

Astronomy Cluster Handbook  
(The Hyades-ng Owner's Manual)

Howard Powell  
[howard@virginia.edu](mailto:howard@virginia.edu)

November 21, 2011



# Contents

<b>1</b>	<b>Cluster Policy</b>	<b>1</b>
<b>2</b>	<b>Cluster Specifications</b>	<b>3</b>
2.1	Hyades-ng . . . . .	3
<b>3</b>	<b>Cluster Storage Areas</b>	<b>7</b>
3.1	Overview . . . . .	7
3.2	/home . . . . .	7
3.3	/mnt/lustre . . . . .	8
3.4	/mnt/bigtmp . . . . .	8
3.5	NFS Access From Desktops . . . . .	8
<b>4</b>	<b>Design</b>	<b>9</b>
4.1	Overview . . . . .	9
4.2	The Master Node . . . . .	9
4.2.1	DHCP . . . . .	10
4.2.2	DNS . . . . .	10
4.2.3	LDAP . . . . .	10
4.2.4	TFTP . . . . .	11
4.2.5	NFS . . . . .	11
4.2.6	Syslog . . . . .	11
4.2.7	PBS . . . . .	11
4.2.8	Firewall . . . . .	11
4.3	Compute Nodes . . . . .	12
4.4	Lustre I/O nodes . . . . .	12
4.5	For More Information . . . . .	13
<b>5</b>	<b>Software</b>	<b>15</b>
5.1	Introduction . . . . .	15
5.2	Cluster Tools . . . . .	15

5.3	Software Modules . . . . .	15
5.4	Compilers . . . . .	16
	5.4.1 GNU . . . . .	16
	5.4.2 Intel . . . . .	17
5.5	Libraries . . . . .	18
	5.5.1 OpenMP . . . . .	18
	5.5.2 MPIch2 . . . . .	18
	5.5.3 OpenMPI . . . . .	19
	5.5.4 MVAICH2 . . . . .	19
5.6	Visualization . . . . .	19
	5.6.1 VisIt . . . . .	19
<b>6</b>	<b>PBS</b>	<b>23</b>
6.1	Portable Batch Scheduler . . . . .	23
6.2	Job Scripts . . . . .	24
	6.2.1 MPIch2 job scripts . . . . .	27
	6.2.2 Using the InfiniBand network with MPIch2 . . . . .	27
	6.2.3 MVAICH2 Jobs . . . . .	28
	6.2.4 OpenMP Jobs . . . . .	29
6.3	Work Queues . . . . .	29
6.4	PBS Commands . . . . .	30
<b>7</b>	<b>Known Issues</b>	<b>33</b>
7.1	ssh keys . . . . .	33
7.2	SCP issues . . . . .	34
7.3	DISPLAY issues in IDL . . . . .	34
7.4	Setting up environment based on hostname . . . . .	36
<b>A</b>	<b>PBS</b>	<b>39</b>
<b>B</b>	<b>MPIch2</b>	<b>41</b>
<b>C</b>	<b>OpenMPI</b>	<b>43</b>
<b>D</b>	<b>MVAICH2</b>	<b>45</b>
<b>E</b>	<b>Running a job</b>	<b>47</b>

# Chapter 1

## Cluster Policy

The following is the draft policy for the Hyades-NG computing cluster adopted April 5, 2004:

Hyades-NG is our own dedicated computing cluster designed for parallel computing in the Astronomy Department, UVa. Hyades is primarily intended for large computational jobs which are a burden or impossible to run on a single workstation, regardless of processor speed, etc.

To the effect of who should be granted access on the Hyades-NG cluster, please note that it is the job of the PBS software to handle how to run jobs in the most efficient manner possible. PBS is designed to assign one processor to one dedicated CPU and wait for a successful completion or an error to occur. Since PBS is fairly straightforward and is not very difficult to use, it should be expected that anyone who wants to use Hyades for large computational tasks, should learn to use the PBS software and then use it to launch any jobs on the cluster itself. The main node of the cluster is not assigned to be used by PBS, so if a user wants to make small changes to their code there, then this is perfectly acceptable. The editors vi, vim and emacs are all available on that machine, as well as the GCC and Intel compilers.

For anyone who thinks that PBS is not necessary for their job, another machine would probably be a more appropriate place to run their code. Several multi-processor public Linux workstations are available in the astronomy computer labs with similar hardware specs to the cluster's nodes. Cases which do not fit these criteria should be considered and decided when necessary.

Accounts should be created and assigned by the cluster manager once a user has agreed to follow the guidelines presented here. It is the job of the cluster manager to make sure that the cluster is in a reasonable condition as much as possible and to make available any

tools, software and etc are available when possible.

Documentation on how to use PBS and the cluster should be provided and made available to anyone interested. Please contact [astro-help@mail.astro.virginia.edu](mailto:astro-help@mail.astro.virginia.edu) if you have any questions.

The clusters should be viewed as a special purpose platforms and not as a general workstation as is commonly available elsewhere. This general statement when followed should allow those who need Hyades-NG most to have access they need to get their work done.

Hyades-NG Cluster Policy

Howard Powell

Original Draft - April 5, 2004

Revised - October 4, 2004

Revised - March 15, 2006 - Removed outdated statistics (no. of processors, software)

Revised - July 10, 2007 - Added Pleione cluster information

Revised - Oct 17, 2011 - Removed Pleione cluster information, renamed Hyades to Hyades-NG

## Chapter 2

# Cluster Specifications

### 2.1 Hyades-ng

Hyades-ng was purchased with funding from the ETF allocation for the 2009 school year.

The hardware that enables the Lustre filesystem was purchased by ETF funds from the 2010 ETF cycle.

Four additional High Memory Computation Nodes (funded by ETF 2011) were added October 2011.

The current specifications for the Hyades cluster include:

- Master Node (Hyades-NG.astro.virginia.edu)
  - Dell PowerEdge 2970
    - \* (2x) 2.0GHz AMD Six-core Processors (12 cores total)
    - \* 4800 MHz Front Side Bus
    - \* 512KB L2 Cache (per processor core)
    - \* 6MB L3 Cache (shared by all cores)
    - \* 32GB Ram
    - \* (2x) 146GB 10K RPM SAS Hard Drives *RAID1*
    - \* Infiniband Interconnect card installed
    - \* Dual Gigabit Ethernet (1 private/1 public)

- XCG Submission Node (Sierra.astro.virginia.edu)
  - Dell PowerEdge SC1425
    - \* (2x) 3.0GHz Intel Dual Core Xeon Processors (4 cores total)
    - \* 800 MHz Front Side Bus
    - \* 2MB L2 Cache
    - \* 4GB Ram
    - \* 160GB 7200 RPM SATA Hard Drive
    - \* 250GB 7200 RPM SATA Hard Drive
    - \* Dual Gigabit Ethernet (1 private/1 public)
- Compute Nodes
  - Dell PowerEdge M605 Blade (*Quantity 32*)
    - \* (2x) 2.0GHz AMD Six-core Processors (12 cores total per machine)
    - \* 4800 MHz Front Side Bus
    - \* 512KB L2 Cache (per processor core)
    - \* 6MB L3 Cache (shared by all cores)
    - \* 32GB Ram
    - \* Infiniband Interconnect card installed
    - \* Dual Gigabit Ethernet (1 private/1 unused)
- High Memory Compute Nodes
  - Dell PowerEdge R815 (*quantity 4*)
    - \* (4x) 2.1GHz AMD 12-core Processors (48 cores total per machine)
    - \* 6400 MHz Front Side Bus
    - \* 512KB L2 Cacher (per processor core)
    - \* 12MB L3 Cache (shared by all cores)
    - \* 128GB Ram
    - \* Infiniband Interconnect card installed
    - \* Quad Gigabit Ethernet (1 private/3 unused)

- Lustre Meta Data Server (MDS Node)
  - Dell PowerEdge R415
    - \* (1x) 2.1GHz AMD Six-core Processor
    - \* 6400 MHz Front Side Bus
    - \* 512KB L2 Cache (per processor core)
    - \* 6MB L3 Cache (shared by all cores)
    - \* 16GB Ram
    - \* (2x) 320GB 15K RPM SAS Hard Drives *Hardware RAID 1*
    - \* Dual Gigabit Ethernet (2 private bonded)
- Lustre Object Storage Servers, Object Storage Targets (OSS Nodes)
  - Dell PowerEdge R515 (*Quantity 2*)
    - \* (1x) 2.1GHz AMD Six-core processor
    - \* 6400 MHz Front Side Bus
    - \* 512KB L2 Cache (per processor core)
    - \* 6MB L3 Cache (shared by all cores)
    - \* 16GB Ram
    - \* (4x) 600GB 15K RPM SAS Hard Drives *Hardware RAID 5*
    - \* Dual Gigabit Ethernet (2 private bonded)
- Bigtmp storage
  - (1x) Promise VTrak M610i iSCSI RAID Array
  - 8x 1.5TB 7200 RPM SATA Hard Drives
  - Hardware RAID 6
  - Dual Gigabit Ethernet (2 private)
- Supporting Hardware
  - 8 Port Dell Avocent Network KVM (Keyboard/Video/Mouse) Switch
  - 8 Port Dell Avocent KVM (Keyboard/Video/Mouse) Switch - controlled by Network KVM

- (2x) Dell M600 Blade Chassis with integrated KVM switches, integrated Gigabit Ethernet and integrated 24 port InfiniBand Switches
- 24 Port Dell 10/100/1000 Copper Ethernet Network Switch (Internal IPs)

## Chapter 3

# Cluster Storage Areas

### 3.1 Overview

The Astronomy Cluster system has three different primary storage areas:

- /home
- /mnt/lustre
- /mnt/bigtmp

### 3.2 /home

When you initially log into the cluster, you will be in the /home filesystem. Files stored here are saved on an NFS share of an external server, which provides services to all of the Linux workstations in the department. This filesystem has a user quota assigned to it, and is backed up nightly.

**Heavy I/O performance to the /home filesystem should be avoided.** Overloading this filesystem will directly impact the performance of every Linux workstation in the Astronomy department. It will also perform poorly since NFS is not optimized for large amounts of parallel I/O.

Table 3.1: /net NFS Shares

Cluster Location	Desktop Location
/home	/home
/mnt/lustre	/net/hyades-ng-lustre
/mnt/bigtmp	/net/hyades-ng-bigtmp

### 3.3 /mnt/lustre

The Lustre filesystem is specifically designed for cluster environments where multiple nodes may read and/or write to a file simultaneously. It handles file locking properly, and offers best performance for heavy I/O operations.

**/mnt/lustre is the preferred location to use when running jobs on the cluster.**

Please try to remove your data from /mnt/lustre periodically so that the space is made available for other users of the cluster. /mnt/bigtmp is an area you may consider using for long term storage.

### 3.4 /mnt/bigtmp

/mnt/bigtmp is a large storage area provided by the cluster's master node. Compute nodes cannot access /mnt/bigtmp (again, because NFS is a poor choice for parallel clusters). The iSCSI RAID array is composed of 7200RPM SATA (read: cheaper desktop-grade) disks but offers long-term storage of data that is not critical to access immediately.

### 3.5 NFS Access From Desktops

You can access the lustre and bigtmp storage areas from any departmental linux workstation via the entries in /net (see table 3.1).

# Chapter 4

## Design

### 4.1 Overview

The haydes-ng cluster is a beowulf design where the master node is connected to each compute node through a high speed network. Additionally, there is a sub-cluster within haydes-ng that handles all data I/O for the Lustre filesystem and a job submission host for the UVa Cross Campus Grid project (XCG).

The compute nodes of the cluster are connected through multiple Gigabit Ethernet connections as well as 10GB InfiniBand. The Lustre I/O nodes of the cluster only have access to the Gigabit Ethernet network.

The OS for all parts of the cluster is based on CentOS , which is a clone of the RedHat Enterprise Linux Operating System.

### 4.2 The Master Node

The master node has the bulk of the software and services necessary to make the cluster work.

- DHCP Server (for the 10.x.x.x network)
- DNS Server (for the .cluster internal domain)
- LDAP Server (for user authentication)
- tftp Server (for node PXE-boot)
- NFS server (for /local and for /mnt/bigtmp)

- Syslog server (for cluster node logging)
- PBS server (for PBS job accounting and allocation)
- Firewall (for segregating internal and external networks)

### 4.2.1 DHCP

The Master node runs `dhcpd` to allocate IP address and boot information to all of the compute nodes within the cluster. It's configured to only respond to DHCP requests on the internal network, and `iptables` (the firewall) also guards the system from DHCP requests from outside sources. The information provided by `dhcpd` is used to provide PXEboot information to the compute nodes during a reboot.

`dhcpd` only provides IP information for the ethernet networks. Once any compute node is booted, in the `/etc/rc.local` script there is a section that creates an `ifcfg-ib0` config file for the InfiniBand interface on the node. The IP information used in the `ifcfg-ib0` script is 'stolen' from the IP address used for the Gigabit Ethernet interface - only modified to be on a separate subnet so that both interfaces can co-exist peacefully. Hence, any compute node will have similar IP addresses on each of the Gigabit and InfiniBand networks.

### 4.2.2 DNS

The Master node runs `bind9` to provide DNS information to all of the internal cluster systems. Bind only responds to DNS queries from its internal network interface.

### 4.2.3 LDAP

Since the cluster is based on the same Linux setup as the Astronomy department workstations, any user has access to log into the cluster and submit jobs that has a valid astronomy Linux account.

The master node is an LDAP replica of the main astronomy LDAP server. Any new or updated information is almost immediately pushed to the cluster's master node where it can be queried and used by all of the compute nodes. Hence, creating an astronomy account will provide access to the cluster and all nodes. Updating any account information, such as shell or password, changes it universally amongst the cluster and all Astronomy Linux workstations simultaneously.

#### 4.2.4 TFTP

The compute nodes are diskless clients, and receive their OS image from a trivial FTP service running on the master node. The master node has a heavily customized LiveCD version of CentOS stored on disk, and the tftp service has the job of providing that OS image to all compute nodes on request. The compute nodes then load this OS into memory and boot.

#### 4.2.5 NFS

NFS is not optimized for parallel or heavy I/O operations, so it's not the primary storage system for the cluster. However, NFS is used on the master node to share storage access to the linux workstations outside of the cluster.

#### 4.2.6 Syslog

The compute nodes are all diskless, so any information that might be useful in troubleshooting hardware or software problems is lost when the systems reboot. Syslogd is running on the master node of the cluster, and all compute nodes are configured to provide a copy of any logging information to the master node as well as track the records locally. In case of a node failure, the master node's copy of `/var/log/messages` or `lastlog` can be useful in determining the cause of the problem.

#### 4.2.7 PBS

PBS is the job accounting and allocation software for users on the Hyades-ng cluster. PBS is covered in depth in its own section of this manual. The master node runs the job scheduler and keeps accounting information for audit at a later date.

#### 4.2.8 Firewall

The iptables-based firewall on the master node segregates the internal cluster network traffic from the outside traffic. It also does ipforwarding so that the cluster nodes can communicate directly to the outside world, even though they cannot be accessed from the outside world.

### 4.3 Compute Nodes

The compute nodes are diskless clients (Dell Blade systems). Hence, all IP and OS information has to be provided dynamically at boot and loaded into a local RAM disk. Rebooting a node effectively re-installs all software and resets everything back to a default configuration.

This is accomplished by using the Linux LiveCD project to create a small, customized and efficient OS image based on our cluster needs. Only the software and libraries needed to run jobs is included in the image - User-level programs like firefox and thunderbird are not installed on the compute nodes.

Compute nodes are streamlined to only run jobs and run them as fast as possible. They have no firewalls, and very little security. However, the cluster network is completely segregated from the internet, and no compute node is directly accessible from an external computer. The master node's firewall provides protection and handles the forwarding of traffic between the compute nodes and any outside resources the compute node requests.

Because the compute nodes are diskless and only use RAM disks for the OS, they can become unstable if anything writes a large amount of information to the system. To correct this problem, a hardware and software watchdog system is in place on every node to automatically check for OS errors (such as read-only root filesystems, corrupted system binaries) and automatically reboot a compute node if needed.

### 4.4 Lustre I/O nodes

There are two types of I/O nodes in the hyades-ng Lustre filesystem - OSS/OST nodes and MDS nodes. The OSS/OST nodes contain the actual storage and files written to the Lustre filesystem. The MDS nodes contain the metadata and location where the data is physically stored on the cluster.

An OSS node is an Object Storage Server, an OST node is an Object Storage Target. In the hyades-ng cluster, the storage targets are the hard drives that are inside of the Object Storage Servers. The naming convention is only noted because it's possible to use an external RAID or similar device which would therefore be an OST separate from the OSS nodes.

The MetaData Server (MDS) node contains the ownership, filenames, and all metadata information about every file on the Lustre filesystem. Additionally, the MDS contains the physical server and target information regarding which physical disk contains the file since the files could be located in many different locations.

The Lustre nodes only have access to the Gigabit Ethernet network, but they have multiple network links aggregated together to increase bandwidth and redundancy.

The Lustre nodes all have access to server grade, high-rpm hard drives (15K RPM per disk) which is RAIDed together for redundancy and speed.

In the case of any Lustre node failure, the Lustre filesystem can become unstable. To deal with this potential problem, a hardware and software watchdog is installed on every Lustre node which will detect any problems and reboot the node should it fail. The lustre clients will pause while the Lustre filesystem is unstable, but should return to normal within a few minutes once the affected node restarts. Most jobs should survive without any errors a Lustre node failure.

## 4.5 For More Information

For more information on the diskless node setup and design, please review the August 2011 issue of Linux Journal Magazine for a full article describing the Hyades-NG cluster [1, p. 64]



# Chapter 5

## Software

### 5.1 Introduction

The master node is configured with the same operating system as any Astronomy Department desktop or server and should integrate into the network easily.

The hyades-ng system is fully 64bit, but has the necessary 32 bit compatible libraries to be able to run any code compiled on an intel-based system. When compiling code on the hyades-ng cluster, it is best to plan to compile with a 64 bit compiler.

### 5.2 Cluster Tools

### 5.3 Software Modules

On Hyades-ng there are multiple compilers and libraries available to the users, some of which conflict with each other. As such, we use the modules program to dynamically change your environment to best suit your needs.

```
$ module avail
```

This command lists the available modules. Each module is listed as package/version format.

```
$ module list
```

This lists the programs and modules you currently have loaded.

```
$ module add <program>[/version]
```

This allows you to add to your environment the new package. The program name is mandatory, and you can optionally specify a particular version if more than one is available. Note that adding a package appends the new environment to the beginning of all of the affected environment variables, such as \$PATH. In other words, if you load a new module, the shell will begin looking in the newest places first for commands.

Note that some packages require other packages to be loaded before they can be set up, such as the dependency on the intel compilers (icc-ife) before MPIch (mpich) can be loaded.

```
$ module rm <program>[/version]
```

This unloads any module from your environment.

```
$ module swap <program1>[/version] <program2>[/version]
```

This swaps the two modules in your environment, so that \$PATH and similar variables will check in the places you want in the correct order you want.

```
$ module whatis <program>[/version]
```

This gives a short summary of what each program is that module knows about.

## 5.4 Compilers

There are two sets of C and Fortran compilers available on the astronomy clusters, the Gnu compilers and the Intel compilers.

The GNU compilers (<http://www.gnu.org>) are a set of cross-platform compilers which exist and run on most Unix, Mac and Windows operating systems, and are the default compiler for the Linux operating system. Hence, you will find that many if not most code available on the web is gnu compliant and should compile with these compilers.

The Intel compilers (<http://www.intel.com>) are developed and released by the same company that produces x86 processors for computers such as our cluster, so they offer better performance in most cases for heavy computational work.

### 5.4.1 GNU

The GNU C compiler is very mature and stable and considered the benchmark of compilers amongst many programmers. The GNU Fortran compiler, however, is much newer and less

developed than it's C cousin, and many Fortran programmers will advise you to not plan on using it.

In general, most people who use the astronomy clusters try to compile their code using the Intel compilers since they produce faster and more efficient code.

The GNU compilers are included in the standard \$PATH and environment when you log into the cluster. There is no special module which needs to be loaded.

Some useful flags that will come in handy using gcc and g77 compilers:

```
-m32  
-m64
```

The GNU compilers when installed on a 64 bit system (such as hyades-ng) will default to producing 64 bit compiled binaries. With the -m32 bit flag, you can override this behavior and produce 32 bit binaries instead.

```
-O[ 1 | 2 | 3 ]
```

This turns on optimizations for the code, higher numbers make the compiler more aggressive at trying to optimize code. Over-optimizing can produce slower code, so -O2 is recommended as a default.

### 5.4.2 Intel

The Intel compilers, icc and ifc, are real compilers which produce optimized code for the processor that the cluster is based on. In general, these compilers are considered superior to the GNU compilers, but many times they will have trouble compiling code which gcc otherwise compiles successfully.

Some useful flags that will come in handy using Intel C compiler (underlined choices are recommended):

```
-O< 1 | 2 | 3 | 0 >
```

This sets the optimization level. -O1 and -O2 are equal, -O0 disables optimizations. Over-optimizing code at compile time can produce slower code, so starting with -O2 is a good default.

```
-ip
```

This enables the inter-procedural optimization scheme of the compiler within each input file. This can lead to problems, read the man page for more information. Not recommended.

-ipo

This enables the inter-procedural optimization scheme of the compiler between multiple input files. This often breaks code, so read the man page about this option. Not recommended. Note that the fortran compilers do not accept the -march and -mcpu options in the 7.0 compilers. Use the -tpp option instead (read the man page for more information.).

## 5.5 Libraries

### 5.5.1 OpenMP

OpenMP is a set of routines built into the compiler which allows for parallelization of code. However, the resulting binary is limited by the number of cores on the executing node - in other words, the job can not be distributed across multiple compute nodes in the cluster.

With this in mind, please see the PBS section of this manual for information on how to get exclusive access to a node so that OpenMP jobs do not interfere with other jobs on the cluster.

OpenMP is built into both gcc and the Intel compilers. To use, simply specify the -fopenmp flag (for the gnu compilers) or the -openmp flag (for intel compilers). Please refer to the man pages and google for more information on OpenMP, the compiler flags, and how to use them with your code.

### 5.5.2 MPIch2

MPI version 2 was suggested in 1999 as an update and replacement to the MPI routines. In late 2005, the developers at Argonne National Laboratory released MPIch2. MPIch2 includes backwards compatibility with most MPIch codes, and is generally considered to be the replacement for MPIch.

MPIch2 allows you to specify at compile time which C or Fortran compiler you want to use. To do this set any or all of the following variable to the correct compiler you want to use:

```
$MPICH_CC  
$MPICH_CXX  
$MPICH_F77  
$MPICH_F90
```

To launch an MPIch2 job, use the `mpiexec` command with the new communications library.

```
$ /astro64/bin/mpiexec ./hello_world
```

### 5.5.3 OpenMPI

OpenMPI is not yet working on hyades-ng. Note that OpenMPI is not related to OpenMP, which is built into the compiler.

### 5.5.4 MVAICH2

MVAICH2 is a patched version of MPIch2 which is designed to run directly on the infiniband fabric. This leads to a factor of about 2x speed improvement.

To run a job using MVAICH2, load the modules and compile your code as you would with MPIch2. When launching the job, use `mpirun_rsh`. An example PBS script is in the PBS chapter as well.

## 5.6 Visualization

### 5.6.1 VisIt

VisIt is a tool developed by the Lawrence Livermore National Laboratory (<https://wci.llnl.gov/codes/visit/>). It can be run on a single workstation or the engine can be distributed in parallel on the hyades-ng cluster so that multiple CPUs can be taken advantage of.

To run a job on a single computer on the astronomy Linux network, simply run the executable in `/astro64/visit/bin/visit`.

To run a parallelized version of VisIt, you can use the host profile already provided within visit to launch a job on the hyades-ng cluster. Simply start visit as you would for a single computer, and when you open your data file select hyades-ng instead of localhost. You should be prompted once you select your data for the number of CPUs you wish to use (default=16). Select OK, and a job should be submitted via PBS to the parallel queue on hyades-ng. Assuming the resources are available, the job should start quickly.

The following instructions are just for future reference. You should not need to use them.

To run a manually-parallelized version of VisIt, start the frontend GUI on your workstation via:

```
/astro64/visit/2.0.2/bin/visit -np 8 -norun engine_par
```

In the above, `-np 8` specifies 8 processors. Change this to the appropriate value.

The frontend process should give you some output on your command line, such as:

```
Running: gui2.0.2 -norun engine_par -launchengine localhost -engineargs ;-np;8
Running: viewer2.0.2 -host 127.0.0.1 -port 5600 -norun engine_par -engineargs ;-np;8
-geometry 1318x998+362+26 -borders 23,5,5,5 -shift 5,26 -preshift 0,-3 -defer
-launchengine localhost
Running: mdserver2.0.2 -host 127.0.0.1 -port 5601 -norun engine_par
RUN USING: mpirun -np 8 /astro64/visit/2.0.2/linux-x86_64/bin/engine_par
-host hyades-ng.astro.Virginia.EDU
-port 5600 -norun engine_par -nolookback -plugindir
:/home/staff/hbp4c/.visit/linux-x86_64/plugins
:/astro64/visit/2.0.2/linux-x86_64/plugins
-visithome /astro64/visit/2.0.2
-visitarchhome /astro64/visit/2.0.2/linux-x86_64
-key 6ec00a91d0f269aebab4
```

Be sure to set your environment variables:

```
LD_LIBRARY_PATH=/astro64/visit/2.0.2/linux-x86_64/lib:
/astro64/intel/compiler_11.0.081/lib/intel64
:/astro64/intel/latest/lib:/usr/lib/xorg:
/usr/lib64/xorg:/usr/local/lib
LD_LIBRARY32_PATH=/astro64/visit/2.0.2/linux-x86_64/lib
LD_LIBRARYN32_PATH=/astro64/visit/2.0.2/linux-x86_64/lib
LD_LIBRARY64_PATH=/astro64/visit/2.0.2/linux-x86_64/lib
LIBPATH=/astro64/visit/2.0.2/linux-x86_64/lib
VISITHOME=/astro64/visit/2.0.2
VISITHELPHOME=/astro64/visit/2.0.2/linux-x86_64/help
VISITULTRAHOME=/astro64/visit/2.0.2/linux-x86_64/ultrawrapper
VISITPLUGINDIR=:/home/staff/hbp4c/.visit/linux-x86_64/plugins:
/astro64/visit/2.0.2/linux-x86_64/plugins
PYTHONHOME=
TRAP_FPE=
MESA_GFX=disable
```

You will then need to submit a job to the PBS queue to start the engine on `hyades-ng`. The output of the frontend visit GUI will give you the information you need to substitute in the correct values in the PBS script below:

```
#!/bin/bash
#PBS -l walltime=00:15:00
```

```

#PBS -l select=8
#PBS -M hbp4c@virginia.edu
#PBS -j oe
#PBS -o visit.out

echo Running on host `hostname`
echo Time is `date`
echo "Nodes used for this job:"
echo "_____ "
cat $PBS_NODEFILE
echo "_____ "

cd $PBS_O_WORKDIR

source /net/astro64/intel/latest/bin/iccvars.sh intel64

LD_LIBRARY_PATH=/net/astro64/visit/2.0.2/linux-x86_64/lib:\
/astro64/intel/compiler_11.0.081/lib/intel64:\
/astro64/intel/latest/lib:/usr/lib/xorg:/usr/lib64/xorg:\
/usr/local/lib
LD_LIBRARY32_PATH=/net/astro64/visit/2.0.2/linux-x86_64/lib
LD_LIBRARYN32_PATH=/astro64/visit/2.0.2/linux-x86_64/lib
LD_LIBRARY64_PATH=/astro64/visit/2.0.2/linux-x86_64/lib
LIBPATH=/net/astro64/visit/2.0.2/linux-x86_64/lib
VISITHOME=/astro64/visit/2.0.2
VISITHELPHOME=/astro64/visit/2.0.2/linux-x86_64/help
VISITULTRAHOME=/astro64/visit/2.0.2/linux-x86_64/ultrawrapper
VISITPLUGINDIR=:/home/staff/hbp4c/.visit/linux-x86_64/plugins\
:/astro64/visit/2.0.2/linux-x86_64/plugins
PYTHONHOME=
TRAP_FPE=
MESA_GLX_FX=disable

/astro64/bin/mpixexec /astro64/visit/2.0.2/linux-x86_64/bin/engine_par \
  -host hyades-ng.astro.Virginia.EDU \
  -port 5600 \
  -norun engine_par \
  -noloopback \
  -pluginindir :/home/staff/hbp4c/.visit/linux-x86_64/plugins\
    :/astro64/visit/2.0.2/linux-x86_64/plugins \
  -visithome /astro64/visit/2.0.2 \
  -visitarchhome /astro64/visit/2.0.2/linux-x86_64 \
  -key 6ec00a91d0f269aebab4

```

```
echo  
echo "Job Ended at `date`"  
echo
```

Note that I substituted the suggested mpirun statement with `/astro64/bin/mpiexec`, which plays better with PBS. Also note that each time you run this setup, the key generated by the visit frontend will change to a new random value. Please update these as appropriate in your PBS script.

# Chapter 6

## PBS

### 6.1 Portable Batch Scheduler

The Portable Batch Scheduler is a piece of software created by Altair Engineering (<http://www.openpbs.com>). This software follows a set of site-designed rules to govern how jobs can be submitted and run on a multi-node computing cluster such as ours.

PBS works by defining a set of work queues which you submit your job to via a specially crafted shell script. These queues are then assigned priorities according to what type of job you are running and the required resources, then automatically submits of your behalf the actual program to all the computing nodes on the cluster to run.

The primary PBS resources available and can be user controlled are:

- Nodes
- Processors per Node
- Memory
- Wall Clock Time

The PBS job script includes directives with which you can request limits for each of the resources. If your job is too big for one particular queue, the job will be moved to another queue unless the job is too complex to be run on the cluster.

The section on queues explains the limits of each queue. First we'll show an example job script.

## 6.2 Job Scripts

A sample (single threaded) job script is shown below. You should notice that its format looks like a simple bash script, with a slightly modified header section.

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l select=1
#PBS -l walltime=96:00:00
#PBS -o seti.out
#PBS -e seti.error
#PBS -m abe
#PBS -M hbp4c@virginia.edu

# Print some information
echo "#####"
echo "Starting on 'hostname' at 'date'"

echo
echo "Node used for this job:"
echo "_____"
```

```
cat $PBS_NODEFILE
echo "_____"
```

```
# Change to the working directory
cd $PBS_O_WORKDIR

cp /home/hbp4c/setiathome/user_info.sah ./
./setiathome -nice 19 -nolock

# print end time
echo
echo "Job Ended at 'date'"
echo "#####"
```

The header section includes the PBS directives that are unique to that program.

```
#PBS -l select=1
#PBS -l walltime=96:00:00
```

The `-l` directives directly modify how your script will be delivered to the computing nodes on the cluster. Here, you see that we have modified two PBS directives, wall time and number of nodes required for the job.

The "select" statement is the total number of chunks of resources you wish to use for your job. A chunk is a group of resources - at it's most basic level it's the total number of cores you wish to use in your job. In the above example, we want to use one chunk of resources. PBS assumes this means one node, and one cpu core on that node.

If you wish to allocate a specific number of cpus per node (and then the remaining cpus on other nodes) then try the `select=X:ncpus=Y` statement:

```
#PBS -l select=2:ncpus=6
```

That example would select 2 chunks of resources, each using 6 cpus, for a total of 12 cpu cores allocated to your job.

Note that PBS might pack both selected chunks onto the same compute node if it has 12 cpu cores available, or it may use two compute nodes if not. To control this better, use the "-l place" directive:

```
#PBS -l select=2:ncpus=6 -l place=scatter
```

The above spreads out jobs as much as possible up to the total number of chunks selected. In this example, we selected two chunks of resources, so PBS would spread the job out amongst two compute nodes.

```
#PBS -l select=2:ncpus=6 -l place=pack
```

The above consolidates resources as much as possible onto as few nodes as possible. In this example, PBS will try to consolidate both chunks of cpu cores onto the same node if possible, but will use two nodes if necessary to run the job.

```
#PBS -l select=2:ncpus=6 -l place=free
```

And this puts the job anywhere that PBS can fit it.

You may also wish to limit a job to a very specific compute node. This should be very rare. To do this, you must specify the real host name:

```
#PBS -l select=1:ncpus=12:host=eth-c2
```

Note that the full hostname of any node has a prefix "eth-cXX" for the ethernet interface.

Instead of (or in addition to) selecting the number of CPU cores to use, you may need to specify how much shared memory on a host you need access to. To do so, use the `mem` PBS directive:

```
#PBS -l mem=1800mb
```

Variable	Definition
<code>\$PBS_NODEFILE</code>	This is the list of machines used by MPI to decide what hosts use.
<code>\$PBS_0_WORKDIR</code>	This is the working directory the user submitted the job from.
<code>\$PBS_JOBID</code>	The PBS assigned number of the current job.

Note that if you exceed the limitations on the work queues, then the job cannot be submitted and you will receive an error. Also, if your job requires more resources than it was allocated, it will die. If in doubt, it is best to overestimate a little what your requirements are for things like memory and walltime, or leave those directives blank (which means your script will receive the defaults for the queue it is submitted to). You will have to be accurate on the nodes required and processors per node directives.

```
#PBS -o seti.out
#PBS -e seti.error
```

The `-o` and `-e` arguments control where to write output and error log files to.

```
#PBS -j oe
#PBS -o seti.log
```

Additionally, you can join output and error with the `-j` option.

```
#PBS -W umask=022
```

Any output that PBS produces will not be readable by other users of the cluster. If you want other people to read your output files, you can instruct PBS to allow this with a `umask`.

```
#PBS -m abe
#PBS -M hbp4c@virginia.edu
```

Finally, the last part of the header explains how and when to email you when PBS does anything with the script. The `-M` argument requires one variable, your email to send alerts to when PBS generates them. The `-m` option specifies when to send you alerts; a is for when a job aborts, b for when a job begins, and e for when a job ends.

The remainder of the script uses the same syntax as a bash script which requires no user interaction. If your program requires user interaction at any point, PBS will not be able to run it and the program will simply sit on the processor idle until the walltime expires.

PBS defines a few variables that the script can access once the job is submitted. The most useful ones are included in table 6.1. A complete list can be found in the PBS Pro User's Guide Appendix.

### 6.2.1 MPIch2 job scripts

Here is a sample MPIch2 script.

```
#PBS -S /bin/bash
#PBS -l walltime=00:15:00
#PBS -l select=8
#PBS -M hbp4c@virginia.edu
#PBS -j oe
#PBS -o hello.out
#PBS -V

echo Running on host `hostname`
echo Time is `date`
echo "Nodes used for this job:"
echo "_____"
```

```
cat $PBS_NODEFILE
echo "_____"
```

```
cd $PBS_O_WORKDIR

/astro64/bin/mpiexec ./hello_world

echo
echo "Job Ended at `date`"
echo
```

Please refer to the section on MPIch2 in this manual for more details. Note that I also used the `-V` PBS directive to inherit all my environment variables so that I didn't have to load modules within the PBS script. Also note that the tic marks above are `"` marks, (on the same key as the tilde mark).

### 6.2.2 Using the InfiniBand network with MPIch2

The InfiniBand interconnect will offer much better performance for jobs that require a large amount of communication between compute nodes (ie. fine-grain mpi programs). However, keep in mind that any communication with a shared disk (either NFS or PVFS based disks) will occur over the gigabit ethernet interface and not the InfiniBand network.

Here is an example of an InfiniBand mpich2 job:

```
#PBS -S /bin/bash
#PBS -l walltime=00:15:00
#PBBS -l select=8
```

```

#PBS -M hbp4c@virginia.edu
#PBS -j oe
#PBS -o hello.out
#PBS -V

/astro64/bin/mpixec -transform-hostname 's/eth/ib/' ./hello_world

echo
echo "Job Ended at 'date'"
echo

```

Those of us who know sed or have used the Vi editor will recognize we're substituting anything that looks like 'eth' with 'ib' instead. What is happening here is the hostname 'eth-c0' and 'ib-c0' connect to the same machine; however 'eth-c0' refers to the ethernet network name, and 'ib-c0' refers to the InfiniBand network name. By substituting IB for eth, we're telling the nodes to talk to one another via the InfiniBand interfaces only.

There is one technical note on this type of setup - MPIch uses standard TCP/IP to communicate between nodes. When using MPIch2 with Infiniband, you'll incur some overhead due to strict TCP packet structures.

The InfiniBand interface is rated a 4x (10Gb/s) speed and has a latency on the order of tens of milliseconds, versus Gigabit Ethernet (1Gb/s) speed with a latency of between 100 and 200 milliseconds. On the Hyades-ng cluster, a fine-grained MPI job can expect a few fold increase in performance, limited by the processing speed of the CPU in most cases.

### 6.2.3 MVAPICH2 Jobs

Using MVAPICH2 should create a faster executable - since the communications will run directly on the infiniband fabric and does not require TCP/IP frameworks to pass between nodes.

Here is an example MVAPICH2 PBS script:

```

#PBS -S /bin/bash
#PBS -l walltime=00:15:00
#PBS -l select=8
#PBS -M hbp4c@virginia.edu
#PBS -j oe
#PBS -o hello.out
#PBS -V

/astro64/bin/mpmixec ./hello_world

```

```
echo
echo "Job Ended at `date`"
echo
```

Make sure that the number of processors (-np) matches the PBS select statement in the header. The hostfile is provided by the \$PBS\_NODEFILE variable.

### 6.2.4 OpenMP Jobs

Please note that OpenMP is a different technology from OpenMPI. OpenMP is a parallel routine built into both the gnu and intel compilers.

The limitation of OpenMP is that all of the threads must execute on the same node as the master thread - so your entire job must exist on one node of the cluster. To do so, you must request exclusive access to a node from your PBS script.

```
#!/bin/bash
#PBS -l walltime=00:15:00
#PBS -l select=1:ncpus=12
#PBS -l place=pack
#PBS -M hbp4c@virginia.edu
#PBS -j oe
#PBS -o openmp.out
#PBS -V

cd $PBS_O_WORKDIR

./a.out

echo
echo "Job Ended at `date`"
echo
```

## 6.3 Work Queues

There are currently four work queues enabled on Hyades-ng: debug, huge\_jobs, parallel and junk. Additionally, there is one special queue which routes jobs into the other queues automatically based on the resources which they request. The astro queue, (this routing queue) is the default queue. Most jobs should be submitted to this queue.

If you define your PBS directives in your job script, then the astro queue will route to the correct location your job.

Table 6.2: Hyades-ng Cluster PBS Queues

Queue	Min CPUs	Max CPUs	Walltime Limit	Priority
astro	<i>This is a routing queue</i>			
	<i>Any jobs submitted will be redirected to the Parallel, Junk or the Single queue.</i>			
Debug	1	32	00:15:00	185
Huge_jobs	1	<i>Cluster Maximum</i>	24:00:00	175
Parallel	2	512	24:00:00	160
Single	1	12	96:00:00	140
Junk	1	<i>Cluster Maximum</i>	168:00:00	120

*Note:* Queues with a lower priority value will be suspended by higher priority jobs.

The debug queue and huge\_jobs queue are both direct-access only queues; jobs will never be routed to either queue automatically. All other queues will only accept jobs routed to them from the default queue (Astro).

The huge\_jobs queue is not normally enabled by default since it can tie up the entire cluster and prevent others from using it. Jobs which get submitted to huge\_jobs will not run without manual intervention by the cluster administrator. To run a job here, please contact your friendly local sysadmin to help you.

On the Hyades-ng cluster, preference and priority will be given to parallel jobs which request more CPUs.

## 6.4 PBS Commands

```
$ qsub [-q queue] [-V] <job.sh>
```

To submit a job to a work queue, you can use the qsub command. Qsub requires at least one argument, the name of the job submission script, but can optionally take a *-q* option with the name of the queue you wish to submit your job to. In our case, if you simply qsub a script, it will try to be submitted to the astro queue. Your job will then be routed to the correct work queue automatically. The *-V* option will pass the current working environment (including your \$PATH information) along to PBS, otherwise you'll have to initialize the environment yourself inside the script.

```
$ qdel <job id>
```

To delete a job for any reason at any time, you can use the qdel command with the job number of the job you wish to kill.

```
$ qstat [-a] [-n]
```

To check the status of a job, you can use the `qstat` command to list all the jobs currently on the cluster. Optionally, you can use `-a` to see the time the jobs have been running as well as the user and current status of the job. Status will most likely be one of Q for queued, S for suspended or R for running. The `-n` option for `qstat` will show exactly which nodes the job is running on.

There is also a `xpbs` program which is a graphical interface to `qsub` and `qstat` for workstations capable of displaying X graphics (such as Linux and Sun workstations). `xpbs` is a good way to keep tabs on the work queue but I find it's job submission interface a bit more complicated than just using `qsub`.

```
$ pbsnodes -a
```

This command is part of the `pbs` suite of tools for job control and submission on the astronomy clusters. Specifically, this command will list all the computing nodes available to the cluster, and give complete details about what they are doing at this given moment in time. This information will include the job number they are running from PBS, as well as the resources available on each node of the cluster. PBS will be covered more later in this document.



# Chapter 7

## Known Issues

There are a couple of problems that you may run into. These are not necessarily issues with the software on the cluster.

### 7.1 ssh keys

I've set up Host-based SSH authentication on the hyades-ng cluster, so users should not have to worry about ssh issues.

In case of problems, I have set up a script called `/local/bin/setup-ssh.sh` which should help automate this section. You must run this script on the hyades-ng master node.

You must set up your shell so that ssh can communicate between nodes without a password. This can be done with the `ssh-keygen` program.

```
[hbp4c@Saguaro ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/staff/hbp4c/.ssh/id_rsa):
Created directory '/home/staff/hbp4c/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/staff/hbp4c/.ssh/id_rsa.
Your public key has been saved in /home/staff/hbp4c/.ssh/id_rsa.pub.
The key fingerprint is:
06:bd:49:67:f1:ca:17:a5:a8:53:01:7a:7c:be:fd:d4
hbp4c@Saguaro.astro.virginia.edu
```

In the above example, I created a ssh key without a passphrase. This created a file in my unix home directory called `.ssh/id_rsa`.

To complete this process, I have to inform the ssh program that I can use this key as authentication:

```
[hbp4c@Saguaro ~]$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

The public key is copied to the remote host and added to the authorized keys file in the ssh directory. Now I can ssh from the host that created the key to the host that received the public key.

In the case of the astronomoy home directories, which are shared between all linux systems (including the nodes on `hyades-ng`), this effectively means we can ssh without a password from any linux machine to any linux machine within the department.

Note that if any malicious user gets a copy of your `id_rsa` (the private key) they will be able to log in as you to any host which has the public key authorized. It would be a much better idea to use a passphrase with the `ssh-keygen` program, so that your private key still requires a password to use.

To make a private key with a password work, you will need to start the `ssh-agent` on each remote system you want to log into (i.e. - every node you want to use on the cluster including the master node). As soon as I figure out how to do this, I'll add an example PBS script.

## 7.2 SCP issues

`scp` is how `stderr` and `stdout` are moved between nodes and the working directory on the master node. `scp` has a few nuances. First, you have to have a working, password-less ssh working. Also, if any output to `stderr` or `stdout` are created during a ssh login, `scp` will break.

In other words, make sure that there are no echo statements or errors in your `.cshrc` or `.bashrc` files. If there are any statements or errors outputted during login, you will not be able to `scp`.

## 7.3 DISPLAY issues in IDL

The compute nodes on the cluster do not have the X11 environment installed. If you try to launch IDL, you'll get an error about cannot set the `DEVICE` variable to `localhost:(some`

number). Multiple ways to work around this can be found at:  
[http://www.dfanning.com/misc\\_tips/noxserver.html](http://www.dfanning.com/misc_tips/noxserver.html).

Option 1: Set your DEVICE to 'null' within your IDL routine:

```
Set_Plot , 'NULL'
```

This draws all windows in an offscreen device.

Option 2: Use the 'Z' buffer:

```
Set_Plot , 'Z'
DEVICE, SET_PIXEL_DEPTH    = 24, $
      SET_RESOLUTION      = [1024, 768], $
      SET_CHARACTER_SIZE  = [6, 10], $
      Z_BUFFERING         = 0
```

This also draws all windows to an offscreen device - here we have created a 24 bit, 1024x768 virtual screen to render graphics to. The Z.buffer is disabled, so this is best for 2D plots. If you want to do a 3D plot, omit the Z\_BUFFER statement above.

Option 3: Use a Xvfb device:

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -l select=1
#PBS -l walltime=96:00:00
#PBS -o idl.out
#PBS -e idl.error
#PBS -m abe
#PBS -M hbp4c@virginia.edu

# Print some useful information.
echo "#####"
echo "Starting on 'hostname' at 'date'"

echo
echo "Node used for this job:"
echo "_____"
```

```
cat $PBS_NODEFILE
echo "_____"
```

```
# Change to the working directory
cd $PBS_O_WORKDIR

/usr/X11R6/bin/Xvfb :1 -screen 0 1280x1024x24 -ac -terminate &
```

```

export DISPLAY :1.0

idl -e script

killall -9 Xvfb

# print end time
echo
echo "Job Ended at `date`"
echo "#####"

```

In this case, I've included the full PBS script to demonstrate how to set up the Xvfb device before starting IDL, and how to kill it once the IDL script has completed. In this case, I do not need to modify my `DEVICE` variable within IDL - it should automatically pick up the new `DISPLAY` variable from your shell.

## 7.4 Setting up environment based on hostname

It may be useful to set up your `.cshrc` or `.bashrc` files so that when you're running a job on the cluster you get a slightly different environment - for example a different `idl` startup routine or a different `$PATH`. The master node is set up like any department workstation, so your standard environment should work on the master node. However, the compute nodes are very stripped down machines - for example, they do not have X11 installed. Any programs in your environment may not work correctly given these limitations.

It's easy to set up your `.cshrc` and `.bashrc` files so that they execute differently when inside of a job on the cluster. Simply include a case statement that looks for the format `c*.cluster`:

`.cshrc` format:

```

switch ($HOST)
case "Hyades-ng.astro.virginia.edu":
    # Do stuff here that only works on the master node.
    breaksw
case "c*.cluster":
    # Do stuff here that only happens on the compute nodes.
    breaksw
case "*":
    # Do the default here, this will happen on any other computer.
    breaksw

```

endsw

.bashrc format:

```
case $HOST in
  Hyades-ng.astro.virginia.edu )
    # Do stuff here that only happens on the master node.;;
  c*.cluster )
    # Do stuff here that only happens on the cluster nodes.;;
  * )
    # Do stuff here that happens on non-cluster machines.;;
esac
```

Note that in the bash version, you have to close your case statements with a double semi-colon: ;;.



# Appendix A

## PBS

<http://www.pbspro.com/UserArea>

PBS main site, for documentation and software downloads.



# Appendix B

## MPIch2

The MPIch2 Website can be found at:

<http://www-unix.mcs.anl.gov/mpi/mpich/>

The MPIch2 User's Guide can be found at:

<http://www-unix.mcs.anl.gov/mpi/mpich/downloads/mpich2-doc-user.pdf>

<http://www.osc.edu/~pw/mpiexec/index.php>

mpiexec is a replacement for the mpirun command included with MPIch and MPIch2.



## Appendix C

# OpenMPI

The OpenMPI suite can be found at:

<http://www.open-mpi.org/>



# Appendix D

## MVAPICH2

The MVAPICH2 website can be found at:  
<http://mvapich.cse.ohio-state.edu/>



## Appendix E

# Running a job

As an example of how to run an InfiniBand-optimized MPI job on hyades-ng:

First, log into the cluster and initialize your environment:

```
[hbp4c@hyades-ng ~]$ module add icc-ipc
[hbp4c@hyades-ng ~]$ module add mvapich2
[hbp4c@hyades-ng ~]$ mkdir /mnt/lustre/hbp4c/test-job
[hbp4c@hyades-ng ~]$ cd /mnt/lustre/hbp4c/test-job
```

Next, compile your code with mpif90:

```
[hbp4c@hyades-ng ~]$ mpif90 -o a.out testcode.f90
```

Then, set up a job file for PBS which looks something like:

```
#!/bin/bash
#PBS -l walltime=00:15:00
#PBS -l select=32
#PBS -M hbp4c@virginia.edu
#PBS -m abe
#PBS -j oe
#PBS -o hello.out
#PBS -V

echo Time is `date`
cd $PBS_O_WORKDIR
mpirun_rsh -np 32 -hostfile $PBS_NODEFILE ./a.out
echo "Job Ended at `date`"
```

Finally, submit your job to PBS:

```
[hbp4c@hyades-ng ~]$ qsub -q debug job.sh  
[hbp4c@hyades-ng ~]$ qstat -n
```

Once you see the job begin executing (taking up wall time and being assigned nodes) then you should be set!

# Bibliography

- [1] Howard Powell. Build a Beowulf HPC System with the Fedora LiveCD Project. *Linux Journal*, pages 64–68, August 2011.

