far, we have managed to transfer the trace jobs from users from Maui successfully. We have proven that the simulator works intended by completing, scheduling and executing tasks in exact times as in Maui.

# Chapter 3

# Conclusion and Future work

The first phase of the project was to test the maturity of CentOS 8. During the exercise, we had a couple of issues that required us to do a workaround or implement a patch on some critical components of the operating system. Some of the essential software that we use in the cluster were not in YUM/DNF, and we had to install it manually. CentOS 8 doesn't attribute to a reliable operating system. I am sure with further releases to come it will be more adaptable.

The second part aimed at porting Maui's configurations to slurm. We used Maui's jobs log files to create a job trace for slurm. However, this was not without hurdles; the real simulation happened in New York time zone, in our initial conversion, we added the time zone change to New York, which made everything meaningless. We corrected this error by switch all time zone to local time zone (GMT+1). In addition to that, we had regular expression in extracting jobs from log files that prevented more jobs from being selected.

Despite these challenges, we managed to implement the Slurm simulator and tweak a couple of parameters to observe how they impact the scheduler. We managed to prove the effects of MAXPS, FIFO using few jobs and Fairshare scheduling. To get to our ultimate goal of a trusted configuration file, we still need to experiment more with other effects of other parameters for us to tweak better and arrive at the results faster using a combination of settings.

Future works will be to continue tweaking and testing parameters in Slurm to get a better understanding of the parameters and on the normal queue. Then lastly we add all the queues just like in Owl's Nest and run all the trace jobs from Maui to get the same or better scheduling. We then experiment with other options to optimize scheduling in Slurm. The ultimate goal is to have a configuration file with high capacity and lower waiting time.

# Bibliography


[1] CloudFlare. *CloudFlare*. URL: https://www.cloudflare.com/learning/dns/what-is-dns/. (accessed: 24.12.2019).

[2] Red Hat. *Ansible Documentation*. URL: https://docs.ansible.com/ansible/latest/index.html. (accessed: 24.12.2019).

[3] CentOS Linux. *The CentOS Project*. URL: https://www.centos.org/about/. (accessed: 13.12.2019).

[4] Red Hat Enterprise Linux. *8.0 RELEASE NOTES*. URL: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/8.0_release_notes/index. (accessed: 13.12.2019).

[5] LMOD. *LMOD*. URL: https://sourceforge.net/projects/lmod/. (accessed: 24.12.2019).

[6] Salvador Martin and Jordi Blasco. *shcedueling slurm training Documentation*. URL: https://docplayer.net/50975263-Scheduling-slurm-training15-salvador-martin-jordi-blasco-hpcnow.html. (accessed: 10.02.2020).

[7] Nikolay Simakov et al. "A Slurm Simulator: Implementation and Parametric Analysis". In: Jan. 2018, pp. 197–217. ISBN: 978-3-319-72970-1. DOI: 10.1007/978-3-319-72971-8_10.

[8] Nikolay Simakov et al. "Slurm Simulator: Improving Slurm Scheduler Performance on Large HPC systems by Utilization of Multiple Controllers and Node Sharing". In: July 2018, pp. 1–8. DOI: 10.1145/3219104.3219111.

[9] slurm. *slurm Documentation*. URL: https://bugs.schedmd.com/show_bug.cgi?id=3600. (accessed: 10.02.2020).

[10] HPC Team. *Owl's Nest*. URL: https://www.hpc.temple.edu. (accessed: 13.12.2019).

[11] Steve Traylen. *Torque and Maui Tutoria Documentation*. URL: https://twiki.cern.ch/twiki/pub/LCG/WhiteAreas/TorqueMauiTutorial-Traylen-White-April2008.pdf. (accessed: 10.02.2020).

[12] Star Tutorial. *Star Tutorial*. URL: https://www.startutorial.com/articles/view/ssh-basics-part-1-introduction. (accessed: 24.12.2019).