USENIX Association

Proceedings of BSDCon '03

San Mateo, CA, USA
September 8–12, 2003

Building a High-performance Computing Cluster Using FreeBSD

Brooks Davis, Michael AuYeung, Gary Green, Craig Lee
The Aerospace Corporation
El Segundo, CA
{brooks,lee,mauyeung}@aero.org, Gary.B.Green@notes.aero.org

Abstract

In this paper we discuss the design and implementation of Fellowship, a 300+ CPU, general use computing cluster based on FreeBSD. We address the design features including configuration management, network booting of nodes, and scheduling which make this cluster unique and how FreeBSD helped (and hindered) our efforts to make this design a reality.

1 Introduction

For most of the last decade the primary thrust of high performance computing (HPC) development has been in the direction of commodity clusters, commonly known 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 adapted to them.

In 2001, driven by a number of separate users with supercomputing needs, The Aerospace Corporation (a non-profit, federally funded research and development center) decided to build a corporate computing cluster (eventually named Fellowship [1]) 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).

Fellowship is operational and being used to solve significant real world problems. Our best benchmark run so far has achieved 183 GFlops of floating point performance which would place us in the top 100 on the 2002 TOP500 clusters list.

In this paper, we first give an overview of the cluster's configuration. We cover the basic hardware and software, the physical and logical layout of the systems, and basic operations. Second, we discuss in detail the major design issues we faced when designing the cluster, how we chose to resolve them, and discuss the results of these choices. In this section, we focus particularly on issues related to our use of FreeBSD. Third, we discuss lessons learned as well as lessons we wish the wider parallel computing community would learn. Fourth, we talk about future directions for the community to explore either in incremental improvements or researching new paradigms in cluster computing. Finally, we sum up where we are and where we are going. Table 2 contains a listing of URLs for many of the projects or products we mention.

2 Fellowship Overview

The basic logical and physical layout of Fellowship is similar to many clusters. There are three core systems, 151 dual-processor nodes, a network switch, and assorted remote management hardware. All nodes and servers run FreeBSD, currently 4.8-STABLE. The core systems and remote manage-

[1] Originally, it was suggested that we name the cluster Frodo, but the need for a name which would provide us with multiple names for core equipment drove us to use Fellowship as the name of the cluster.

Figure 1: Fellowship Circa April 2003

ment hardware sit on the Aerospace corporate network. The nodes and core systems share a private, non-routed network (10.5/16). This equipment is mounted in a row 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 April 2003. The layout of the node racks is shown in Figure 2.

The core systems are a user or shell server, a data server which serves NFS shared scratch space and does backups, and a management server which runs the scheduler, serves NIS, and manages the nodes. The user server, `fellowship`, is the gateway through which users access the cluster. Users log into it and launch jobs from there. Home directories are stored on `fellowship` and exported via NFS to the nodes. The data server, `gamgee`, hosts 271 GB of shared scratch space for use by users during computations. It also runs a MySQL database for results storage and AMANDA for backups of key cluster systems. The management server, `frodo`, performs a wide variety of tasks. These include exporting account information via NIS, network booting the nodes, and scheduling user jobs.

The nodes are dual CPU x86 systems ranging from 1 GHz Pentium IIIs to 2.4GHz Xeons with 1GB of RAM installed. Table 1 gives a complete breakdown of CPU types used in Fellowship. All Pentium III nodes were purchased with 40GB IDE disks. The Xeon nodes were purchased with 80GB disks and Pentium III disks are being replaced with 80GB disks as they fail. The nodes are connected via Gigabit Ethernet through a Cisco Catalyst 6513 switch. The Pentium III systems are Tyan Thunder LE (1GHz systems) and Tyan Thunder LE-T with 3Com Gigabit Ethernet adapters installed in their expansion slots. They are mounted in 14" deep rackmount cases and were integrated by iXsystems. The Xeon systems are Intel 1U server platforms with dual on board Gigabit Ethernet interfaces. We purchased them from Iron Systems.

CPU Type	*Nodes*	*CPUs*
Pentium III 1GHz	8	16
Pentium III 1.26GHz	40	80
Pentium III 1.4GHz	39	78
Xeon (P4) 2.4GHz	64	128
Total	151	302

Table 1: CPUs in Fellowship nodes.

Unit	*Contents*
45	filler panel (*empty*)
44	filler panel (*empty*)
43	filler panel (*empty*)
42	48-port terminal server (r01ts: 10.5.1.0)
41	Cable management
40	
39	node (r01n32: 10.5.1.32)
38	node (r01n31: 10.5.1.31)
37	node (r01n30: 10.5.1.30)
36	node (r01n29: 10.5.1.29)
35	node (r01n28: 10.5.1.28)
34	node (r01n27: 10.5.1.27)
33	node (r01n26: 10.5.1.26)
32	node (r01n25: 10.5.1.25)
31	Baytech RPC4 8-port power controller
30	Baytech RPC4 8-port power controller
29	node (r01n24: 10.5.1.24)
28	node (r01n23: 10.5.1.23)
27	node (r01n22: 10.5.1.22)
26	node (r01n21: 10.5.1.21)
25	node (r01n20: 10.5.1.20)
24	node (r01n19: 10.5.1.19)
23	node (r01n18: 10.5.1.18)
22	node (r01n17: 10.5.1.17)
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	Baytech RPC4 8-port power controller
12	Baytech RPC4 8-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	5 110V 20A Circuits
1	

Figure 2: Layout of Node Rack 1

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 no manual configuration of nodes is required beyond determining their MAC address when they are installed.

Local control of cluster machines is made possible through a KVM-switch connected to a 1U rackmount LCD keyboard, monitor, and track pad. Remote access is available through Cyclades TS-series terminal servers. All nodes and servers as well as networking gear are connected to these terminal servers and console redirection is enabled on all FreeBSD machines. We have BIOS console redirection enabled on the Xeon systems, but not on the Pentium III systems as a bug tends to cause them to hang, even at very low baud rates. In addition to console access, everything except the terminal servers and the switch are connected to BayTech RPC4-15 serial remote power controllers. 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 single 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 Parallel Virtual Machine and the MPICH and LAM implementations of the Message Passing Interface [MPI]. Currently, our sole commercial parallel application is Grid Mathematica for which we were the launch customer.

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 is our current

Resource	*URL*
Big Sister	http://bigsister.graeff.com/
BProc	http://bproc.sourceforge.net/
Diskmark	http://people.freebsd.org/~brooks/diskmark/
Diskprep (enhanced)	http://people.freebsd.org/~brooks/diskprep/
Diskprep (original)	http://people.freebsd.org/~imp/diskprep.pl
DQS	http://www.scri.fsu.edu/~pasko/dqs.html
EmuLab	http://www.emulab.net/
FreeBSD	http://www.FreeBSD.org/
Ganglia Cluster Monitor	http://ganglia.sourceforge.net/
GEOM Overview	http://phk.freebsd.dk/geom/overview.txt
Global File System (GFS)	http://www.sistina.com/products_gfs.htm
Grid Mathematica	http://www.wolfram.com/products/gridmathematica/
LAM-MPI	http://www.lam-mpi.org/
LinuxBIOS	http://www.linuxbios.org/
LSF	http://www.platform.com/products/wm/LSF/
Maui Scheduler	http://www.supercluster.org/maui/
Myrinet	http://www.myri.com/myrinet/
MPICH	http://www-unix.mcs.anl.gov/mpi/mpich/
Nagios	http://www.nagios.org/
OpenPBS	http://www.openpbs.org/
Parallel Virtual Machine	http://www.csm.ornl.gov/pvm/
Rocks Cluster Distribution	http://www.rocksclusters.org/
Scalable OpenPBS	http://www.supercluster.org/projects/pbs/
Sun Grid Engine (SGE)	http://gridengine.sunsource.net/
Clusters @ TOP500	http://clusters.top500.org/

Table 2: Resources for Clusters

design. In this section we highlight the major design decisions we made while building Fellowship.

3.1 Operating System

The first major design decision any cluster faces is usually the choice of operating system. By far, the most popular choice is some Linux distribution. Certainly Linux is the path of least resistance and most people assume that, if it is a cluster, it runs Linux. In fact a cluster can run almost any operating system. Clusters exist running Solaris [SciClone], HP-UX, AIX, MacOS X, FreeBSD [Jeong, Schweitzer], and even Windows. ASCI Blue Mountain is actually a cluster of 48 128-CPU SGI systems running Irix [SGI].

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 toolkits such as NPACI's Rocks Cluster Distribution. 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, easy 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 part 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. In practice, our attempts to push changes back into the base operating system have met with mixed success. We have merged a few small changes, but the generally applicable portion of our diskless boot script changes have not been merged due to lack of time to sort out

conflicting changes to the main source tree.

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. In some cases, existing ports were not flexible enough for our needs, but for most applications, it works well. The availability of Linux emulation meant we did not give up much in the way of application compatibility. We have successfully run Grid Mathematica on the cluster after following the Mathematica installation documentation in the FreeBSD Handbook.

The disadvantages of FreeBSD for our purposes are 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 has not been a major issue for our users to date. Most of our jobs are compute-bound so the poor SMP performance under heavy IO is a moot problem. Threading has been more of an issue. We have users who would like to use threading for SMP scaling. We expect this situation to improve when we migrate to FreeBSD 5.x.

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. Additionally, there is a distinct shortage of compiler support for modern versions of FORTRAN.

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 many other choices are available. 64-bit SPARC and Alpha clusters are fairly common, and clusters based on Apple's XServe platform are popular in Macintosh shops. The major issues to consider are price, performance, power consumption, and operating system compatibility. For instance, Intel's Itanium2 has excellent performance, but is expensive and power hungry as well as suffering from immature operating system support. In general, x86 based systems are currently 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 have 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 newer Athlon based systems, the Xeon systems we purchased in this year's expansion, Apple XServes, and now AMD Opteron systems. Athlons have failed to match the power/performance ratios of Intel systems. Similarly, XServes are attractive, but offer sub-par performance and little improvement in power consumption in addition to being an incompatible architecture. We will not make a decision until we know what the hardware market landscape looks like late this year, but preliminary reports seem to indicate that the amd64 port of FreeBSD will allow us to explore using systems with much larger system memories while retaining x86 compatibility for users who do not want to think about which machines they are running on.

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 external interfaces then single processor system.

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, very large disks, or large amounts of RAM may be justified despite the exponential increase in cost.

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

Table 3: Configuration of first Fellowship nodes.

CPU	2 x Xeon 2.4GHz
Network Interface	On board gigabit
RAM	2GB
Disk	80GB 7200RPM IDE

Table 4: Configuration of latest Fellowship nodes.

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 then maximum RAM, and disks in the same size range as mid-range desktops. This resulted in the initial configuration shown in Table 3. The most recent node configuration is shown in Table 4.

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 datasets, little bandwidth is required and 100Mbps Ethernet is the obvious choice. For other, tightly coupled jobs, Myrinet with its low latency and 2 Gbps+2 Gbps bandwidth is the right solution. Other interfaces, such as upcoming InfiniBand products, provide alternatives for high speed interfaces.

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 Myrinet. We plan to improve the efficiently of our network by upgrading to use JumboFrames (9000 byte MTUs) in the near future. 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. Looking to our expansion next year, Gigabit Ethernet is standard on the motherboard, and with the large switches our cluster requires, the cost per port is less then 20% higher then 100Mbps Ethernet. We are considering the idea of creating sub-clusters within Fellowship with faster network interfaces such as Myrinet.

3.5 Addressing and Naming Schemes

There are three basic approaches to allocating IP addresses in a cluster. For small clusters, many architects simply put all the machines on an existing network. This has the advantage that no additional routing is needed for the nodes to talk to arbitrary external data sources. The disadvantage is that it typically means the IP-addresses do not correspond to physical objects so it is hard to distinguish machines. Additionally, not subnetting the cluster can make it too easy for inter-node communication to impact the rest of the network. The other two approaches involve placing nodes on their own subnet, either with public or private [RFC1918] addresses. Using public addresses has the advantage that with appropriate routers, cluster nodes can exchange data with arbitrary external data sources. On a subnet, IP-addresses can be mnemonic to help administrators remember which machine a particular address belongs to. The main disadvantage of using public addresses is that address space is becoming increasingly scarce and large allocations are difficult or expensive to obtain. The use of private addresses eliminates this pressure by allowing the use of 2^{24} addresses in the 10/8 address space. This allows useful mnemonic naming schemes without any pressures to use addresses efficiently. The disadvantage is that nodes cannot reach external data sources directly. If all they need to do is access HTTP or FTP servers, a proxy can be used, but many grid computing tools assume that all machines in a computation are on fully routed networks.

On Fellowship we chose to use the 10.5/16 private network. We chose this approach because we needed our own subnet to avoid consuming other networks resources and we would have needed at least a /23 allocation, which was not available at the time. Within our network 10.5.0/24 is reserved for core equipment. 10.5.255/24 is available for temporary

DHCP allocation to allow devices to acquire a network address before they have their MAC address recorded in the DHCP config file. The $10.5.X/24$ blocks are allocated to node racks numbered from 1. Originally, $10.5.X.0$ was the terminal server for that rack and $10.5.X.Y$ $(0 < Y < 255)$ corresponds to node Y within that rack. We have since moved the terminal servers onto the corporate network because they will not support JumboFrames. This allocation scheme would not be possible with public addresses due to address allocation authority requirements.

Choosing host names within a cluster is another issue faced by a cluster architect. The usual rules of host naming [RFC1178] apply to naming core servers. However, unless the cluster is very small and likely to remain so, a numerical naming scheme such as `node00`, `node01`, etc. is likely to be a better idea then trying to come up with a naming scheme that can handle hundreds of unique machines.

For Fellowship, we choose to name our core machines after members of The Fellowship of the Ring [Tolkien]. At some point we may run out of names and need to start using other characters from The Lord of the Rings, but the theme should easily hold for the core systems. We choose to name nodes after their host rack and their position within that rack. Nodes are numbered from the bottom (because we fill the racks from the bottom). Thus each node's name looks like `r##n##` with the first node in rack 1 being `r01n01`. Terminal servers were originally named `r##ts`, but have since been changed to `gimli-r##` with `gimli` being the terminal server for the core systems. The nice things about naming devices in the node racks this way is that conversion between IP-addresses and host names can be accomplished with a simple regular expression.

Domain names add a slight complication to the naming process. It is often useful to make the cluster into its own DNS zone. With Fellowship, all external systems reside within the `aero.org` zone and nodes reside within an internal use only `fellow.aero.org` zone. The disadvantage of this is that some software prefers that hosts are within the same zone.

3.6 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 failover support to improve uptime.

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 has three core servers: the data server, the user server, and the management server. All of these servers are currently 1GHz Pentium III systems with SCSI RAID5 arrays. The data server, `gamgee`, serves a 250GB shared scratch volume via NFS, runs a MySQL database for users to store results in, and does nightly backups to a 20 tape library using AMANDA. We are in the process of upgrading the scratch portion of the data server to a dual Xeon box containing 2.8TB of IDE RAID. Backups and databases will remain on `gamgee`. The user server, `fellowship`, serves NFS home directories and gives the users a place to log in to compile and run applications. The management server, `frodo`, hosts 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. We are in the process of upgrading `fellowship` and `frodo` to dual 2.4GHz Xeons with 285GB of SCSI RAID5 storage each, doubling their previous capacity.

These services were isolated from each other for performance reasons. In our model, hitting the shared scratch space does not slow down ordinary compiles and compiling does 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. Given unlimited funds, we would probably move most NFS service to an appliance type device

such as a NetApp file server.

3.7 Node Configuration Management

Since nodes generally 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 [Perlstein]. We then use the diskless remount support to mount `/etc` as `/conf/base/etc` and override ssh keys on the nodes. For many applications, this configuration would be sufficient. However, we have applications which require significant amounts of local scratch space. As such, 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. Instead of manual disk configuration, we created a program (`diskmark`) which uses an invalid entry in the MBR partition table to store a magic number and version representing the current partitioning scheme. At boot we use a script which executes before the body of `rc.diskless2` to examine this entry to see if the current layout of the disk is the required one. If it is not, the diskless scripts automatically use Warner Losh's `diskprep` script to initialize the disk according to our requirements.

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 about the nodes and rebooting them, they are ready for use.

Maintenance of the netboot image is handed by chrooting to the root of the installation and following *s*tandard procedures to upgrade the operating system and ports as needed. For operating system upgrades, we copy the entire root to a new location, upgrade it, and test a few nodes before modifying the DHCP configuration for all nodes and rebooting them to use the new root. We install software available through the ports collection via the standard process and manage it with portupgrade. Software which is not available in the ports collection is installed in the separate `/usr/aero` hierarchy.

One part of network booting Fellowship's nodes that has not worked out as planned is BIOS support for PXE. PXE is a standard feature on server-class motherboards, but seems to be poorly tested by manufacturers. More than once, our vendor had to go back to the motherboard manufacture to have them create a new BIOS to fix a PXE problem. We have found PXE to be somewhat unreliable on nearly all platforms, 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, particularly Cisco switches. Recently, we have been working on an enhanced version of `diskprep` which will allow us to create a FreeDOS partition that will automatically reboot the machine, giving it infinite retries at PXE booting.

3.8 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

Mountpoint	*Source*
/	frodo:/nodedata/roots/freebsd/4.8-STABLE
/conf/base/etc	frodo:/nodedata/roots/freebsd/4.8-STABLE/etc
/etc	mfs
/usr/aero	frodo:/nodedata/usr.aero
/tmp	/dev/ad0s2a
/var	/dev/ad0s2d
/home	fellowship:/home
/scratch	gamgee:/scratch
/db	gamgee:/db
/dev	mfs

Table 5: Sample node (r01n01 aka 10.5.1.1) mount structure

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, NQS, and DQS. These systems typically include a scheduler, but many of them also support running the Maui backfill scheduler on top of them. OpenPBS and SGE are freely available open source applications and are the most popular options for cluster scheduling.

For some applications, batch queuing is not a good answer. This is usually either because the application requires that too many jobs for most batch queuing systems to keep up or because the runtime 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 scheduleable 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. We have since contributed that port back to the main SGE source tree.

[2]A recent fork called Scalable OpenPBS may eventually remedy these issues.

3.9 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 are clusters that require multi-level security. We have some interest in the issues in such a system, but at this point have not done any serious investigation.

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. Internally we encourage the use of SSH for connecting to nodes, but do allow RSH connections. Our Sun Grid Engine install uses a PKI-based user authentication scheme. We discovered this is necessary because SGE's default privilege

model is actually worse than RSH in that it does not even require the dubious protection of a lower port. Inter-node communications are unencrypted for performance reasons.

3.10 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 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. A major advantage of Ganglia is that no configuration is required to add nodes. They are automatically discovered via multicast. We have also considered using Nagios to monitor nodes, but have not yet successfully deployed it. Monitoring is an area we need to improve on Fellowship. We have had disks fail after a reboot without anyone noticing, because the default FreeBSD diskless behavior causes it to boot anyway. It was nice that the nodes kept working, but we were surprised to find some machines had small memory based `/tmp` directories instead of 36GB+ disk based ones.

3.11 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, installing serial terminal servers to allow remote access to consoles and remote power controllers may be advisable.

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 1000 miles from the data center, making direct access even more difficult. As a result, we have configured all computers to provide remote console access via terminal servers and have provided their power through remote power controllers. This allows us to reliably reboot systems at will, which greatly aids recovery and remote diagnosis of faults. Not all problems can be solved this way, but many can. We were able to diagnose a reboot caused by running out of network resources, but not a crash caused by a RAID controller that died. We have had mixed results with BIOS console access. On the Intel Xeon systems it works well, but the Tyan Pentium III motherboards tend to hang on boot if BIOS console redirection is enabled. In both cases we are able to access FreeBSD's console, which has proven useful.

3.12 Form Factor

The choice of system form factor is generally a choice between desktop systems on shelves versus rack mounted servers. 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 more expensive due to components which are produced in much 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, rackmount systems deliver higher density, integrated cable management, and, usually, improved aesthetics.

Higher density is a two-edged sword. Low-end cases are often poorly designed and inadequately tested, resulting in overheating due to cramped quarters and badly routed cables. Additionally, a single rack can generate an amazing amount of heat. We estimate there is a 20-30 degree (F) difference between the front and back of Fellowship's racks of Xeons despite being in a well air conditioned underground data center. Those racks have a peak power consumption of over 6000W each.

A minor sub-issue related to rackmount 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 rackmount configuration immediately. We planned from the start to eventually have at least 300 CPUs, which is pushing reasonable bounds with shelves. The only thing that has not gone well with our racks is that we chose six inch wide vertical cable management, which gets cramped at times. We plan to use ten inch wide vertical cable management when we expand to a second row of racks next fiscal year.

4 Lessons Learned

The biggest lesson we have learned is that hardware attrition is a real issue. While we have not seen many hard-to-track instability problems, we have lost at least one machine nearly every time we have had a building-wide power outage, scheduled or unscheduled. As a result, we have learned that it is important to have a vendor who will repair failed or failing systems quickly. The fact that nodes fail more frequently then we had initially expected also means that neat cabling is more crucial then we first thought. 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. When we expand the cluster to a second row of racks next year, we plan to switch to having patch panels at the top of each rack connecting to panels beside the switch.

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 hope that more FreeBSD users will consider clustering with FreeBSD.

System automation is even more important then we first assumed. For example, shutting down the system for a power outage can be done remotely, but currently it requires logging in to all 20 remote power controllers. We are currently working on automating this as well as adding automatic shutdown of nodes in the event of external power loss.

5 Future Directions & Conclusions

Currently Fellowship is working well, but there are still improvements to be made, particularly in the areas of automation and scheduling.

We have planned for an evolving system, but we have not actually got to the stage of replacing old hardware so we do not know how that is going to work in practice. Clearly, at some point, nodes will be wasting more power then they are worth, but we do not know what that point is. A measure of FLOPS/Watt will be helpful in determining this. We also do not know if systems will start failing en masse in the future or if they will die slowly over a long period of time.

Other directions we need to pursue are in the area of scheduling. We need to better handle job models that do not fit well within the batch paradigm where users have a good idea how long their jobs will run. Some of our users have jobs that will run for weeks or months at a time, so this is a pressing concern. We are currently pursuing internal research funding to explore this issue further.

Another area of interest is some sort of cluster-on-demand [Moore] scheme to allow use of nodes in different ways at different times. One suggestion has been to create an Emulab [White] sub-cluster which can be used for computation when not being used for network simulation.

Distributed file systems like GFS and distributed process models like BProc are an area we would like to see explored further on FreeBSD. Currently there is significant work on Linux, but little on FreeBSD.

We are working to develop new higher level parallel programming toolkits to support specific applications such as mesh generation for computational fluid dynamics models. We are currently in the process of deploying the Globus Toolkit on the Aerospace network which will potentially allow users

to run applications which span multiple computing resources including Fellowship, other Aerospace clusters, and SMP systems such SGI Origins. Such applications could be built using programming tools such as GridRPC [Seymour] being developed by the GridRPC Working Group of the Global Grid Forum.

In the mid-term we are looking toward a migration to FreeBSD 5.x for improved SMP performance and threading support. For improved threading alone, this will be an important step for us. There are some significant challenges we need to overcome, the most significant one being the need to upgrade our network-boot infrastructure to the NetBSD derived rc.d boot scripts [Mewburn] and the GEOM disk subsystem [Kamp].

Fellowship currently runs a wide mix of jobs which are used being used to make significant decisions regarding space systems. 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.

Acknowledgments

We would like to acknowledge the support of the GPS and STSS program offices. Additional support was provided by the Aerospace Computer Systems Division. Without their funding of administrative costs, Fellowship would not be what it is today.

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.

[Moore] Justin Moore, David Irwin, Laura Grit, Sara Sprenkle, and Jeff Chase. *Managing Mixed-Use Clusters with Cluster-on-Demand*. Department of Computer Science. Duke University.
`http://issg.cs.duke.edu/cod-arch.pdf`

[Kamp] Kamp, Poul-Henning. geom(4). GEOM - modular disk I/O request transformation framework. *FreeBSD Kernel Interfaces Manual* FreeBSD 5.1.

[Perlstein] Perlstein, Alfred. FreeBSD Jumpstart Guide.
`http://www.freebsd.org/doc/en_US.ISO8859-1/articles/pxe/`

[Mewburn] Mewburn, Luke. The Design and Implemenation of the NetBSD rc.d system.
`http://www.mewburn.net/luke/papers/rc.d.pdf`

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

[SGI] Silicon Graphics, Inc. *Energy Department's Blue Mountain Supercomputer Achieves Record-Breaking Run*.
`http://www.sgi.com/newsroom/press_releases/2000/may/blue_mountain.html`

[Jeong] Garrett Jeong, David Moffett. *Welcome to ACME - The Advanced Computer Matrix for Engineering*.
`http://acme.ecn.purdue.edu/`

[Schweitzer] A. Schweitzer. *The Klingon bird of Prey PC cluster*.
`http://phoenix.physast.uga.edu/klingon/`

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

[RFC1918] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear. *Address Allocation for Private Internets*.

[RFC1178] D. Libes. *Choosing a Name for Your Computer*.

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

[White] B. White et al. An Integrated Experimental Environment for Distributed Systems and Networks. *In 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[Seymour] Seymour, K., Nakada, H., Matsuoka, S., Dongarra, J., Lee, C., Casanova, H., An Overview of GridRPC: A Remote Procedure Call API for Grid Computing. *3rd International Workshop on Grid Computing*, November, 2002.