Lessons Learned Building a General Purpose Cluster for Space Mission Applications

Brooks Davis, Michael AuYeung, Matt Clark, Craig Lee,
Mark Thomas, James Palko, and Robert Varney
The Aerospace Corporation
{brooks,mauyeung,mclark†,lee,mathomas†,jpalko†,varney†}@{†rush.}aero.org

Abstract

Over the past five years we have built and operated a computing cluster for company-wide unclassified technical and scientific computing. In this paper we discuss our cluster architecture, some of the architectural decisions we faced in the process, a sampling of applications run by our users, and lessons learned from building and operating this cluster. We also discuss future plans to use the cluster to support continuity of technical operations and netcentricity.

1. Introduction

Since 2001 we have designed, built, upgraded, and expanded the Fellowship[1] (short for The Fellowship of the Ring[2]) cluster, a computing cluster for unclassified technical and scientific computing. Fellowship was built both to meet the computing needs of our users and to give us practical experience with the issues of building and operating clusters. The Aerospace Corporation's role as systems engineer architect in support of national security space means that we perform many different technical activities and thus the cluster must support a array of different types of computations. With this diversity of applications comes an equally diverse set of users, many of whom have conflicting desires. These desires lead us to the current architecture of Fellowship. In this paper we discuss our current architecture, some of the architectural decisions we faced and how we met them. Where initial decisions have not stood the test of time we indicate that.

We then discuss a subset of our users' applications, how those applications have been deployed on Fellowship, the challenges they represent, and ways they could be modified to better fit our and other multi-user clustered environments.

Finally we conclude with lessons learned in the process and our plans for future work.

2. The Fellowship Cluster

Fellowship, the Aerospace Corporate Computing cluster consists of 232 compute nodes and a set of core systems connected by a gigabit Ethernet switch. The nodes each have two Intel Xeon or AMD Opteron processors with 32 of the Opterons having dual cores for a total of 528 CPU cores. We have equipped some nodes with specialized hardware: 32 of the nodes are connected by a separate 2Gbps Myrinet network providing low latency communications and eight others have high end graphics processing units (GPUs) for GPU computing research. The core systems consist of a variety of UNIX servers and a Network Appliance filer that currently provides user home directories. A high level diagram of Fellowship is shown in Figure 1.

When users connect to Fellowship, they do so via a 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.

This equipment is housed in our corporate data center. Sufficient cooling has been available from the start and initially power was adequate, but as the cluster has grown our power use has taxed existing resources requiring significant facilities enhancement projects. It has been necessary to add additional power distribution units and a new dedicated UPS to handle the cluster's load. Our nodes and networks switches are housed in two-post telco-style racks for

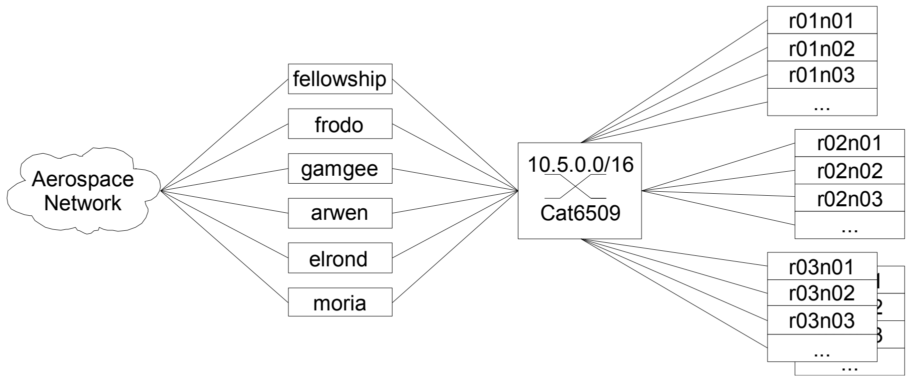

Figure 1. Fellowship layout

easy of access and free air circulation. Our core systems are installed in standard cabinets required by most larger servers. To enable two-post mounting of our nodes, the Opteron systems use custom 18-inch deep chassis with all data ports on the front and power supply ports on the back.

To enable automation of as many tasks as possible and thus make routine administration feasible we have made significant investments in remote access technology for our cluster. In particular, we have connected all machines to remote power controllers and created scripts to control them over secure shell (SSH) sessions. We have also installed network accessible keyboard-video-mouse (KVM) switches to allow access to core systems and selected nodes. Initially we used terminal servers to provide remote access to serial consoles on the nodes, but we later concluded that it was not generally worthwhile, largely because we found redirection of the hardware console before boot to be extremely unreliable.

All nodes and core systems run the FreeBSD operating system with exception of `moria` which runs a proprietary system. Most of them run FreeBSD 6-STABLE with a few older core servers running FreeBSD 4-STABLE. All of the nodes are booted using the Intel Preboot Execution Environment (PXE) and use a shared NFS home directory. This enables us to change the operating system configuration on a node by either updating the existing image or created a new one and rebooting the node. This will eventually cause scaling problems as the cluster expands and the single file server is increasingly loaded, but we have not seen significant problems to date. We are exploring options for that eventuality.

For parallel programming we support the Message Passing Interface (MPI) and the Parallel Virtual Machine (PVM) along with higher level libraries including the genetic algorithms libraries PGAPack and the locally developed GAL discussed in section 4.4. We also support Grid Mathematica and MATLAB. The specific MPI flavors we support include MPICH, MPICH2, and Open MPI. We are also beginning to test OpenMP support from the GNU OpenMP (GOMP) project as part of the GNU Compiler Collection (GCC) version 4.2.

3. Architectural Decisions

Now that we have covered the basic architecture of Fellowship, we will discuss the major decisions we faced and some of the available options. Space considerations prohibit discussing all the issues. The first and foremost decisions faced by a cluster architect are that of processor architecture and operating system. Other important decisions include physical form factor, network interconnect, storage, remote management infrastructure, node configuration management, scheduling model, and node naming and addressing scheme. Many of these are interdependent. The most obvious dependencies are between operating system and processor architecture, but others exist including some not so obvious ones like network interconnect and form factor (if the interconnect or cables will not fit, they can not be used.)

3.1. Processor Architecture and Operating System

The first architectural decision is often the choice of processor and operating systems since these will further constrain the other cluster design parameters. The primary requirement when choosing an operating system and processor architecture is that the chosen solution be compatible with the applications the cluster is designed for. If commercial applications are to be used this can be a constraint, for internally developed applications it may be a lesser issue though the difficulty of porting from one Unix variant to another varies significantly between applications.

For operating system selection price, compatibility with existing applications, availability of supporting tools, and local support staff experience are the most critical criteria. Staff familiarity is often overlooked as a key issue, but can be of major significance. The need for custom automation and the need to diagnose often Byzantine failures makes staff experience extremely valuable, particularly since the most crucial experience is difficult to learn except through practice. All things being equal, some version of Linux will likely be the easiest choice, particularly given the number of cluster-oriented distributions available and the general focus on using Linux for clustering. One emerging option of interest is to use a virtualization system such as Xen or VMWare to deploy OS images on demand[3]. This can allow improved utilization, better isolation or processes, better redundancy and reliability, and easier maintenance[4]. We choose the FreeBSD operating system due to local experience, the ability to feed changes back to the developers (the chief cluster architect is a FreeBSD committer), and the fact that it can run most Linux binaries.

For the processor architecture, major considerations include the ratios of power and price to performance and any architectural features that impact the performance the prospective job mix. All things being equal, the IA32 architecture or the AMD64 (aka x86_64) architecture is the path of least resistance with the best economies of scale and the widest tool chain support. We choose the IA32 architecture for our initial nodes with AMD64 machines (currently running in IA32 mode) in later purchases.

3.2. Network Interconnect

The major choices for cluster interconnects today are gigabit Ethernet, 10 gigabit Ethernet, Infiniband, and 10 gigabit Myrinet. 100Mbps Ethernet is technically an option, but for clusters of any size it is unlikely to save enough money relative to gigabit Ethernet to be worthwhile, except perhaps as a control network. The other options are currently all 10Gbps with 10 gigabit Ethernet having higher latency, but with the advantage of being just like any other Ethernet only faster. 20 and 40Gbps versions of Infiniband are on the horizon as well. At this point there is no clear winner in the 10Gbps interconnect space making consideration of price, upgrade path, and latency are paramount in clusters where gigabit Ethernet is not sufficient. When we made our initial cluster purchases, 100Mbps Ethernet, gigabit Ethernet, and 2Gbps Myrinet were the major options. The use of gigabit Ethernet represented a compromise between the users who wanted 100Mbps Ethernet due to their nearly uncoupled applications and those who wanted the high speed and low latency of Myrinet for tightly coupled applications like computational fluid dynamics (CFD) and weather prediction.

3.3. Storage

Cluster storage options range from disks on each node with little or no shared storage to diskless nodes where everything from the operating system to the user data is in central storage. Central storage can take several forms including a Storage Area Network (SAN), a clustered storage system such as Lustre, or protocol based storage either from a Network Attached Storage (NAS) product or a server. SAN based solutions have the advantage of providing each node direct access to the disks eliminating the bottlenecks associated with conventional protocol based storage. The down side is that SANs tend to be expensive and shared access to data requires special parallel file systems such as GFS, Xsan, or CXFS. NAS and host based storage have the downside of bottlenecks in access to the actual data, but the up sides of better support across multiple operating systems, avoiding the requirement for separate network, and in the case of host based storage, lower cost (sometimes dramatically). Clustered storage including represents an interesting middle ground with disks connected to an ordinary network by dedicated storage nodes. The solutions from Cluster File Systems (Lustre) and Panasas can provide a single file system view to Linux hosts with appropriate drivers and those solutions along with Isilon's solution provide protocol based access to the storage from multiple systems allowing network capabilities to grow with storage. Due to cost, initially low bandwidth requirements, and the limited options supported by FreeBSD of we selected host-based storage with local disks on each node to provide swap, scratch space, and temporary storage for large data sets. In 2005 we migrated our home directories from host based storage to a Network Appliance filer to improve performance and increase capacity. We have begun evaluating clustered and NAS storage for future high bandwidth applications.

3.4. Scheduler

Another major architectural decision is the choice of schedulern. There are three classes of options available: some form of batch job scheduling system, an application specific scheduler, or no scheduler at all. No scheduler is generally an option only when a cluster is dedicated to one use or to a small group of users with a single goal and no priority conflicts. Otherwise some form of scheduler is generally required to avoid overloading nodes (causing them to waste resources on swapping or recovering from cache misses) and to allow the enforcement of priorities between users or projects. When possible, using a batch job scheduling system is the easiest approach. A number of popular batch job scheduling systems exist including the Platform LSF, Portable Batch System (PBS), Sun Grid Engine (SGE), and Torque. In some cases such batch scheduling systems are impractical due to the nature of jobs being run. For example in applications where jobs are of indeterminate length with most being very short, it may be difficult for the

Table 1. Product Reference URLs

Product	***URL***
CXFS	http://www.sgi.com/products/storage/tech/file_systems.html
FreeBSD	http://www.freebsd.org/
Ganglia	http://ganglia.info/
GFS	http://www.redhat.com/software/rha/gfs/
Globus	http://www.globus.org/toolkit/
GOMP	http://gcc.gnu.org/projects/gomp/
Grid Mathematica	http://www.wolfram.com/products/gridmathematica/
Infiniband	http://www.infinibandta.org/home
Isilon	http://www.isilon.com/
LSF	http://www.platform.com/products/LSF/
Lustre	http://www.clusterfs.com/
MATLAB	http://www.mathworks.com/
MPICH2	http://www-unix.mcs.anl.gov/mpi/mpich/
Myrinet	http://www.myri.com/
Nagios	http://www.nagios.org/
OpenMP	http://www.openmp.org/
OpenMPI	http://www.open-mpi.org/
Panasas	http://www.panasas.com/
PBS	http://www.pbspro.com/
PGAPack	ftp://ftp.mcs.anl.gov/pub/pgapack
PVM	http://www.csm.ornl.gov/pvm/
Sun Grid Engine	http://www.sun.com/software/gridware/ http://gridengine.sunsource.net/
Torque	http://www.supercluster.org/projects/torque/
Xsan	http://www.apple.com/xsan/

scheduler to keep up with the jobs and to keep the nodes busy and the workaround bundling jobs together will result in poor time to solution. In such cases, it may be advantageous to write a simple scheduler where worker processes pull work from a database. For Fellowship we chose the SGE scheduler as the only free option that was stable on FreeBSD at the time. If we were looking today we would consider other options such as Torque which provides backfill scheduling.

4. Applications

In this section we discuss some representative applications being run on Fellowship. They include VISPERS, a launch vehicle telemetry processing application, a parameter sweep tool in the Satellite Orbit Analysis Program (SOAP), a tightly coupled code for atomistic modeling of radiation damage, and a trajectory design tool that uses genetic algorithms. For each application we discuss the application and how it fits into the cluster environment. Where appropriate, we discuss ways in which the application could be adapted to better fit the model of computation provided by modern, batch queued clusters. Finally, we will discuss our current research using the Fellowship cluster to address improving application support and expand our experience with a state of the art cluster design for space related applications.

4.1. VISPERS

VISPERS (Vibroacoustic Intelligent System for Predicting Environments, Risk and Specifications) is a suite of software tools used to analyze vibroacoustic measurement data captured via launch vehicle telemetry. Analysis of telemetry data is complicated by the need to reconstruct and repair the original telemetry stream by combining multiple stream fragments received from several distinct Telemetry Data Receiving Stations (TDRSs). Stream disruption can occur because the radio path from a launch vehicle antenna to a TDRS can be blocked, either by the vehnicle's exhaust plume, by a fin or other protrusion on the vehicle, by weather, or by the horizon. Such blockages produce so-called "data dropouts" in the telemetry stream, places where measurement data is missing. When stream fragments separated by data dropouts are joined, discontinuities are produced. In addition, vibration of the spacecraft can cause intermittent glitches in the telemetry stream such as spikes, or a single bit error can produce a measurement value that appears as a spike. Furthermore, charge build-up on the sensor can cause a DC drift, and an AC current can induce a spurious signal on the telemetry.
All of this would be easily dealt with if the telemetry was timestamped at the source, but timestamps are not inserted until the telemetry is received on the ground. Each TDRS inserts its own timestamps, but there is no guarantee that timestamps from different TDRSs will by synchronized. There are also additional timing anomalies that occur as a result during the transmission-reception process. As a result it is necessary to process and combine multiple imperfect telemetry streams in order to produce a single stream that is as close as possible to the original telemetry.

The VISPERS toolkit includes many software components for this purpose, but the two most computationally intensive are the VISPERS Artificial Intelligence Library (VAIL) for anomaly removal and *TACT* for stream consolidation. VAIL assists a human user to identify and eradicate dropouts and glitches. It automatically identifies locations in a telemetry stream where anomalies occur by searching the stream for predefined patterns. TACT (Telemetry Alignment and Consolidation Tool) consolidates multiple input streams into a single result and attempts to remove all anomalies (dropouts, spikes, DC-drift, signal bleed-through, discontinuities, etc.) identified by VAIL.

Both VAIL and TACT process telemetry streams by repeatedly applying algorithms to different local segments of the stream until the entire stream has been processed. This structure is well-suited for parallel implementation to obtain solutions faster. Since VAIL and TACT were part of a desktop tool kit, the decision was made to run these functions as *grid services* on remote machines rather than as traditional batch jobs to preserve the model of interaction. Hence, grid-enabled versions, called gVAIL and gTACT, were developed. Initially VAIL was ported to run on Globus version 3.2 and deployed on a handful of workstations[5]. Subsequently, both VAIL and TACT were ported to run on a later version of Globus version 4, also deployed on a handful of workstations. Near linear speedup of processing time was achieved by the gVAIL and gTACT combination[6].

The current design resulted from a straightforward port of the original Java source code to Globus. While expeditious, the resulting architecture has a number of disadvantages. First, as part of implementing VAIL and TACT as grid services, an *application-specific scheduler* had to be developed in order to partition the stream processing work across the particular set of workstations running the analysis services (deployed under Globus). Second, the analysis services must be pre-deployed to nodes that have been configured to register themselves with some central index. While this design was implemented on a cluster, it is impractical to run on many nodes due to the difficult of deploying the Globus Toolkit and problematic interactions with a cluster scheduler. In the future we hope to address some of these issues by enhancing the separation between the telemetry processing domain and the scheduler, ideally relying on the clusters scheduler rather than an internal one. We are also investigating the possibility of making some components stream oriented to allow live data to be processed during launches with reasonable delay.

4.2. Satellite Orbit Analysis Program

SOAP is a cross-platform interactive simulation engine for a variety of analyses related to orbital mechanics, such as coverage, line-of-sight visibility, and positional dilution of precision. Analytical results are displayed using three-dimensional animation of the relative motions of satellites, ground stations, aircraft, ships, the Sun and the Moon. The positions and velocities of these moving platforms are calculated from user-defined initial conditions using embedded propagation algorithms. Users can build coordinate system hierarchies that are then used to construct three-dimensional views, orient sensors, and define spacecraft attitudes.

One recently added feature of SOAP is a parametric study tool. This tool allows an analyst to vary one or more attributes of a platform or platforms across a range of values to optimize a particular metric over a time period. Some example uses of this tool are determining the optimum orbit for a new satellite in an existing constellation to maximize coverage of a particular region or determining the spacecraft solar panel angles that will produce the best average solar power. The tool works by examining the entire search space at a user specified level of detail and producing a table of metric values for examination by the analyst. Any one point may not take long to analyze, but the number of points can grow very rapidly with current high-resolution analyses taking hours or days to complete on a desktop. Fortunately this is an obvious candidate for parallelization. SOAP users at the Jet Propulsion Laboratory (JPL) observed this possibility and are working with Aerospace to implement a parallel capable version of SOAP. An initial MPI version is currently running and we plan to have a grid-enabled version working in July. The goal of this work is to provide nearly transparent access to Fellowship and other Aerospace computing resources for SOAP users by wrapping the parametric study function as a grid service. Users will fill out the parametric study form on their desktop client as they currently do and select the option to use grid resources. This will result in the *gSOAP client* contacting the available Parametric Study grid services and submitting their work. A *job handle* is returned to the client that can be used to check job status and retrieve results.

A variety users who need to perform trade studies including the Aerospace Concept Design Center (CDC)[7] are interested in the gSOAP capability. To support conceptual ground system design sessions at the CDC, we must provide a capability to guarantee turnaround times during primary study hours. Supporting this will require adjustments to scheduler policy.

4.3. Atomistic Modeling of Radiation Damage

Large-scale molecular dynamics (MD) simulation capabilities are being developed on Fellowship as part of a multi-scale modeling effort encompassing Monte-Carlo, MD, and ab-initio electronic structure computations aimed at a more fundamental understanding of radiation damage in electronic components. MD is a materials modeling technique in which the trajectories of individual atoms or particles are integrated based on interaction potentials that can be derived from empirical or quantum mechanical analyses. This project employs LAMMPS, a fully parallelized MD code utilizing spatial decomposition and MPI for interprocess communication.

Radiation is the primary hazard facing satellites in the space environment. Certain components (e.g. solar cells, CCDs) are particularly susceptible to degradation of the electronic properties of their semiconductor materials due to the displacement of atoms in their lattices by energetic particles in the radiation field. We are performing atomistic simulations to understand the electrical effects of isolated clusters of damage in silicon produced by the space radiation environment. By following the motions of the atoms during irradiation using MD, we are developing a catalog of defect structures produced by displacement damage in silicon and evaluating their relevant properties. Ab-initio electronic structure simulations will then be applied to this catalog to allow a better understanding of the types of damage responsible for the electrical degradation of silicon devices and the mechanism of these defects. Subsequently, we will use the damage-structure catalog to examine those events that are relevant to the annealing of complex defects. That information will then be employed in kinetic Monte-Carlo kMC simulations to allow compilation of the electrical effects of annealed damage structures in devices.

These systems are being investigated using the LAMMPS MD code distributed by Dr. Steve Plimpton of Sandia National Laboratory under an open source license[8, 9]. LAMMPS uses spatial decomposition, meaning that each processor is responsible for a specific region of space and any particles that may reside in it. The trajectories of the particles are integrated using a simple finite difference method, with the forces being calculated from the gradient of a combination of pair potentials and multibody terms. After each timestep, positions for those particles that have left the processor's region or are near enough to the boundary to produce a force on particles in neighboring regions (i.e. within the force cutoff distance) are transmitted to neighboring processors via a 6-way stencil. For electrostatic potentials that are important for the treatment of defect structures in semiconductors and decay inversely with distance, a cutoff of the potential produces unacceptable errors and the contribution to the electric field from the entire system must be calculated at each particle's position. LAMMPS accomplishes this by implementing the particle-mesh Ewald method where a series solution for the electric field is divided into real and reciprocal space contributions, and the structure factor is then solved using fast Fourier transforms (FFTs), resulting in computational expense *O(n log(n))*[10]. These operations tend to result in latency limited communication. The low latency Myrinet interconnects of Fellowship are ideally suited for this type of tightly-coupled application, allowing treatment of millions of particles.

4.4. Genetic Algorithms for Trajectory Design

The Navigation and Geopositioning Systems Department of Aerospace has produced the Genetic Algorithm Library (GAL), which consists of routines written in C (with interfaces for C and FORTRAN), to perform the basic functions involved in genetic algorithm optimization. It is capable of running in both serial and parallel implementations. The latter uses PVM interface for interprocess communication. This library is generally applicable to all genetic algorithm problems providing capabilities such as multiple objects allowing for multiple objective functions, positive and negative assortive mating, and mutation rates that depend on population diversity, among others.

This library has been applied in parallel form to orbital and trajectory analysis projects on the Fellowship cluster. One specific example of which was the design of optimal trajectories for a mission to the Jovian moon Europa. The SILO orbit propagator routines were used to integrate interplanetary trajectories for the individual solutions using a constant thrust in three sequential arcs with different rules controlling thrust direction. Two objective functions determined the fitness of the trajectories produced. The first was used to minimize the amount of fuel required, while the second maximized the ease of capture into orbit around Europa.

The search space consisted of launch date, total time of flight, out of plane thrust for the first arc, time of start for 2nd arc, and time of start for 3rd arc. 100 runs were conducted with different seeds for the initial random population, and each run was conducted for 60 cycles. The total resulting runtime was approximately 3 weeks on about 350 CPUs of Fellowship.

Out of a search space of 281 trillion possibilities the algorithm narrowed the results down to six possible optimal trajectories. Using a brute forces search a final optimal trajectory was chosen. The results of this analysis have shown that the primary factors affecting the optimal trajectory are the launch date and start time of the second thrust arc.

While the use genetic algorithms allows a great reduction in the number of points actually explored, the elapsed computing time of roughly 20 CPU years placed a significant strain on the resources of Fellowship. We were able to accommodate it because the system was lightly loaded at the time and we could run the genetic algorithms processes at a reduced priority on all nodes but this is incompatible with a fully scheduled system and with some of our applications that depend on minimal variation in latency. Work on a solution to this problem is discussed in the next section.

4.5. Scheduler Research

As previously mentioned, one of the goals in building a cluster at Aerospace was gain direct experience with cluster design and operation and perform research on finding solutions to problems encountered along the way. One of those problems is the challenge of balancing the needs of users who need rapid turnaround with those who need to run long jobs like the three week trajectory design run described in the previous section. Running the job in the background worked at the time, but that will not be a viable approach in the long term.

In a fortunate coincidence the genetic algorithms library's parallel mode was designed to support unreliable clusters where machines are periodically removed due to reboots or crashes. The code assumes such events are relatively rare and thus takes little care to be efficient in handling them, but it does work. We have been working on a prototype of a system to extend Sun Grid Engine's concept of parallel environments (PE) to allow them to change in size over the life of the computation. In the case of the GAL we have created a modified PVM PE that attempts to add additional nodes to the computation by submitting special jobs to the scheduler. When run, these jobs cause their node to join the running PVM virtual machine, wait a period of time, and then depart. At this time we have a limited proof of concept running and hope to have a production capable version working soon which we will use to produce benchmarks to determine if this approach is practical.

5. Lessons Learned

As a result of our direct experience with the design, implementation and operation of our cluster we encountered many issues and problems. All of them were things we had expected at some level, but were reinforced or illuminated in the cluster context. Key lessons include:

- Automate everything possible including monitoring
- Hardware failures happen with much greater frequency in a cluster
- Managing user expectations is hard, and
- It is often difficult to convince facilities people what your needs actually are.

We discuss each in turn.

5.1. Automation and Monitoring

The automation of routine or tedious tasks is important on most computer systems, but is especially critical on clusters where the same task may need to be performed hundreds of times in a short period of time. For example, once written, the software we use to manage our remote power controllers has saved countless hours, allowing quick and easy reboots of hung nodes. Likewise, a little time spend familiarizing ourselves with advanced uses of `ssh` and `xargs` has greatly eased the pain of performing simple tasks on each node. We have noticed that cluster builders who start small often assume that a more manual approach will remain feasible as the cluster grows when in fact it is unlikely to unless sufficient staff support is available.

Closely related to automation is monitoring. Initially we did very little monitoring of machines other than using Ganglia to monitor their load and resource use. This had the unfortunate effect of users finding problems with machines before we did. We have since significantly improved our monitoring infrastructure by adding a Nagios monitoring system and we now generally find out about problems before our users tell us.

5.2. Hardware Failures

Nearly all computer users have experienced the occasional, apparently random hardware failure on their desktop or laptop computers. With hundreds or thousands of machines, failures are a much more frequent occurrence. The two lessons we learned related to this are to watch for bad batches of components and to insure that your installation is neat enough to facilitate node repair. We have experience problems with bad batches of both power supplies and disks. In both cases, initial failures looked random, but it soon became apparent that we were looking at systemic failure. In both cases we ended up ordering replacement parts to insure that we could repair nodes immediately rather than waiting for a replacement from the vendor. In the case of the disks we also switched brands once it became apparent that we were dealing with a bad series of disks. In the process of replacing this equipment we found that places where our cable management was less than perfect made node repair significantly more difficult. In later installations we were much more careful about the process and repairs haven been easier there. Thus the observation that neatness counts. While there were not really a viable option when we starting architecting our cluster, this is one major advantage of well designed blade system. The ability to replace nodes or disks without touching any cables could be very useful.

5.3. User Expectations

One of the challenges that we have continually faced that we did not anticipate is that of managing user expectations. The most specific problem has been the classic problem of convincing users that schedulers are a good idea. Related to that the problem of adjusting users' often incorrect mental models of how modern computers and clusters work. For example, it is often hard to convince users that a job that starts later due to queue delay may well finish sooner than a job that starts immediately on a cluster with no scheduler due to competing loads and thrashing. Many of our problems with getting users to use the scheduler stem from the fact that we delayed imposing mandatory use of the scheduler for several years due to lack of time to document its usage. At that point users had a model of using the cluster and were disinclined to accept the requirements of the scheduler. We are working through those issues, but it is difficult and time consuming. Based on this experience we strongly recommend that implementers enforce scheduler usage from day one to avoid teaching users bad habits. We have also found that many users assume they can simply start using the cluster with little or no Unix experience. We have yet to find a good way to address the communication and operation problems this causes. One possible solution to some of these may be web-based portals to specific applications. We are investigating this possibility.

Another interesting user related challenge is that of balancing the user's need to be able to run their code locally with their need to run it on the cluster. In many cases, users want to be able to run their code on their workstations or laptops to ease development. Unfortunately this means they can not easily use services such as the scheduler's programmatic interface for allocating work to processors. This means the often end up with implementations that are significant compromises relative to the ideal architecture they would have developed if they were building code only for the cluster. We have not found a good solution to this problem, but suspect there may be room for some more parallel computing toolkits to help with the process.

5.4. Facilities

One final lesson learned from both the Fellowship cluster and other clusters is to make absolutely certain that whether they believe your system needs it or not, your facilities people install sufficient power and cooling. With Fellowship we have had problems were we asked for a certain number of circuits of a given capacity and they were provided, but they were fed by power distribution units that did not have the total capacity required. Other people installing clusters have has similar issues, albeit on a smaller scale. We have also seen problems with inadequate cooling being installed causing significant system damage when the units shut themselves down unexpectedly.

6. Future Work

These experiences with the design and use of a cluster computer have clearly defined several key directions for future work.

First, it is clear that a service architecture will make it much easier to support desktop applications for users who are non-computer-specialists. Applications such as SOAP and VISPERS will have a desktop client that is capable of many tasks. For those analyses or functions that are too compute intensive for a desktop machine, however, the client will be able to (a) discover the available services and machine resources, (b) select a particular machine or set of machines, (c) invoke the desired service, and (d) collect the results. Each of these steps could be pre-configured or automated to make the interaction with the remote resources as transparent as possible. Such desktop clients could also orchestrate the workflow among multiple remote services by using a workflow engine that independently manages the transfer of data and the execution of services [11]. These service-oriented functions would be implemented using emerging standards for grid computing. Globus [12] is a de facto standard since it is used by many projects. The current Globus version 4 (GT4) is based on the Web Services Resource Framework (WSRF) family of web service standards [13]. The application clients could be implemented in a variety of ways, including web-based portals, that rely on the appropriate security models.

As already noted, the issue of scheduling and scheduler policies is a key aspect. The ideal cluster scheduler must be able to manage a jobs mix that includes short jobs, very long-running jobs, queued jobs, interactive jobs, jobs of known, deterministic running time, jobs with unknown running times, jobs that required a known, fixed number of nodes, and jobs that can require a variable number of nodes during execution that is not known until run-time. No single scheduler currently addresses all of these requirements well.

We are investigating ways to schedule variable numbers of nodes for a particular job. The service architecture approach for using the cluster, however, will also impact the scheduler since some applications would greatly benefit from the ability to run a persistent service on the cluster, a service that is long-lived and waits for requests to arrive. The use of workflow engines could also impact the scheduler since certain applications could require co-scheduling, where multiple services must be scheduled at the same time to meet mission-critical processing deadlines. This could entail multiple services on the cluster, or between the cluster and other resources.

Besides these technical scheduling issues, we also point out that non-technical, organizational scheduling policies are just as important. Despite the best planning, there will be times when mission and programmatic requirements demand that the scheduler be overridden, e.g., to devote the entire cluster to one particular application for some period of time. Hence, it is best that an organizational scheduler exception policy be in place that is well-known to the user community. Such a policy should define the possible scheduler exceptions, their duration and the required level of management authorization. When a scheduler exception is authorized, other users should understand why and be given as much advance notice as possible to re-arrange their own impacted computational requirements.

Another key topic is that the notion of enhanced scheduling capabilities and the service architecture concept directly supports the goal of *continuity of technical operations*. Many mission-critical functions are computer-based and must be able to recover from catastrophic failures, such as a massive power failure or network outage. Hence, it should be possible to flexibly re-schedule applications and re-direct remote clients to available resources. The Fellowship cluster and scheduler will be key resources in our efforts to dynamically manage sets of distributed applications and resources, including multiple clusters.

Finally we note that the use of remote clients, portals, workflow engines, enhanced scheduling, and dyanmically managing distributed resources, are hallmarks of *network-centric operations* or *netcentricity*. As these concepts are mapped to best practices and concrete implementations in DoD and the defense community, we will be ideally positioned to evaluate and guide this long-term process.

7. Conclusions

Building and operating Fellowship has been a challenging, but rewarding experience. Our users have produced significant results and are in the process of developing many interesting applications. We are working to enhance Fellowship and are constantly on the lookout for ways to improve the overall user experience. Grid service interfaces to the cluster from within new or existing external applications, and application-specific web portals, are promising approaches we plan to pursue.

8. References

[1] Davis, B. , Michael AuYeung, Gary Green, Craig A. Lee: Building a High-performance Computing Cluster Using FreeBSD. *BSDCon 2003*, 35-46

[2] Tolkin, J.R.R. *The Lord of the Rings.* 1955.

[3] Barham, P., B. Dragovic, et al. (2003). Xen and the Art of Virtualization. *SOSP'03*, October 19-22, 2003, Bolton Landing, New York.

[4] Clark, C., K. Fraser, et al. Live Migration of Virtual Machines. *NSDI 2005*.

[5] Bentow, B., J. Dodge, A. Homer, C. Moore, R. Keller, C. Lee, M. Thomas, J. Seidel, M. Presley, R. Davis, and J. Betser, "Grid-Enabling a Vibroacoustic Analysis Toolkit", *International Journal of High Performance Computing and Networking*, 2006. (To appear.)

[6] Bentow, B., J. Dodge, A. Homer, C. Moore, R. Keller, M. Presley, R. Davis, J. Seidel, C. Lee, and J. Betser, "Grid-Enabling a Vibroacoustic Analysis Application", 6th *International Workshop on Grid Computing*, November 13-14, 2005, pp. 33-39.

[7] Smith, P.L., A.B. Dawdy, T.W. Trafton, R.G. Novak, and S.P. Presley. Concurrent Design at Aerospace, *Crosslink*. The Aerospace Corporation. Winter 2001.

[8] Plimpton, S. (1995). "Fast Parallel Algorithms for Short-Range Molecular-Dynamics." *Journal of Computational Physics* 117(1): 1-19.

[9] Plimpton, S. J., R. Pollock, et al. (1997). Particle-Mesh Ewald and rRESPA for Parallel Molecular Dynamics Simulations. *Eighth SIAM Conference on Parallel Processing for Scientific Computing*, Minneapolis, SIAM.

[10] Essmann, U., L. Perera, et al. (1995). "A Smooth Particle Mesh Ewald Method." *Journal of Chemical Physics* 103(19): 8577-8593.

[11] Lee, C., B.S. Michel, E. Deelman and J. Blythe, "From Event-Driven Workflows towards A Posteriori Computing", *Future Generation Grids*, (Reinefeld, Laforenza, Getov, eds.), Springer-Verlag, 2006, pp. 3-29.

[13] OASIS Web Services Resource Framework (WSRF) TC, http://www.oasis-open.org/committees/workgroup.php?wg_abbrev=wsrf.