

Case Study: Setting up and running a production Linux cluster at Pacific Northwest National Laboratory

*Gary Skouson, Molecular Science Computing Facility, William R. Wiley
Environmental Molecular Sciences Laboratory, Pacific
Northwest National Laboratory*

Ryan Braby, Molecular Science Computing Facility, William R. Wiley
Environmental Molecular Sciences Laboratory, Pacific
Northwest National Laboratory*

Abstract

With the low price and increasing performance of commodity computer hardware, it is important to study the viability of using clusters of relatively inexpensive computers to produce a stable system, capable of the current demands for high performance massively parallel computing. A 192-processor cluster was installed to test and develop methods that would make the PC cluster a workable alternative to using other commercial systems for use in scientific research. By comparing PC clusters with the cluster systems sold commercially, it became apparent that the tools to manage the PC cluster as a single system were not as robust or as well integrated as in many commercial systems. This paper is focused on the problems encountered and solutions used to stabilize this cluster for both production and development use. This included the use of extra hardware such as remote power control units and multi-port adapters to provide remote access to both the system console and system power. A Gigaset cLAN fabric was also used to provide a high-speed, low-latency interconnect. Software solutions were used for resource management, job scheduling and accounting, parallel filesystems, remote network installation and system monitoring. Although there are still some tools missing for debugging hardware problems, the PC cluster continues to be very stable and useful for users.

* Currently at Livermore Computing, Lawrence Livermore National Laboratory

1 Introduction

At the Pacific Northwest National Laboratory, the users of the Molecular Science Computing Facility (MSCF) routinely employ IBM SP systems to run large parallel jobs for scientific research. Because of the increased popularity of Linux and the cost/performance numbers shown for PC systems, it was decided to build and test a Linux cluster for use as a computational resource. However there are many features in the large commercial systems that are not packaged with a Linux cluster. For example, there is not a packaged method to get remote console access to a node in order to debug boot or networking problems. To install or work on a node would require an administrator to go to the computer facility and connect a monitor and keyboard to the box. When all of the systems arrived, someone had to walk around with a diskette and a CD to install each of the nodes. These and other problems make the installation and use of a Linux cluster somewhat tedious. However, with additional hardware, planning and software, it has been possible to build a relatively stable and easy to manage system. This cluster was named Colony, which is the term for a group of penguins. This seemed appropriate because the Linux mascot, Tux, is a penguin.

1.1 Hardware

To assemble a high performance Linux cluster, 98 Dell PowerEdge servers were purchased and installed. The base configuration of each machine is a Dual Pentium III 500 MHz, with 512 MBs of RAM, 9 GBs of SCSI disk, Gigaset cLAN adapter, and 100 Mb Ethernet adapter. With the 192 processors, it has a theoretical peak performance of 97 gigaflops. Five of the nodes were purchased to act as file system servers, with 1 GB of RAM, RAID adapters, and 50 GBs or more of disk. Two of the nodes were purchased for large serial jobs, with 2 GBs of RAM, RAID controllers, and lots of disk. Also four of the nodes were purchased to be dual boot with a second 9 GB hard disk. This hardware configuration forced the solutions that were used, to work on heterogeneous systems.

2 Installation

When the cluster was purchased and installed, the people installing the nodes carried around a diskette and a CD and installed each system individually. Because they used RedHat Linux with a kickstart file, this was not difficult, just labor intensive. After getting all of the nodes set up and installed, numerous items had to be changed on all of the nodes. These changes included: changing configuration files, adding packages to the nodes, and other modifications to the kernel and modules that were necessary. However by doing this, it became more difficult to get a failed node back to the same configuration as the rest of the cluster. It was also difficult to reinstall all 96 nodes by this method.

After the systems are up and running, it is important to keep the configuration of many of the files on each node at the same revision level. To do that, the *cfengine* program was used. This allows an administrator to change a file on one system and have all of the other nodes of the cluster updated with these changes. *Cfengine* is discussed in more detail below.

2.1 Desired features

The features that we desired from the installation program were: 1) No manual intervention in the computer facility. It should work from the administrator's office, remote locations such as an administrator's residence, etc. 2) It would need to deal with the non-standard installation that we had implemented. The focus was on having the system available and usable for users, not in a state of operating system flux requiring many reinstalls and much system downtime. 3) It should be able to deal with different hardware without significant problems in case more nodes were added to the cluster. 4) Several of the above require that the network be used for the installation

2.2 Possible solutions

In looking over the packages that were available, three main solutions were found that were in use by others. The simplest and probably most widely used method for installing large numbers of systems was the use of kickstart files, using the Red Hat installation method. This method can be easily used to install systems over the network. The main drawback was that it would be necessary to write post-install software to take care of the non-standard features that were installed or it would require all of the extra features to be packaged as RPM files. While the idea of having everything in packages for installation was nice, the RPM packages weren't there yet and time was not available to package all that was needed. The other drawback was that the installation method required some extra work to make it boot the nodes without someone putting in a diskette.

With VA SystemImager it was very easy to get from whatever setup is being used to having an image to install on other systems. It does not care about what kind of packages the system comes from. It just takes an image of a running system that has been chosen and uses that image to install the other nodes. The main drawback to this approach was that it was going to require that all of the hardware was the same. Also, the system for managing different images was somewhat more tedious with SystemImager.

The LUI (Linux Utility for Installation) system was chosen for the installation because it represented a compromise between the above methods. It would allow the installation image to be made from a running system or to have a list of RPM packages to install from, or some combination of the two. It also had packaged methods to allow the node to be booted from either the network or

from the hard disk depending on the installation server settings. Direct system administrator physical intervention is not required. Another benefit was the ease of configuration management for different hardware. This would give the solution that was needed in the short term and allow for a migration to package-based installation in the future.

2.3 Using Etherboot and LUI

The installation of the Etherboot package to create Ethernet boot ROM images was required in order to get LUI set up. These boot ROM images could be copied onto diskettes rather than burned onto an Ethernet boot ROM. The Ethernet boot setup can be made to boot from the network or the local disk and can also be configured to check the network for boot instructions. In addition to these boot options, the console I/O can use either the normal keyboard and monitor or a serial port. Using this image, the DHCP or BOOTP server can specify whether the node should boot from the network or from the local disk. Normally the node could be set to check the network for boot instructions. If there is no response from the network the node may be booted from the local disk. The DHCP or BOOTP server can also determine whether the node will be reinstalled from a network boot or if it should boot to a maintenance mode to allow an administrator to check or fix problems without a reinstall.

To install a node, requires that node parameters be entered into the LIM database. A set of system resources must be defined in LIM. This is done with the *mklmr* command to define disk partition information, partition contents, RPM packages to be installed, kernel images, etc. to be used in installing cluster nodes. For each node or group of nodes, the administrator can then allocate any of the defined resources to be used for the install. This allows many custom configurations for different hardware types or needs, without having to have an entire installation for each node type. For the initial use of the LUI software, tarballs from working nodes were used as resources in LIM for installation on subsequent node installations. Incremental changes have been made to increase the stability of nodes on installations. For example, when the nodes were initially installed, a 2GB partition was used as the root partition with /scratch and swap being the only other partitions on the disk. This can lead to problems as the root filesystem can fill up from programs dumping too much data to /tmp. This can easily be fixed for new node installations by simply changing the partition resource to be used for installation. Further node installs can then have separate partitions for /tmp, /usr, etc. For more information on LUI, see the web page at: <http://oss.software.ibm.com/developerworks/projects/lui>

2.4 Cfengine

The *cfengine* program is not only part of the installation method, but also part of the maintenance method. *Cfengine* allows the administrator to modify one set of files, which will then be propagated to the other nodes in the cluster. After

installing all of the software using LUI, *cfengine* is run as a post-installation method. This does much of the customisation of the different node types.

By creating a configuration file, the administrator can specify the files that should be copied and their locations. These files will be checked to see if there are differences and the files are copied when updates are needed. One important feature of *cfengine* is that it is not a one-size-fits-all configuration. It is quite easy to create different configurations based on groups of systems or custom configurations for one system in particular. This can all be done within the same *cfengine* configuration file.

In this cluster in particular, some of the nodes are allocated as login nodes. On these nodes features, programs and configurations appropriate for login users need to be available. For example commands to query and submit to the batch queue need to be available. These nodes also need to be configured to allow access to things external to the cluster and to allow for external login etc. All of this can be grouped in a “login” group within *cfengine*. In addition to login nodes, there can be compute nodes, file servers, etc., each needing some slightly unique settings and files. This can easily be accomplished from the same control file.

As well as file copies based on node groups, *cfengine* also has a number of other useful system administration and configuration control features. It has a robust system for managing file links, allowing links to be maintained on all nodes from one directory to different NFS filesystems where user home directories exist. It also has an “AppendIfNoSuchLine” construct that is used to make sure certain services, including *cfengine* itself, are enabled in some configuration files. *Cfengine* also provides the ability to disable *.rhosts* files and other high security risk items. *Cfengine* can be used to run shell scripts on various nodegroups. To provide a high level of control over what *cfengine* does at each running, it provides configurable action sequences, which allow it to perform different functions depending on how it is called.

3 Power Control

There will always be hardware failures on computer systems. As the number of nodes on a system increases, so also will the likelihood that some part of the system will fail. When a node fails and needs to be rebooted, it is helpful if an administrator happens to be logged in at that time and is able to reboot the node. However, there are numerous times when the reason that the node needs to be rebooted is that the node has failed to respond to any input. For normal PC systems this means that the administrator has to be physically located at the system and reboot the node. For a cluster that will be in use continuously, it is good to be able to cycle the node power remotely, especially when the system administrators are not on-site.

To be able to have the option of cycling the node power remotely, BayTech RPC3-20 remote power control boxes were purchased. The eight power outlets on these boxes can be controlled using IP connections to the Ethernet port. Using telnet, the administrator can use the built-in menus to control the state of the power to individual or multiple power outlets. In order to simplify this control, a Perl script was used. A configuration file is used that maps the node names to the power control boxes and power outlets. This script can then query the state of the power ports or change the state for any given node. It will also accept Perl regular expressions to allow a range of systems to be powered off or on.

The power program can be used as:

```
# power -show p01n1[3-9]

Power Status:

p01n13- [On ]   p01n15- [On ]   p01n17- [On ]   p01n19- [On ]
p01n14- [On ]   p01n16- [On ]   p01n18- [On ]
```

Other arguments to the *power* command are *-reset*, *-on* and *-off*. These commands will cycle the power or turn the power to the nodes on or off. This ability has saved numerous trips to the computer facility for down nodes. It has also reduced the number of times when the wrong system was powered off.

4 Available Console

One of the big selling points of commercial HPC systems, such as the IBM SP, is that most administrative tasks can be done remotely. Even machines with a bad Ethernet card can be dealt with on an SP using the *s/term* command to pull up a serial console. On commercial HPC systems, these solutions are largely hardware based. For instance Compaq Alpha systems have a bios that includes serial console and remote power control of the node. How the remote power control was handled for Colony was discussed above. This section will address the serial console management.

Our Linux cluster, Colony, was delivered without any working console management solution. The initial solution to a problem node was a monitor, keyboard, and mouse on a cart that was rolled over to the crashed node, connected, and used to debug the problem. However, since this required a significant amount of time, another solution was needed. This solution was found largely in the Linux kernel. For some time now, the Linux kernel has supported serial consoles. This needed to be enabled on all the clients and some method of connecting all these serial consoles to a server was needed. Cyclades-Z serial multiport adapters were placed in three servers, each with 32 RJ-45 serial connections. Each server would provide serial console access to a

third of Colony. Cyclades support was compiled into the kernels on these servers through use of the `CONFIG_CYCLADES=y` flag in the `/usr/src/linux/.config` file, the kernel was then compiled and installed on the serial console servers. All of the clients are running kernels compiled with `CONFIG_SERIAL=y` and `CONFIG_SERIAL_CONSOLE=y`. RJ45 to DB9 converters were constructed for each node, and the wiring was done using CAT 5 patch cables. To activate the serial console on each client, changes were made to the `/etc/lilo.conf`. Each image section needed the following line:

```
append="console=ttyS0 console=tty0"
```

This tells the kernel to use both `ttyS0`, the first serial port, and `tty0`, the normal KVM console as consoles. Also, the following line was added to the general lilo options at the top of the `/etc/lilo.conf` file to allow lilo to interact over the serial console:

```
serial=0,9600n8
```

To conveniently access these serial consoles, the `conserver` software is utilized. Currently Colony is running `Conserver-7.4`. `Conserver` provides logging of the serial console output at the servers and a simple console command that can be directed through any server to the serial console of any node connected to that server.

5 High Speed Interconnect

To compete with the commercial HPC systems, a high-bandwidth, low-latency communications system was needed between the nodes on Colony. While some cluster applications do not require high performance communications, the scientific modelling that Colony was built for does.

Colony was purchased with a Giganet cLAN network as well as a 100Mb Ethernet. This allows the Ethernet to be used for systems administration, system installs, etc. without impacting the network resources available to user jobs. The 100Mb Ethernet also gives us a network that is more reliable than the Giganet cLAN network.

Giganet cLAN is a proprietary hardware implementation of the Virtual Interface Architecture (VIA). Giganet was marketing this technology as an alternative to Myrinet as a high-speed cluster network. Giganet sold cLAN networks in fat-tree topologies of up to 128 ports. Giganet has since been acquired by Emulex, who is now marketing cLAN as a high-speed storage network.

The cLAN network provides very fast VIA communications. According to Emulex (wwwip.emulex.com/ip/products/clan5000.html) 2.5 Gb/s bi-directional with port-to-port latencies of $.5\mu\text{s}$ seconds. However, actual performance tests

done at PNNL show round trip latencies of 7 μ s with unidirectional throughputs of 100MBytes/s.

There is also TCP/IP LAN Emulation over VIA for the cLAN network. Although this LAN emulation works for most standard TCP/IP programs, broadcast traffic is not supported. While this allows most programs to run over the cLAN network, it greatly increases the latency because the data has to traverse the TCP/IP stack within the OS. Therefore, using TCP/IP we end up with performance similar to GigE – high-bandwidth, high-latency.

6 MPI Parallel API

User applications also need to manage parallel communications over this high speed interconnect. This is done using the Message Passing Interface (MPI), a standard API for message passing in parallel programs. There are free distributions of MPI, but none were available that provided MPI over VIA when Colony was installed. Therefore, MPI-Pro from MPI SoftTech was purchased. This commercial version of MPI does MPI over the Gigaset cLAN VIA layer. This provides user applications with low-latency, high-bandwidth MPI communications.

7 Scheduling and Resource Management

Software has to be developed to enable users to run their jobs in a batch, shared resource mode. For example the software must prevent users from running jobs on top of each other, as this causes contention for resources that ruins performance and can crash nodes. The classic solution to this is a Batch Queuing system. Typical large parallel jobs do not require user interaction and users rarely require immediate job turn-around times. Therefore, a batch-processing model works quite nicely and is common to nearly all HPC systems.

At present, the leading batch system for Linux clusters is the Portable Batch System (PBS), available at <http://www.openpbs.org>. However, OpenPBS is not as complete as commercially available products like LoadLeveler for the IBM SP. One of the weaknesses of OpenPBS is the lack of a highly configurable scheduler. There are some basic schedulers that come with OpenPBS, but advanced scheduler capabilities are left out and must be programmed by the user. Our approach was to install the Maui Scheduler on Colony (<http://mauischeduler.sourceforge.net/>). This is the same scheduling software that is running on the IBM SP's in the MSCF, and therefore helped to provide a consistent environment to the users.

Before installing Maui, make sure that the PBS server, and PBS mom's are installed and working. The OpenPBS documentation provides good procedures to follow for this. Then follow the Maui-PBS Integration Guide found at

<http://supercluster.org/documentation/maui/pbsintegration.html>. Colony is running OpenPBS 2.2p11 and Maui Scheduler 3.0.3.8

It can be very useful to enable the QOS through the use of the addparam script. At PNNL this was necessary for metascheduling demonstrations.

To prevent some stability problems in PBS, we disabled the Reliable Packet Protocol (RPP) and installed the Sandia Fault Tolerance Patches (<http://www.cs.sandia.gov/cplant/doc/pbs/pbs.html>). Applying the Sandia Fault Tolerance Patches was fairly straightforward. However, disabling RPP turned out to be a bit tricky. At first it looked like passing `-disable-rpp` to the configure script would be enough. However, we found that this did not work. We modified the configure script by replacing every instance of `#define RPP 1` with `#define RPP 0`. After running configure again and recompiling, we had a version of OpenPBS that was not using RPP.

OpenPBS proved to be much more stable with RPP disabled and the Sandia Fault Tolerance Patches applied. Before performing these patches, the `pbs_moms`, and `pbs_server` would need to be recycled every few days to keep OpenPBS functional on Colony. After these patches were installed, it has been rare for the `pbs_moms` or `pbs_server` to need to be recycled.

8 Accounting

To control use of the cluster, it was important to not only account for the use of the system, but also to limit access to resource, based on rules defining the amount of resource that users are allowed to consume. To do this, the QBank CPU allocations bank was chosen. QBank was developed in the MSCF/EMSL at PNNL for use on the MSCF clusters. QBank offers the flexibility to allocate time to different accounts and offers management on a per user basis within these accounts. It allows the allocations to have an expiration time if desired. QBank also has commands to provide to the users the balance of time available to them and can be used to provide accounting reports for resource use. In addition to the allocation accounting, the allocations bank has been integrated with the scheduling system so that a job requesting more time than is available to them will be placed on hold until sufficient time for the job becomes available.

To make this work, as jobs start, the Maui scheduler will calculate the amount of time needed by the job from the number of nodes to be used and the wall clock limit of the job. It will then reserve that amount from the allocation account and start the job. When the job completes, the actual amount of time used will be deducted from the account and the reservation will be released.

That means that a job could be placed on hold because of insufficient time in the account, while a job from the same account, with a long wall clock limit is

running. If the running job completes earlier than the amount of time requested, more time will become available when the reservation is released. That will allow the job to be moved back into the queue to be considered for running.

Having time allocations that expire is also important. It helps when multiple groups require a specified fraction of the system for their use. The use of this portion of the system use can be granted and the allocation can be set to expire on a weekly, monthly, quarterly etc. basis. This helps at the end of the year, when several projects all want to use their years worth of time. They will already understand that some of their allocated time has expired.

For more examples and documentation see the web pages at:
<http://www.emsl.pnl.gov:2080/docs/mscf/qbank-2.8/overview.html>

9 Parallel Filesystems

Many of the jobs that run in a parallel environment can produce a large amount of output data. As the number of nodes used in a job grows, the amount of data that will need to be written increases. When using the NFS filesystems from a large number of client systems, the reads or writes to the same server become a bottleneck for the parallel jobs. On some of commercial systems, there are filesystems to improve the throughput of these file operations.

In looking at the filesystems for parallel systems, there are generally two types of filesystems that people want to use. In some cases, it is important that the filesystem offer high availability for use in projects where the data is critical, or needs to be highly available. In other cases, it is important that the filesystem offer high throughput from a large number of nodes writing at the same time.

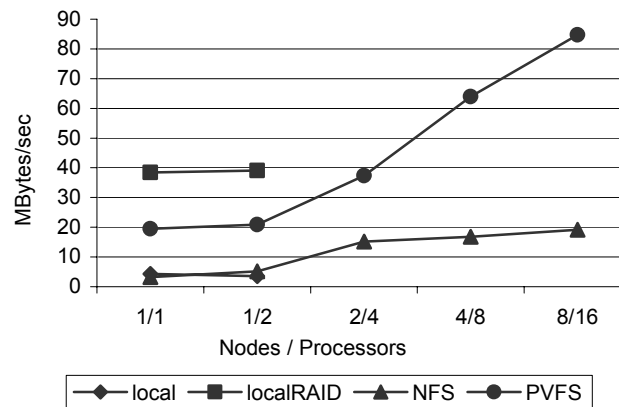


Figure 1: Disk throughput writing 1GB of data using large block writes.

For scientific applications to be run and tested on the Linux cluster, we wanted to provide a filesystem to be used for high throughput parallel jobs. The filesystem that we chose was the Parallel Virtual File System (PVFS). This filesystem was designed as a high throughput file system for parallel jobs. We chose to use three I/O servers and another server for the metadata. Each I/O server has 50GB of RAID0 disk and are connected to both the 100Mb Ethernet and to the Gigaset cLAN. As shown in Figure 1, the performance of PVFS was much better than NFS for writing large files from multiple nodes at the same time. As the number of clients grows the difference between the throughput numbers also increases. The throughput to the filesystem becomes especially attractive when using the Gigaset cLAN as the file transport. In some cases writing to the PVFS filesystem can give better performance than writing to a local disk.

In addition to the performance improvement by using normal file I/O subroutines for writing files, even greater flexibility and performance can be obtained from using the custom PVFS I/O routines provided. For more information on PVFS see the web site at: <http://parlweb.parl.clemson.edu/pvfs/>

10 System Monitoring

One of the most important parts of running a “production” system is to have the administrators find the problems with the system before the users do. Thus, system monitoring is required, especially for those things that the users are going to notice. Generally Linux clusters do not have the highly developed monitoring tools common on commercial HPC systems. However there are several common tools that can be used to monitor many of the common problems. One important issue when using system monitoring, is the amount of resource used by the monitoring programs. If the system use or disruption by the monitoring programs is greater than the benefit given, it may be necessary to decrease the level of monitoring. On the other hand if system time is lost without knowing why, it may be important to increase the administrator's awareness of what is going on with the system.

10.1 Cron

As with all system administration tasks, *cron* is again one of the sys-admins greatest friends. *Cron* jobs are being used to monitor the log files from both the resource manager and the scheduling system. If either of these programs quit updating the log files for extended periods of time, they can be killed and restarted. This monitoring can also notify the administrators by email etc, so that they check log files etc. for information about the failures and fix the problems.

10.2 Instant messages

As an administrator, it is always nice to be able to ignore the system while working on something else. Remember that it is important to find problems before users find them. To accomplish this, an instant messaging system was used. Because of previous experience with the *zephyr* messaging system, it was installed and set up. This allows an administrator to be sent pop-up messages from the system when problems occur.

10.3 rsh/ssh, ping

The ping program and the rsh and ssh programs are also a standard tool for most system monitoring. In addition to using *ping* or *fping* to check that the network interface is up, it is fairly easy to write a simple script to go through each of the nodes and check the uptime information and ensure that the OS is functioning at a somewhat higher level than the ICMP level. Any of these should be able to catch problems with nodes not responding on the network.

10.4 Client/server programs

Another option is to use one of the client server packages that have been developed. The most basic thing to use is a simple server process that runs on one system and receives "checkin" messages from a client that runs on each node. This keeps the server from needing to poll each node, but can also be easily extended to have the nodes send some system status information rather than just a blank message when they check in.

One program that does this quite nicely is the Ganglia program. It is used in several cluster systems. It is basically a client process that runs on the nodes with a server to contact. Programs are also available to query the data provided to the server through a command-line interface and through a web interface.

10.5 Clanview

Monitoring tools for the cLAN network were provided with the Giganet cLAN. The most simple of these is clanview. Run on a node, clanview gives a view of the cLAN adapters and switchports that it can see, including information on any that it used to see, but doesn't anymore.

11 Applications and libraries

Software that has been used on this cluster includes:

- NWChem - NWChem is a computational chemistry package that is designed to run on high-performance parallel supercomputers as well as conventional workstation clusters. It aims to be scalable both in its

ability to treat large problems efficiently, and in its usage of available parallel computing resources.

(<http://www.emsl.pnl.gov:2080/docs/nwchem/nwchem.html>)

NWChem software has been also used on Colony through the Extensible Computational Chemistry Environment (Ecce).

(<http://www.emsl.pnl.gov:2080/docs/ecce/>)

- NWPhys - NorthWest Physics code is a 3D, full-physics, first principles, time-domain, free-Lagrange code for parallel processing using hybrid grids. It contains a collection of solvers for CFD, continuum mechanics and diffusion problems.

(<http://www.emsl.pnl.gov:2080/nwphys/>)

- ARMCI (Aggregate Remote Memory Copy Interface) - ARMCI provides general-purpose, portable and efficient remote memory copy operations (one-sided communication) optimized for noncontiguous (strided, scatter/gather, I/O vector) data transfers.

(<http://www.emsl.pnl.gov:2080/docs/parsoft/armci/index.html>)

- NWGrid - NorthWest Grid generator is a library of user-callable tools that provide hybrid mesh generation, mesh optimization, and dynamic mesh maintenance in three dimensions for a variety of multi-scale, multi-physics applications. Geometric regions within arbitrarily complicated geometries are defined as combinations of bounding surfaces, where the surfaces are described analytically or as collections of points in space. Geometries may be defined interactively, either in terms of intersections and unions of surfaces or via input from Computer Aided Design (CAD) output.

(<http://www.emsl.pnl.gov:2080/nwgrid/>)

12 Acknowledgements

This research was performed in the William R. Wiley Environmental Molecular Sciences Laboratory (EMSL) at PNNL. This research was performed in part using the Molecular Science Computing Facility (MSCF) in the EMSL. The EMSL is a national user facility funded by the Office of Biological and Environmental Research in the U.S. Department of Energy.

Many people have helped to make everything work. We especially acknowledge the following: From PNNL: Edo Apra, Eric Bylaska, Gary Black, David Dixon, Rob Eades, George Fann, Andrew Felmy, David Jackson, Scott Jackson, Jarek Nieplocha, Harold Trease and Ralph Wescott. From Dell: Jenwei Hsieh and Anders Snorteland. From Giganet: Stephen Hird and Carlos Alvarez. From MPI Software Technology: Anthony Skjellum. From the University of Utah: Brian Haymore.

13 References

- Open PBS Homepage, <http://www.openpbs.org>
- Giganet Homepage, <http://www.giganet.com>
- Baytech Homepage, <http://www.baytechcd.com>
- Cyclades Homepage, <http://www.cyclades.com>
- MPI Software Technologies Homepage, <http://www.mpi-softtech.com>
- Cfengine Homepage, <http://www.iu.hioslo.no/cfengine/>
- Conserver Homepage, <http://www.conserver.com>
- Supercluster.org Homepage, <http://www.supercluster.org>
- Maui/PBS integration Guide,
<http://supercluster.org/documentation/maui/pbsintegration.html>
- Maui Scheduler Homepage, <http://mauischeduler.sourceforge.net>
- Qbank 2.8 Homepage, <http://www.emsl.pnl.gov:2080/docs/mscf/qbank-2.8/>
- Molecular Sciences Computing Facility Homepage,
<http://www.emsl.pnl.gov:2080/mscf/>
- Linux Kernel Howto, <http://www.kernel.org/LDP/HOWTO/Kernel-HOWTO.html>
- Linux Serial Howto, <http://www.kernel.org/LDP/HOWTO/Serial-HOWTO.html>
- Herding Penguins: or Clusters for a Multi-User, Production Computing Environment, Robert Eades, Presented at Linux Supercluster Users Conference 2000.
<http://clumber.tc.cornell.edu/actc/Supercluster/abstracts/Production.html>
- PVFS Homepage, <http://parlweb.parl.clemson.edu/pvfs/>
- GFS Homepage, <http://www.sistina.com/gfs/>
- System Imager Homepage, <http://www.systemimager.org>
- Red Hat Linux Homepage, <http://www.redhat.com>