Reflections on Building a High-performance Computing Cluster Using FreeBSD

Brooks Davis, Michael AuYeung, J. Matt Clark, Craig Lee, James Palko, Mark Thomas
The Aerospace Corporation
El Segundo, CA
{brooks,mauyeung,mclark,lee,jpalko,mathomas}@aero.org

Abstract

Since late 2000 we have developed and maintained a general purpose technical and scientific computing cluster running the FreeBSD operating system. In that time we have grown from a cluster of 8 dual Intel Pentium III systems to our current mix of 64 dual Intel Xeon and 289 dual AMD Opteron systems. This paper looks back on the system architecture as documented in our BSDCon 2003 paper "Building a High-performance Computing Cluster Using FreeBSD" and our changes since that time. After a brief overview of the current cluster we revisit the architectural decisions in that paper and reflect on their long term success. We then discuss lessons learned in the process. Finally, we conclude with thoughts on future cluster expansion and designs.

1 Introduction

From the early 1990's on, the primary thrust of high performance computing (HPC) development has been in the direction of commodity clusters, commonly referred to as Beowulf clusters [Becker]. These clusters combine commercial off-the-shelf hardware to create systems which rival or exceed the performance of traditional supercomputers in many applications while costing as much as a factor of ten less. Not all applications are suitable for clusters, but a signification portion of interesting scientific applications can be successfully adapted to them.

In 2001, driven by a number of separate users with supercomputing needs, The Aerospace Corporation (a California nonprofit corporation that operates a Federally Funded Research and Development Center) decided to build a corporate computing cluster (eventually named Fellowship for The Fellowship of the Ring [Tolkien]) as an alternative to continuing to buy small clusters and SMP systems on an ad-hoc basis. This decision was motivated by a desire to use computing resources more efficiently as well as reducing administrative costs. The diverse set of user requirements in our environment led us to a design which differs significantly from most clusters we have seen elsewhere. This is especially true in the areas of operating system choice (FreeBSD) and configuration management (fully network booted nodes).

At BSDCon 2003 we presented a paper titled "Building a High-performance Computing Cluster Using FreeBSD" [Davis] detailing these design decisions. This paper looks back on the system architecture as documented in that paper and our changes since that time. After a brief overview of the current cluster we revisit the architectural decisions in that paper and reflect on their long term success. We then discuss lessons learned in the process. Finally, we conclude with thoughts on future cluster expansion and designs.

2 Fellowship Overview

The basic logical and physical layout of Fellowship is similar to many clusters. There are six core systems, 352 dual-processor nodes, a network switch, and assorted remote management hardware. All nodes and servers run FreeBSD, currently 6.2-RELEASE. The core systems and remote management hardware sit on the Aerospace corporate network. The nodes and core systems share a private, non-routed network (10.5/16). The majority of this equipment is mounted in two rows of seven-foot tall, two-post racks residing in the underground data center at Aerospace headquarters in El Segundo, California. Figure 1 shows Fellowship in Fall 2006. The layout of a recent node racks is shown in Figure 2. The individual racks vary due to design changes over time.

When users connect to Fellowship, they do so via a

©2007 The Aerospace Corporation.

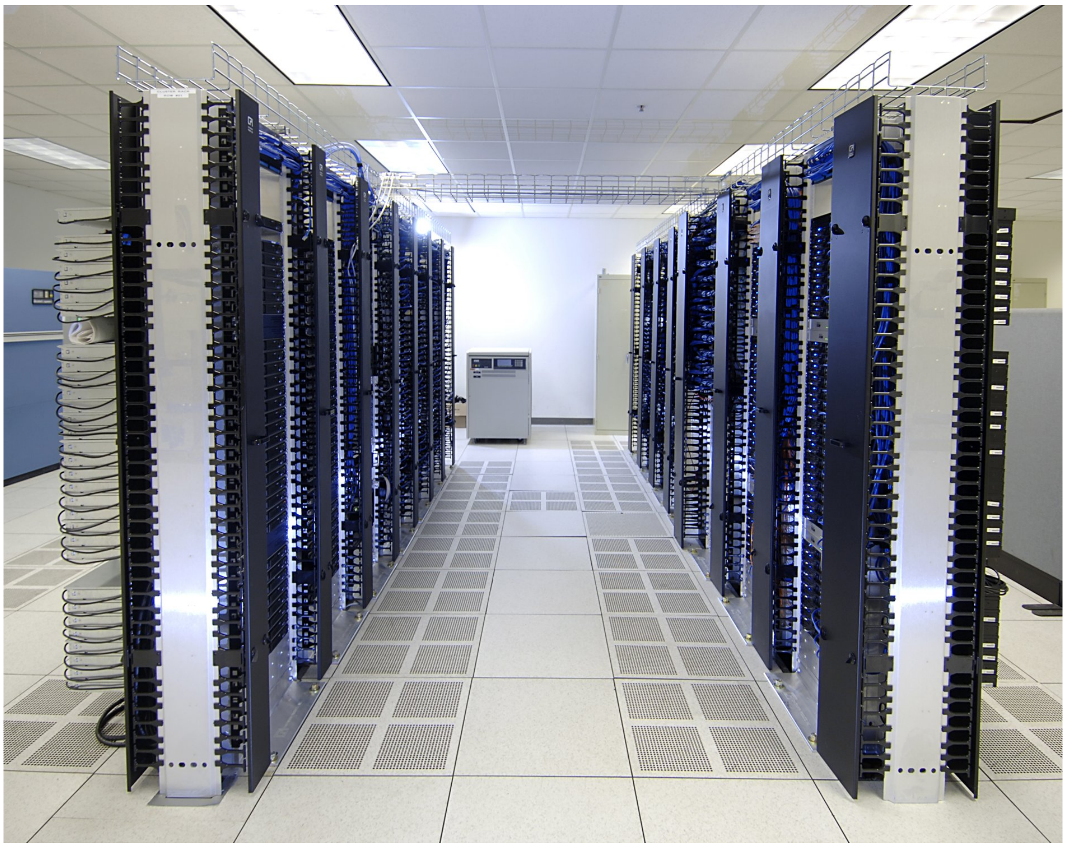

Figure 1: Fellowship Circa February 2007

core server named `fellowship` that is equipped to provide shell access. There they edit and compile their programs and submit jobs for execution on a node or set of nodes. The scheduler is run on the core server `arwen` that also provides network boot services to the nodes to centralize node management. Other core servers include: `frodo` which provides directory service for user accounts and hosts the license servers for commercial software including the Intel FORTRAN compiler and Grid Mathematica; `gamgee` which provides backups using the Bacula software and a 21 tape LTO2 changer; `elrond` and `legolas` which host shared temporary file storage that is fast and large respectively; and `moria`, our Network Appliance file server.

The nodes are currently a mix of older Intel Xeons and single and dual-core AMD Opterons. Table 1 gives a breakdown of general CPU types in Fellowship today. Figure 4 and Figure 5 show the composition of Fellowship over time by node and core count. Each node as an internal ATA or SATA disk that is either 80GB or 250GB and between 1 and 4 gigabytes of RAM. The nodes are connected via Gigabit Ethernet through a Cisco Catalyst 6509 switch[1] The

CPU Type	*Nodes*	*CPUs*	*Cores*
Xeon	64	128	128
Opteron single-core	136	272	272
Opteron dual-core	152	304	608
Total	352	704	1008

Table 1: CPUs in Fellowship nodes.

Opterons are mounted in custom 1U chassis approximately 18 inches deep with IO ports and disks facing the front of the rack. Figure 3 shows the front of first generation Opteron nodes.

Although the nodes have disks, we network boot them using PXE support on their network interfaces with `frodo` providing DHCP, TFTP, NFS root disk, and NIS user accounts. On boot, the disks are automatically checked to verify that they are properly partitioned for our environment. If they are not, they are automatically repartitioned. This means minimal configuration of nodes is required beyond determining their MAC address and location. Most of that configuration is accomplished by scripts.

Local and remote control of core machines is made

[1]This was originally a Catalyst 6513, but most slots in the 6513 have reduced available bandwidth so we upgraded to the smaller 6509.

Unit	*Contents*
45	Patch panel
44	(Connection to patch panel rack)
43	*empty*
42	*empty*
41	*empty*
40	node (r01n32: 10.5.1.32)
39	node (r01n31: 10.5.1.31)
38	node (r01n30: 10.5.1.30)
37	node (r01n29: 10.5.1.29)
36	node (r01n28: 10.5.1.28)
35	node (r01n27: 10.5.1.27)
34	node (r01n26: 10.5.1.26)
33	node (r01n25: 10.5.1.25)
32	Cyclades 10-port power controller
31	Cyclades 10-port power controller
30	node (r01n24: 10.5.1.24)
29	node (r01n23: 10.5.1.23)
28	node (r01n22: 10.5.1.22)
27	node (r01n21: 10.5.1.21)
26	node (r01n20: 10.5.1.20)
25	node (r01n19: 10.5.1.19)
24	node (r01n18: 10.5.1.18)
23	node (r01n17: 10.5.1.17)
22	rackmount KVM unit
21	node (r01n16: 10.5.1.16)
20	node (r01n15: 10.5.1.15)
19	node (r01n14: 10.5.1.14)
18	node (r01n13: 10.5.1.13)
17	node (r01n12: 10.5.1.12)
16	node (r01n11: 10.5.1.11)
15	node (r01n10: 10.5.1.10)
14	node (r01n09: 10.5.1.9)
13	Cyclades PM 10-port power controller
12	Cyclades PM 10-port power controller
11	node (r01n08: 10.5.1.8)
10	node (r01n07: 10.5.1.7)
9	node (r01n06: 10.5.1.6)
8	node (r01n05: 10.5.1.5)
7	node (r01n04: 10.5.1.4)
6	node (r01n03: 10.5.1.3)
5	node (r01n02: 10.5.1.2)
4	node (r01n01: 10.5.1.1)
3	
2	4 120V 30A & 1 120 V 20A Circuits
1	

Figure 2: Layout of Node Rack 1

possible through a Lantronix networked KVM-switch connected to a 1U rackmount keyboard, and track pad and an 18-inch rackmount monitor which doubles as a local status display. In the newer racks, 1U rack mount keyboard, monitor, mouse (KVM) units are installed to allow administrators to easily access local consoles during maintenance. In addition to console access, everything except the terminal servers and the switch are connected to serial remote power controllers. The older racks use a variety of 8-port BayTech power controllers and the new ones use 10-port Cyclades AlterPath PM series controllers. All

Figure 3: Front details of Opteron nodes

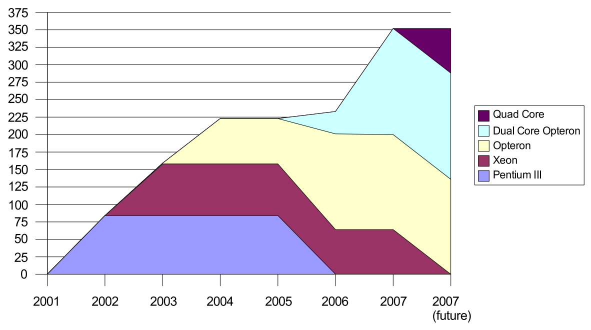

Figure 4: Fellowship node count and type over time

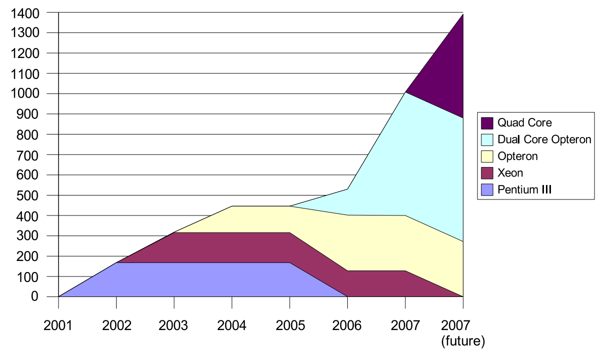

Figure 5: Fellowship core count and type over time

of these controllers are capable of supplying a total of 30 Amps of power at 120 Volts. This allows us to remotely reboot virtually any part of the system by connecting to the power controller via the appropriate terminal server.

On top of this infrastructure, access to nodes is controlled by Sun Grid Engine (SGE), a scheduler implementing a superset of the POSIX Batch Environment Services specification. SGE allows users to submit both interactive and batch job scripts to be run on one or more processors. Users are free to use the processors they are allocated in any reasonable manner. They can run multiple unrelated processes or massively parallel jobs.

To facilitate use of Fellowship, we provide a basic Unix programming environment, plus the parallel programming toolkits, and commercial parallel applications. For parallel programming toolkits we provide MPICH, MPICH2, and OpenMPI implementations of the Message Passing Interface [MPI] (MPI) as well as the Parallel Virtual Machine (PVM). We also provide Grid Mathematica and MATLAB under Linux emulation.

3 Design Issues

One of the biggest challenges in building Fellowship was our diverse user base. Among the users at the initial meetings to discuss cluster architecture, we had users with loosely coupled and tightly coupled applications, data intensive and non-data intensive applications, and users doing work ranging from daily production runs to high performance computing research. This diversity of users and applications led to the compromise that was our initial design. Many aspects of this design remain the same, but some have changed based on our experiences. In this section we highlight the major design decisions we made while building Fellowship and discuss how those decisions have fared in the face of reality.

3.1 Operating System

The first major design decision any cluster designer faces is usually the choice of operating system. By far, the most popular choice is a Linux distribution of some sort. Indeed, Linux occupies much the same position in the HPC community as Windows does in the desktop market to the point most people assume that, if it is a cluster, it runs Linux. In reality a cluster can run almost any operating system. Clusters exist running Solaris [SciClone], MacOS X, FreeBSD, and Windows[WindowsCCS] among others. NASA's Columbia [Columba] super computer is actually a cluster of 24 SGI Altix systems 21 of which are 512-CPU system.

For an organization with no operating system bias and straight-forward computing requirements, running Linux is the path of least resistance due to free clustering tool kits such as Rocks [ROCKS] or OSCAR [OSCAR]. In other situations, operating system choice is more complicated. Important factors to consider include chosen hardware platform, existence of experienced local system administration staff, availability of needed applications, ease of maintenance, system performance, and the importance of the ability to modify the operating system.

For a variety of reasons, we chose FreeBSD for Fellowship. The most pragmatic reason for doing so is the excellent out of the box support for diskless systems which was easily modifiable to support our nodes network booting model. This has worked out very well.

Additionally, the chief Fellowship architect uses FreeBSD almost exclusively and is a FreeBSD committer. This meant we had more FreeBSD experience than Linux experience and that we could push some of our more general changes back into FreeBSD to simplify operating system upgrades. We have been able to feed back a number of small improvements, particularly in the diskless boot process which has benefited us and other FreeBSD users. For changes which are too Aerospace- or Fellowship-specific to contribute back, we have maintained an internal Perforce repository with a customized version of FreeBSD.

The ports collection was also a major advantage of using FreeBSD. It has allowed us to install and maintain user-requested software quickly and easily. For most applications the ports collection has worked well. The major exception is anything related to MPI. MPI implementations are generally API compatible, but use different symbols, libraries, and header files. As a result MPI implementations traditionally provide a set of wrapper scripts for the compiler along the lines of `mpicc`, `mpic++`, `mpif77`, etc which take care of the details of linking. Unfortunately a different compilation of MPI is needed for each compiler and it is useful to have multiple MPI versions. The ports framework is not equipped to deal with such combinatorial explosions.

The availability of Linux emulation meant we did not give up much in the way of application compatibility. We were the launch customer for Grid Mathematica using the Linux version. We have also run other third-party Linux programs including MPI applications.

Initially the disadvantages of FreeBSD for our purposes were immature SMP and threading support, and an widely held view within the high performance computing community that if it isn't a commercial supercomputer, it must be a Linux system. SMP support was not a major issue for our users because most of our jobs are compute-bound so the poor SMP performance under heavy IO was a moot problem. With the 5.x and 6.x series of FreeBSD releases, this issue has largely been resolved. Threading was more of an issue. We had users who wanted to use threads to support SMP scaling in certain applications and with the old user space threading provided by `libc_r` they could not do that. With our migration to the FreeBSD 6.x series, this problem has been resolved.

The Linux focus of the HPC community has caused us some problems. In particular, many pieces of software either lack a FreeBSD port, or only have a poorly tested one which does not actually work out of the box. In general we have been able to complete ports for FreeBSD without issue. One significant exception was the LAM implementation of MPI. In 4.x it worked fine, but in 5.x and 6.x it builds but crashes instead of running. We have been unable to find and fix the problems. Initially there was a shortage of compiler support for modern versions of FORTRAN. This has been changed by two things: the gfortran project's FORTRAN 90 and 95 compiler and the wrapping of the Intel Linux FORTRAN compiler to build FreeBSD software. A FreeBSD FORTRAN compiler is also available from NAG. One other issue is the lack of a parallel debugger such as TotalView.

We are happy with the results of running FreeBSD on the cluster. It has worked well in for us and the occasional lack of software had been more than made up for by our existing experience with FreeBSD.

3.2 Hardware Architecture

The choice of hardware architecture is generally made in conjunction with the operating system as the two interact with each other. Today, most clusters are based on Intel or AMD x86 CPUs, but other choices are available. When developing Fellowship, SPARC and Alpha clusters were fairly command as were Apple XServe clusters based on the PowerPC platform. Today, the Alpha is essentially gone and Apple has migrated from PowerPC to Intel CPUs leaving x86 with the vast majority of the HPC cluster market. The major issues to consider are price, performance, power consumption, and operating system compatibility. For instance, Intel's Itanium2 has excellent floating point performance, but is expensive and power hungry. Early on it also suffered form immature operating system support. In general, x86 based systems are the path of least resistance given the lack of a conflicting operating system requirement.

When we were selecting a hardware architecture in 2001, the major contenders were Alpha and Intel or AMD based x86 systems. We quickly discarded Alpha from consideration because of previous experiences with overheating problems on a small Aerospace Alpha cluster. Alphas also no longer had the kind of performance lead they enjoyed in the late 1990's. We looked at both Pentium III and Athlon-based systems, but decided that while the performance characteristics and prices did not vary significantly, power consumption was too problematic on the Athlon systems.

Over the life of Fellowship, we have investigated other types of nodes including Athlon based systems, the Xeon systems we purchased for the 2003 expansion, AMD Opteron systems, and Intel Woodcrest systems. Athlons failed to match the power/performance ratios of Intel Pentium III systems, but with the Xeon and Opteron processors the balance shifted resulting in purchases of Opterons in 2004 through 2006. We are now evaluating both AMD and Intel based solutions for future purchase. One interesting thing we've found is that while the Intel CPUs themselves consume less power, the RAM and chipsets they use are substantially more power hungry. This illustrates the need to look a all aspects of the system not just the CPUs.

3.3 Node Architecture

Most of the decisions about node hardware will derive from the selection of hardware architecture, cluster form factor, and network interface. The biggest of the remaining choices is single or multi-processor systems. Single processor systems have better CPU utilization due to a lack of contention for RAM, disk, and network access. Multi-processor systems can allow hybrid applications to share data directly, decreasing their communication overhead. Additionally, multi-processor systems tend to have higher performance interfaces and internal buses then single processor systems. This was significant consideration with Fellowship's initial design, but the current direction of processor development suggest that only multiple core CPUs will exist in the near future.

Other choices are processor speed, RAM, and disk space. We have found that aiming for the knee of the price curve has served us well, since no single user dominates our decisions. In other environments, top of the line processors, large disks, or large amounts of RAM may be justified despite the exponential increase

CPU	2 x Pentium III 1GHz
Network Interface	3Com 3C996B-T
RAM	1GB
Disk	40GB 7200RPM IDE

Table 2: Configuration of first Fellowship nodes.

CPU	2 x Opteron 275 2x2.2GHz
Network Interface	On board gigabit
RAM	4GB
Disk	250GB 7200RPM SATA

Table 3: Configuration of latest Fellowship nodes.

in cost.

For Fellowship, we chose dual CPU systems. We were motivated by a desire to do research on code that takes advantage of SMP systems in a cluster, higher density than single processor systems, and the fact that the 64-bit PCI slots we needed for Gigabit Ethernet were not available on single CPU systems. As a result of our focus on the knee of the price curve, we have bought slightly below the performance peak on processor speed, with 2-4 sticks of smaller-than-maximum RAM, and disks in the same size range as mid-range desktops. This resulted in the initial configuration shown in Table 2. The most recent node configuration is shown in Table 3.

3.4 Network Interconnects

Like hardware architecture, the selection of network interfaces is a matter of choosing the appropriate point in the trade space between price and performance. Performance is generally characterized by bandwidth and latency. The right interface for a given cluster depends significantly on the jobs it will run. For loosely-coupled jobs with small input and output data sets, little bandwidth is required and 100Mbps Ethernet is the obvious choice. For other, tightly-coupled jobs, InfiniBand or 10 Gbps Myrinet which have low latency and high bandwidth are good options For some applications 1 Gbps or 10 Gbps Ethernet will be the right choice.

The choice of Gigabit Ethernet for Fellowship's interconnect represents a compromise between the cheaper 100 Mbps Ethernet our loosely coupled applications would prefer (allowing us to buy more nodes) and 2Gbps Myrinet. When we started building Fellowship, Gigabit Ethernet was about one-third of the cost of each node whereas Myrinet would have more than doubled our costs. Today Gigabit Ethernet is standard on the motherboard and with the large switches required by a cluster Fellowship's size, there is no price difference between 100 Mbps and 1 Gbps ether ports.

Gigabit Ethernet has worked well for most of our applications. Even our computational fluid dynamics (CFD) applications have run reasonably well on the system. A few applications such as the CFD codes and some radiation damage codes would benefit from higher bandwidth and lower latency, but Gigabit Ethernet appears to have been the right choice at the time.

3.5 Core Servers and Services

On Fellowship, we refer to all the equipment other then the nodes and the remote administration hardware as core servers. On many clusters, a single core server suffices to provide all necessary core services. In fact, some clusters simply pick a node to be the nominal head of the cluster. Some large clusters provide multiple front ends, with load balancing and fail over support to improve up time.

Core services are those services which need to be available for users to utilize the cluster. At a minimum, users need accounts and home directories. They also need a way to configure their jobs and get them to the nodes. The usual way to provide these services is to provide shared home and application directories, usually via NFS and use a directory service such as NIS to distribute account information. Other core services a cluster architect might choose to include are batch schedulers, databases for results storage, and access to archival storage resources. The number of ways to allocate core servers to core services is practically unlimited.

Fellowship started with three core servers: the data server, the user server, and the management server. All of these servers are were dual 1GHz Pentium III systems with SCSI RAID5 arrays. The data server, `gamgee`, served a 250GB shared scratch volume via NFS, and performed nightly backups to a 20 tape LTO library using AMANDA. The user server, `fellowship`, served NFS home directories and gave the users a place to log in to compile and run applications. The management server, `frodo`, ran the scheduler, NIS, and our shared application hierarchy mounted at `/usr/aero`. Additionally, the management server uses DHCP, TFTP, and NFS to netboot the nodes.

Today, Fellowship has eight core servers. The data server has been split into three machines: `elrond`, `gamgee`, and `legolas`. `elrond` is a 2.4GHz dual Xeon with SCSI raid that provides `/scratch`. `gamgee` it self has been replaced with a dual Opteron with 1TB of SATA disk. It runs backups using Bacula a network based backup program. `legolas` provides 2.8GB of shared NFS disk at `/bigdisk`. The user server, `fellowship`, is now a quad Opteron system and home directories are served by `moria`, a Network Appliance FAS250 filer. It is supplemented by a second quad Opteron, `fellowship-64` which runs FreeBSD amd64. The original management servers, `frodo`, still exists, but currently only runs NIS. It has mostly been replaced by `arwen` which has taken over running the Sun Grid Engine scheduler and netbooting the nodes.

These services were initially isolated from each other for performance reasons. The idea was that hitting the shared scratch space would not slow down ordinary compiles and compiling would not slow down scratch space access. We discovered that, separation of services does work, but it comes at the cost of increased fragility because the systems are interdependent, and when one fails, they all have problems. We have devised solutions to these problems, but this sort of division of services should be carefully planned and would generally benefit from redundancy when feasible. Our current set of servers is larger than optimal from an administrative perspective. This situation arose because new core servers were purchased opportunistically when end of year funds were available and thus older servers have not been gracefully retired.

3.6 Node Configuration Management

Since nodes outnumber everything else on the system, efficient configuration management is essential. Many systems install an operating system on each node and configure the node-specific portion of the installation manually. Other systems network boot the nodes using Etherboot, PXE or LinuxBIOS. The key is good use of centralization and automation. We have seen many clusters where the nodes are never updated without dire need because the architect made poor choices that made upgrading nodes impractical.

Node configuration management is probably the most unique part of Fellowship's architecture. We start with the basic FreeBSD diskless boot process [diskless(8)]. We then use the diskless remount support to mount `/etc` as `/conf/base/etc`. For many applications, this configuration would be sufficient. However, we have applications which require significant amounts of local scratch space. To deal with this each node contains a disk. The usual way of handling such disks would be to manually create appropriate directory structures on the disk when the system was first installed and then let the nodes mount and fsck the disks each time they were booted. We deemed this impractical because nodes are usually installed in large groups. Additionally, we wanted the ability to reconfigure the disk along with the operating system.

In our original FreeBSD 4.x based installation, we created a program (`diskmark`) which used an invalid entry in the MBR partition table to store a magic number and version representing the current partitioning scheme. At boot we used a script which executed before the body of `rc.diskless2` to examine this entry to see if the current layout of the disk was the required one. If it was not, the diskless scripts automatically use Warner Losh's `diskprep` script to initialize the disk according to our requirements. With FreeBSD 6.x we adopted a somewhat less invasive approach. We still use a version of `diskprep` to format the disk, but now we use `glabel` volume labels to identify the version of the disk layout. The script that performs the partitioning is installed in `/etc/rc.d` instead of requiring modification of `/etc/rc`. We install the script in the node image using a port.

With this configuration, adding nodes is very easy. The basic procedure is to bolt them into the rack, hook them up, and turn them on. We then obtain their MAC address from the switch's management console and add it to the DHCP configuration so each node is assigned a well-known IP address. After running a script to tell the scheduler and Nagios about the nodes and rebooting them, they are ready for use.

Under FreeBSD 4.x, maintenance of the netboot image is handed by chrooting to the root of the installation and following standard procedures to upgrade the operating system and ports as needed. With our move to FreeBSD 6.x we have also moved to a new model of netboot image updating. Instead of upgrading a copy, we create a whole new image from scratch using parts of the nanobsd [Gerzo] framework. The motivation for switching to this mode was that in the four years since we got the initial image working we had forgotten all the customizations that were required to make it fully functional. Since 6.x has an large number of differences in configuration from 4.x, this made it difficult to upgrade. Our theory with creating new images each time is that it will force us to document all the customizations either in the script or in a separate document that will be manually modified. Thus far, this has worked to some degree, but has not been perfect as creation of some of the images has been rushed resulting in a failure to document everything. In the long term, we expect it to be a better solution than in place upgrades. Software which is not available from the ports collection is installed in the separate `/usr/aero` hierarchy.

One problem we found with early systems was poor

Mountpoint	*Source*
`/`	`arwen:/export/roots/freebsd/fbsd62`
`/etc`	`/dev/md0`
`/tmp`	`/dev/ufs/tmp`
`/var`	`/dev/ufs/var`
`/home`	`moria:/home`
`/usr/aero`	`frodo:/nodedata/usr.aero`
`/usr/local/sge/fellowship`	`arwen:/export/sge/fellowship`
`/scratch`	`elrond:/scratch`
`/bigdisk`	`legolas:/bigdisk`

Table 4: Sample node (`r01n01` aka 10.5.1.1) mount structure

quality PXE implementations. We have found PXE to be somewhat unreliable on nearly all platforms, particularly on the Pentium III systems, occasionally failing to boot from the network for no apparent reason and then falling back to the disk which is not configured to boot. Some of these problems appear to be caused by interactions with network switches and the spanning tree algorithm. To work around this problem we have created a `diskprep` configuration that creates an extra partition containing FreeDOS and an `AUTOEXEC.BAT` that automatically reboots the machine if PXE fails rather than hanging. It would be better if server motherboard vendors added an option to the BIOS to keep trying in the event of a PXE failure.

3.7 Job Scheduling

Job scheduling is potentially one of the most complex and contentious issues faced by a cluster architect. The major scheduling options are running without any scheduling, manual scheduling, batch queuing, and domain specific scheduling.

In small environments with users who have compatible goals, not having a scheduler and just letting users run what they want when they want or communicating with each other out of band to reserve resources as necessary can be a good solution. It has very little administrative overhead, and in many cases, it just works.

With large clusters, some form of scheduling is usually required. Even if users do not have conflicting goals, it's difficult to try to figure out which nodes to run on when there are tens or hundreds available. Additionally, many clusters have multiple purposes that must be balanced. In many environments, a batch queuing system is the answer. A number exist, including OpenPBS, PBSPro, Sun Grid Engine (SGE), LSF, Torque, NQS, and DQS. Torque and SGE are freely available open source applications and are the most popular options for cluster scheduling. When we started building Fellowship OpenPBS was quite popular and SGE was not yet open source.

For some applications, batch queuing is not a good solution. This is usually either because the application requires too many jobs for most batch queuing systems to keep up, or because the run time of jobs is too variable to be useful. For instance, we have heard of one computational biology application which runs through tens of thousands of test cases a day where most take a few seconds, but some may take minutes, hours, or days to complete. In these situations, a domain specific scheduler is often necessary. A common solution is to store cases in a database and have applications on each node that query the database for a work unit, process it, store the result in the database, and repeat.

On Fellowship, we have a wide mix of applications ranging from trivially schedulable tasks to applications with unknown run times. Our current strategy is to implement batch queuing with a long-term goal of discovering a way to handle very long running applications. We initially intended to run the popular OpenPBS scheduler because it already had a port to FreeBSD and it is open source. Unfortunately, we found that OpenPBS had major stability problems under FreeBSD (and, by many accounts, most other operating systems)[2]. About the time we were ready to give up on OpenPBS, Sun released SGE as open source. FreeBSD was not supported initially, but we were able to successfully complete a port based on some patches posted to the mailing lists. That initial port allowed jobs to run. Since then we have added more functionality and the FreeBSD port is essentially at feature parity with the Linux port.

The major problem we had with scheduling is that initially, we allowed direct access to cluster nodes without the scheduler. While we had few users and not

[2]The Torque resource manager is a successful fork of OpenPBS to support the Maui scheduler

all systems were full at all times, this was not a big deal. Unfortunately, as the system filled up, it became a problem. We had assumed that users would see the benefits of using the scheduler such as unattended operation and allocation of uncontested resources, but most did not. Worse, those few who did often found themselves unable to access any resources because the scheduler saw that all the nodes were overloaded. Those users then gave up on the scheduler. We eventually imposed mandatory use of the scheduler along with a gradual transition to FreeBSD 6.x on the nodes. There was a fair bit of user rebellion when this happened, but we were able to force them to cooperate eventually. In retrospect failure to mandate use of the scheduler as soon as it was operational was a significant error.

3.8 Security Considerations

For most clusters, we feel that treating the cluster as a single system is the most practical approach to security. Thus for nodes which are not routed to the Internet like those on Fellowship, all exploits on nodes should be considered local. What this means to a given cluster's security policy is a local issue. For systems with routed nodes, management gets more complicated, since each node becomes a source of potential remote vulnerability. In this case it may be necessary to take action to protect successful attacks on nodes from being leveraged into full system access. In such situations, encouraging the use of encrypted protocols within the cluster may be desirable, but the performance impact should be kept firmly in mind.

The major exception to this situation is clusters where jobs have access to data that must not be mingled. We have begun an investigation into ways to isolate jobs from each other more effectively. We believe that doing so will yield both improved security and better performance predictability.

For the most part we have chosen to concentrate on protecting Fellowship from the network at large. This primarily consists of keeping the core systems up to date and requiring that all communications be via encrypted protocols such as SSH and HTTPS. Internally we discourage direct connections between nodes except by scheduler-provided mechanisms that could easily be encrypted. Inter-node communications are unencrypted for performance reasons.

3.9 System Monitoring

The smooth operation of a cluster can be aided by proper use of system monitoring tools. Most common monitoring tools such as Nagios and Big Sister are applicable to cluster use. The one kind of monitoring tool that does not work well with clusters is the sort that sends regular e-mail reports for each node. Even a few nodes will generate more reports then most admins have time to read. In addition to standard monitoring tools, there exist cluster specific tools such as the Ganglia Cluster Monitor. Most schedulers also contain monitoring functionality.

On Fellowship we are currently running the Ganglia Cluster Monitoring system, Nagios, and the standard FreeBSD periodic scripts on core systems. Ganglia was ported to FreeBSD previously, but we have created FreeBSD ports which make it easier to install and make its installation more BSD-like. We have also rewritten most of the FreeBSD specific code so that it is effectively at feature parity with Linux (and better in some cases). A major advantage of Ganglia is that no configuration is required to add nodes. They are automatically discovered via multicast. We have also deployed Nagios with a number of standard and custom scripts. With Nagios notification we typically see problems with nodes before our users do.

3.10 Physical System Management

At some point in time, every system administrator finds that they need to access the console of a machine or power cycle it. With just a few machines, installing monitors on each machine or installing a KVM switch for all machines and flipping power switches manually is a reasonable option. For a large cluster such as Fellowship, more sophisticated remote management systems are desirable.

In Fellowship's architecture, we place a strong emphasis on remote management. The cluster is housed in our controlled access data center, which makes physical access cumbersome. Additionally, the chief architect and administrator lives around 1500 miles (about 2400 kilometers) from the data center, making direct access even more difficult. As a result, we have installed remote power controllers on all nodes are core systems and remote KVM access to all core systems. Initially we had also configured all nodes to have serial consoles accessible through terminal servers. This worked well for FreeBSD, but we had problems with hangs in the BIOS redirection on the Pentium III systems which forced us to disable it. That combined with the fact that we rarely used the feature and the

$100 per port cost lead us to discontinue the purchase of per-rack terminal servers when we built the second row of racks. One recent change we have made is adding a per-rack 1U KVM unit in newer node racks. At around $650/rack they are quite cost effective and should save significant administrator time when diagnosing failures.

In the future we would like to look at using IPMI to provide remote console and reboot for nodes eliminating the need for dedicated hardware.

3.11 Form Factor

The choice of system form factor is generally a choice between desktop systems on shelves, rack mounted servers, and blade systems. Shelves of desktops are common for small clusters as they are usually cheaper and less likely to have cooling problems. Their disadvantages include the fact that they take up more space, the lack of cable management leading to more difficult maintenance, and generally poor aesthetics. Additionally, most such systems violate seismic safety regulations.

Rack mounted systems are typically slightly expensive due to components which are produced in lower volumes as well as higher margins in the server market. Additionally, racks or cabinets cost more then cheap metal shelves. In return for this added expense, rack mount systems deliver higher density, integrated cable management, and, usually, improved aesthetics.

Blade systems are the most expensive by far. They offer higher density, easier maintenance, and a neater look. The highest density options are often over twice as expensive with significantly lower peak performance due to the use of special low-power components.

A minor sub-issue related to rack mount systems is cabinets vs. open, telco style racks. Cabinets look more polished and can theoretically be moved around. Their disadvantages are increased cost, lack of space making them hard to work in, and being prone to overheating due to restricted airflow. Telco racks do not look as neat and are generally bolted to the floor, but they allow easy access to cables and unrestricted airflow. In our case, we use vertical cable management with doors which makes Fellowship look fairly neat without requiring cabinets.

The projected size of Fellowship drove us to a rack mount configuration immediately. We planned from the start to eventually have at least 300 CPUs, which is pushing reasonable bounds with shelves. We had a few problems with our initial rack confirmation. First, the use of six inch wide vertical cable management did not leave use with enough space to work easily. We used ten inch wide vertical cable management when we expanded to a second row of racks to address this problem. Second, the choice of making direct runs from nodes to the switch resulted in too much cable running to the switch. When we expanded to a second row of racks we added patch panels to them and the existing rack and moved the switch next to a new rack of patch panels. This substantially simplified our cabling. The patch switch and central patch panel can be seen in Figure 6. The third problem we encountered was that we were unable to mount some core systems in the racks we allocated for the core systems. We have mounted some of our core systems in a separate cabinet as a result and plan to add a dedicated cabinet in the future.

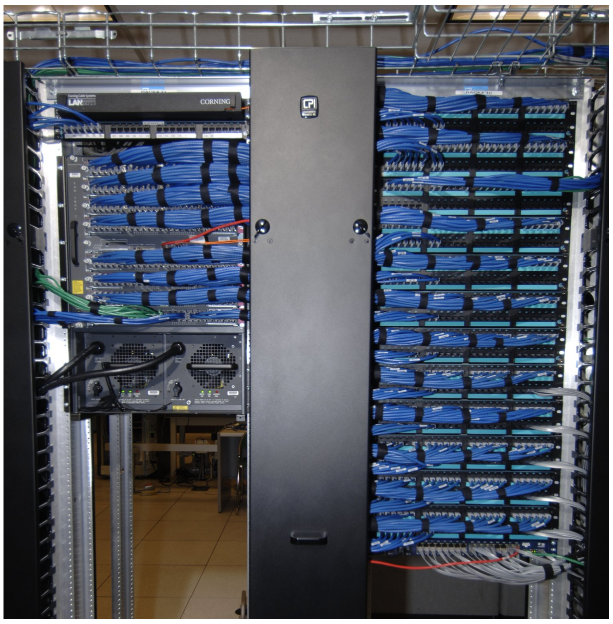

Figure 6: Fellowship's switch and patch panel racks

4 Lessons Learned

We have learned several lessons in the process of building and maintaining Fellowship. None took us completely by surprise, but they are worth covering as they can and should influence design decisions.

The first and foremost lesson we have learned is that with a cluster, relatively uncommon events can become common. For example, during initial testing with the Pentium III nodes we infrequently encountered two BIOS related problems: if BIOS serial port redirection was enabled, they system would occasionally hang and PXE booting would sometimes fail.

With the console redirection, we thought we had fixed the problem by reducing the serial ports speed to 9600 bps, but in fact we had just made it occur during approximately one boot in 30. This meant that every time we rebooted, we had to wait until it appeared everything had booted and then power cycle the nodes that didn't boot. In the end we were forced to connect a keyboard and monitor and disable this feature. Similarly, PXE problems did not appear serious and appeared resolved with one node, but with 40 nodes, they became a significant headache. In the end we implemented the reboot hack described in the Node Configuration Management section. In addition to these BIOS failures, we initially experienced hardware failures, most power supplies, at nearly every power cycle. This seemed high at first, but the problems mostly settled out over time. With a single machine this wouldn't have been noticeable, but with many machine it became readily apparent that the power supplies were poorly manufactured. Later on this was reinforced as at around three years of operation the nodes stared failing to POST. We eventually concluded the problem was with due to incremental degradation in the power supplies because the boards worked with a new supply. After the power supplies, the next most common component to fail has been the hard drives. In the Pentium III nodes they were the notorious IBM Deathstar disks which lead to a high failure rate. In other system the failure rate has been lower, but still significant. When we created specifications for the the custom cases used for the Opterons, we specified removable disks. This has simplified maintenance significantly.

A lesson derived from those hardware failures was that neatness counts quite a bit in racking nodes. To save money in the initial deployment, we ran cables directly from the switch to the nodes. This means we have a lot of slack cable in the cable management, which makes removing and reinstalling nodes difficult. We ended up adding patch panels in each rack to address this problem. Based on this experience we have considered blades more seriously for future clusters, particularly those where on site support will be limited. The ability to remove a blade and install a spare quickly would help quite a bit. Thus far the increased initial cost has out weighed these benefits, but it is something we're keeping in mind.

A final hardware related lesson is that near the end of their lives, nodes may start to fail in quantity as particular components degrade systemically. The main thing here is to keep an eye out for this happening. When a trend becomes apparent, it may be time for wholesale replacement rather than expending further effort on piecemeal repairs.

```
#!/bin/sh
FPING=/usr/local/sbin/fping
NODELIST=/usr/aero/etc/nodes-all

${FPING} -a < ${NODELIST} | \
    xargs -n1 -J host ssh -l root host $*
```

Figure 7: `oneallnodes` script

```
#!/bin/sh
restart_key=/home/root/.ssh/sge_restart.key
if [ -r ${restart_key} ]; then
        keyarg="-i ${restart_key}"
fi
export
QSTAT=/usr/local/sge/bin/fbsd-i386/qstat
FPING=/usr/local/sbin/fping
export SGE_ROOT=/usr/local/sge
export SGE_CELL=fellowship

${QSTAT} -f | grep -- -NA- | \
    cut -d@ -f2 | cut -d' ' -f1 | \
    ${FPING} -a | \
    xargs -I node ssh ${keyarg} root@node
/etc/rc.d/sge restart
```

Figure 8: `kickexecds` script

We have also learned that while most HPC software works fine on FreeBSD, the high performance computing community strongly believes the world is a Linux box. It is often difficult to determine if a problem is due to inadequate testing of the code under FreeBSD or something else. We have found that FreeBSD is usually the cause of application problems even when Linux emulation in involved. We have had good luck porting applications that already support multiple platforms to FreeBSD. There are some occasional mismatches between concepts such as the idea of "free" memory, but the work to add features such as resource monitoring is generally not difficult and simply requires reading manpages and writing simple code. We hope that more FreeBSD users will consider clustering with FreeBSD.

System automation is even more important than we first assumed. For example, shutting down the system for a power outage can be done remotely due to our power controllers, but until we wrote a script to allow automated connections to multiple controllers, it required manual connections to dozens of controllers making the process time consuming and painful. Other examples include the need to perform operations on all nodes, for example restarting a daemon to accept an updated configuration file. To simplify this we have created a simple script which handles most cases and nicely demonstrates the power of the Unix tool model. The script, `onallnodes` is shown in Figure 7. In general we find that many problems can be solved by appropriate application of xargs and appropriate Unix tools. For example the script in Figure 8 restarts dead SGE execution daemons on nodes. By running this out of `cron` we were able to work around the problem while working to find a solution.

In addition to the technical lessons above, we have learned a pair of related lessons about our users. Both are apparently obvious, but keep coming up. First, our users (and, we suspect, most HPC users) tend to find something that works and keep using it. They are strongly disinclined to change their method of operation and are unhappy when forced to do so. For this reason, we recommend that as much standard procedure as possible be developed and working before users are introduced to the system. It also suggests that voluntary adoption of practices will only work if a large benefit is involved and will never completely work. We have found this to be particularly true in regards to the scheduler. Second, because our users and domain experts[3] and not computer scientists, they often maintain mental models of the cluster's operation that are not accurate. For example, many believe that jobs that start immediately would inevitably complete before jobs that start later. While this seems logical, the only way jobs could start immediately would be for the system to be heavily over subscribed leading to substantial resource contention and thus large amounts of unnecessary swapping and excessive context switches which in turn can result in much longer job completion times. Another example is that while many users are interested in optimization and often micro-optimization, they often have a mental model of hardware the assumes no memory caches and thus discount cache effects.

5 Thoughts on Future Clusters

To meet our users ever expanding desire for more computing cycles and to improve our ability to survive disasters, we have been investigating the creation of a second cluster. In the process we have been in considering ways the system should be different from the current Fellowship architecture. The main areas we have considered are node form factor, storage, and network interconnect.

The locations we have been looking at have been engineered for cabinets so we are looking at standard 1U nodes and blades instead of our current custom, front-port solutions. In many regards the density and maintainability of blades would be ideal, but cost considerations are currently driving us toward 1U systems. The new systems will probably have at least 8 cores in a 1U form factor though.

Due to the fact that disks are the largest source of failures in our newer machines and that most users don't use them, we are considering completely eliminating disks in favor of high performance networked storage. The aggregate bandwidth from clustered storage products such as those from Isilon, NetApp, and Panasas easily exceeds that of local disk without all the nuisance of getting the data off the local storage at the end. There are two issues that concern us about this option. First, we can easily swap to network storage. Second, clustered storage is fairly expensive. We would be eliminating around $100 per node in disk costs, but that will not be enough to buy a large quantity of clustered storage. We think both of these issues are not too serious, but they are potential problems.

The use of Gigabit Ethernet as the Fellowship interconnect is working, but we do have some applications like computational fluid dynamics and radiation damage modeling where a higher bandwidth, lower latency link would be more appropriate. Additionally, our goal of moving away from having any storage on the nodes is leading us toward an architecture which places heavier demands on the networks. As a result we are considering both InfiniBand and 10Gb Myrinet interconnects.

For the most part, the other decisions in Fellowship's design have worked out and we think maintaining the basic architecture would be a good idea.

6 Future Directions & Conclusions

Currently Fellowship is working well and being used to perform important computations on a daily basis. We have more work to do in the area of scheduling, particularly on improving response time for short jobs, but things are working fairly well overall. Another area for improvement is better documentation to allow users to find what they need quickly and use the system correctly. We have made some progress recently with the migration of most of our documentation to a MediaWiki based Wiki system. We hope the ease of editing will help us write more documentation.

We are currently working with a team of students at Harvey Mudd College to add a web based interface to Fellowship. The interface is being built to allow users to submit inputs for specific jobs, but is being built on

[3]Including real rocket scientists.

top of tools which allow generic access to the cluster. We hope this will allow us to attract new classes of users.

Additionally, we have ongoing research work in the area of job isolation to improve both the security of jobs and the predictability of their run time. We are looking at ways to extend the light weight virtualization facilities of the FreeBSD jail [jail(8)] framework to add support for stronger enforcement of job boundaries.

We feel that FreeBSD has served us well in providing a solid foundation for our work and is generally well-supported for HPC. We encourage others to consider FreeBSD as the basis for their HPC clusters.

References

[Becker] Donald J. Becker, Thomas Sterling, Daniel Savarese, John E. Dorband, Udaya A. Ranawak, Charles V. Packer, *Beowulf: A Parallel Workstation for Scientific Computation* Proceedings, International Conference on Parallel Processing, 1995.

[Columba] *NAS Project: Columbia.* `http://www.nas.nasa.gov/About/Projects/Columbia/columbia.html`

[Davis] Brooks Davis, Michael AuYeung, Gary Green, Craig Lee. *Building a High-performance Computing Cluster Using FreeBSD.* Proceedings of BSDCon 2003, p. 35-46, September 2003.

[diskless(8)] *FreeBSD System Manager's Manual.* diskless(8).

[jail(8)] *FreeBSD System Manager's Manual.* jail(8).

[Gerzo] Daniel Gerzo. *Introduction to NanoBSD.* `http://www.freebsd.org/doc/en_US.ISO8859-1/articles/nanobsd/`

[MPI] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard.* `http://www.mpi-forum.org/docs/mpi-11.ps`

[OSCAR] *Open Source Cluster Application Resources.* `http://oscar.openclustergroup.org/`

[ROCKS] *Rocks Cluster Deployment System.* `http://www.rocksclusters.org/`

[SciClone] The College of William and Mary. *SciClone Cluster Project.* `http://www.compsci.wm.edu/SciClone/`

[Tolkien] J.R.R. Tolkien. *The Lord of the Rings* 1955.

[WindowsCCS] *Windows Compute Cluster Server 2003.* `http://www.microsoft.com/windowsserver2003/ccs/`

All trademarks, service marks, and trade names are the property of their respective owners.