

# Massively Parallel Computing Using Commodity Components \*

Ron Brightwell<sup>†</sup>   Lee Ann Fisk<sup>†</sup>   David S. Greenberg<sup>†</sup>   Tramm Hudson<sup>†</sup>  
Mike Levenhagen<sup>†</sup>   Arthur B. Maccabe<sup>‡</sup>   Rolf Riesen<sup>†</sup>

September 22, 1998

## Abstract

The Computational Plant project at Sandia National Laboratories is developing a large-scale, massively parallel computing resource from a cluster of commodity computing and networking components. We are combining the knowledge and research of previous and ongoing commodity cluster projects with our expertise in designing, developing, using, and maintaining large-scale MPP machines. In this paper, we present the design goals of the cluster and contrast them to the goals of other cluster systems. We discuss our initial assumptions regarding the state of commodity software and hardware with respect to scalability and performance. We also present our approach to developing a commodity-based computational resource capable of delivering performance comparable to production-level MPP machines. We present some of the lessons learned from the 96-node prototype cluster, and observations that led to several improvements in the design and implementation of a 400-node addition to the cluster.

Keywords: Massively Parallel, Workstation Cluster, Beowulf, Distributed Memory, Message Passing, MPI.

## 1 Introduction

Using tightly-coupled clusters of commodity off-the-shelf (COTS) personal computers (PC's) to build a dedicated parallel computing resource has become popular due to cost/performance benefits. Small-scale clusters have shown performance comparable to MPP systems[32]. While the initial steps toward commodity supercomputing have shown promise, many obstacles remain for the development of large-scale clusters with the compute and I/O power required to compete directly with MPP systems.

Three efforts to build clusters from commodity components are the Berkeley NOW[7], Beowulf[28] consortium, and the HPVM[20] project. While these projects address many of the issues relevant to the size and capability of their targeted systems, we believe many issues and complexities must be addressed to scale these types of systems to reach Sandia's current teraFLOPS level of compute

---

\*This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000.

<sup>†</sup>Computational Sciences, Computer Sciences, and Mathematics Center, Sandia National Laboratories, P.O. Box 5800 M.S. 1110, Albuquerque, NM, 87111-1110, (505)845-7432, (505)845-7442 FAX, cplant-dev-nm@cs.sandia.gov

<sup>‡</sup>Department of Computer Science, University of New Mexico, FEC 313, Albuquerque, NM, 87131, (505)277-6504, (505)277-6927 FAX, maccabe@cs.unm.edu

performance. In this paper, we present Sandia's approach to constructing a large-scale, massively parallel processing machine constructed from commodity workstation and networking hardware.

In the next section, we present an overview of relevant workstation cluster projects and an overview of the Sandia/Intel TeraFLOPS machine. Section 3 summarizes the shortcomings of these approaches to high-performance computing. In section 4 we discuss the concept and design of the Sandia approach to a scalable, high-performance cluster. Section 5 presents an overview of the Cplant prototype cluster. Section 6 summarizes our experiences to date with the prototype. Section 7 contains an analysis of the lessons learned from the prototype as well as the design of the next generation machine and a discussion of future work. We conclude with a summary of the key contributions of this research.

## 2 Background

The Computational Plant (Cplant) effort has been driven by our experience in developing system software for large-scale MPP systems. In recent years, it has become apparent that special-purpose MPP systems will be replaced by clusters built from commodity components. In many ways, the Cplant project reflects our efforts to build the environment provided by a large-scale MPP system using commodity components. In this section, we survey the state of the art in building cluster systems from commodity components. In particular, we discuss the Berkeley NOW, Beowulf, and HPVM systems. In addition, we give an overview of our efforts in large-scale MPP systems.

### 2.1 Berkeley NOW

The goal of the Berkeley NOW (Network Of Workstations) is to "demonstrate the viability of constructing large-scale parallel computing systems from commodity components, resulting in a fast, inexpensive, and highly available resource" [7]. The Berkeley NOW system is not based entirely on commonly available software components. Much of the software that enables the capability of the cluster was specifically developed as part of the project to support a large-scale cluster.

GLUnix (Global Layer Unix)[8] is a layer of system software designed to: provide transparent remote execution; support interactive parallel and sequential jobs; and provide load balancing. This system is currently running on a 100+ node cluster of Sun UltraSparc workstations connected by Myrinet[2] as well as Ethernet. Each workstation runs a full-featured Solaris operating system. GLUnix provides process management across the NOW using UNIX sockets, signals, and daemons. In the context of GLUnix, process management provides for starting, stopping, and interacting with

a cooperating set of processes across a set of nodes. The GLUnix software runs entirely at the user level and is designed to be portable across a wide variety of UNIX platforms provided there is support for a shared file systems and an underlying homogeneous operating system.

GLUnix provides a cluster-wide name space by establishing a unique network process id for each job and also establishes a virtual node number (rank) for each process in a job. It handles redirection of standard I/O and signals from the controlling process to all of the processes in a job. Remote processes can be gang scheduled or co-scheduled, and user application faults, as well as system application faults, are handled gracefully.

GLUnix has a master process that controls resource allocation within a cluster. The master process receives information concerning the current load on each node in the cluster from a daemon process running on each node. This daemon is also responsible for starting and signaling jobs, as well as detecting job termination and providing for process synchronization across a set of processes within a job. Each process can access GLUnix services through a set of library calls. These calls allow a process to determine the global process id for the job and the virtual node number of the process within the job. Additionally, calls to send a signal to another process in the job or to synchronize all of the processes in a job are provided. GLUnix also has a reservation facility whereby a set of nodes can be partitioned for use by a specific group of users. This facility was not in the original design, but the developers realized the importance of this feature as the machine became more heavily used and as the need for accurate performance measurements increased.

In addition to the GLUnix facilities, the Berkeley NOW cluster has a prototype file system, xFS, that provides a global, high-performance file system. Files are striped across across nodes in a RAID-like fashion to increase I/O bandwidth. Two prototype global virtual memory systems that allow processes to page to the memory of remote idle nodes exist.

An active message layer[15] is provided for high-performance message passing. Two different implementations of AM are available, one that only supports a single process on a node, and one that can multiplex messages between multiple processes on a node. Various message passing libraries, such as MPI, are available for AM, as well as the Split-C[6] programming language.

While GLUnix does provide many of the features desirable for cluster computing, some shortcomings have been identified[8]. Scalability is limited by the number of file descriptors that a master process can have open at any time. Each process in a parallel job opens three TCP/IP connections back to the master process for standard I/O services. Solaris file descriptor limits, even when increased to 1024 per process, still only allow for a maximum of 341 processes per job.

The lack of a GLUnix port to Myrinet limits access to the high speed network to application

processes. In particular, GLUnix services do not have access to high-speed communication.

GLUnix only supports an SPMD-style interface and does not have support for launching a single job composed of multiple binaries. While it is clear that most applications use a single binary, there are applications that require the flexibility of having multiple binaries.

While the remote execution facility does provide some valuable features, the transparency of the remote execution facility can be compromised easily. Any system calls made on a remote node reflect the current state of that node. The environment in which these calls execute may not be the same as that of the node on which the master process is running, which can lead to unpredictable behavior.

Finally, the size of the cluster has also exposed problems with the weak cache consistency policy of NFS where stale NFS cache values can lead to use of old executables. This is an example of scalability being limited by relying on a software component that was not designed with scalability in mind.

The Berkeley NOW cluster was used to achieve a world record for disk-to-disk sorting performance, but to date, its largest use has been for distilling images off of Usenet and for compiling via a parallel make utility[8].

Researchers at Berkeley are building on the foundation of the NOW project with the Millennium project[31]. The goal of this project is to develop and deploy a campus-wide cluster of clusters to support advanced scientific computing, simulation, and modeling applications. This project spans several different campus departments in an attempt to support expanding research collaborations and increasing numbers of computationally intensive investigations.

## 2.2 Beowulf

A Beowulf-class system is defined to be “a combination of hardware, software, and usage model which yields a domain of computing that is scalable, exhibits excellent price/performance, provides a sophisticated and robust environment, and has broad applicability” [28]. Beowulf systems are clusters of PC’s that employ COTS technology and typically run low-cost or open-source UNIX system software (e.g., Linux, FreeBSD, or OpenBSD). These systems generally support an explicit message passing programming model, and most applications are scientific or engineering in nature. In the Beowulf context, the definition of COTS has been extended to include “mass market common off-the-shelf” ( $M^2$ COTS). This allows Beowulf systems to use software that does not come from commodity vendors but which is publicly available from universities or other research institutions. In fact, a few key components of the networking software for Beowulf systems have been created

specifically for Beowulf systems by Beowulf developers .

The complexity in building a Beowulf system is in surveying the market to determine availability of the necessary components and in assembling and integrating them into a system. Any missing or inadequate pieces are identified, and common software suppliers (e.g., the Linux community) are encouraged to fill those gaps. The following attempts to characterize the typical Beowulf environment and provide an overview of some the existing shortcomings that have been identified[28].

Beowulf clusters are built from a conglomeration of individual, independent workstations that can function either autonomously or as part of a group. Each workstation has a complete environment for each user. This is usually achieved by NFS mounting users' home directories from a common file server. Starting processes on remote machines is accomplished through the UNIX remote shell (rsh) mechanism. Interprocess communication is based on standard UNIX network programming constructs such as sockets with either TCP/IP or UDP/IP network protocols. Message passing software, such as MPICH[10], LAM[18], or PVM[29] is also provided to give applications a level of abstraction above the underlying network layers.

Beowulf clusters are dependent upon UNIX sockets and wide-area network protocols. The inability of these protocols to effectively utilize high-speed system-area network hardware has been well established[22]. The primary limitation to scaling and efficiency for Beowulf systems has been directly attributed to the lack of high-performance network infrastructure. The bi-section bandwidth, latency, and global synchronization performance of Fast Ethernet switches and wide-area network protocols are not sufficient to support large-scale systems. Therefore, Beowulf developers have found that many applications are not well suited to this type of cluster, especially applications that are latency sensitive[28]. Given our experience with large-scale MPP systems, we believe that there are also many bandwidth sensitive applications that are not well suited to this type of cluster.

The user environment is frequently dictated by the software package being used. Each package has a unique compile environment, and usage model. For example, MPICH has a set of compilers and a startup mechanism.

Choosing the nodes to use is largely in the hands of the user. It is the user's responsibility to start any necessary daemons or to provide the list of hostnames to use in allocating compute resources. In order to allocate the least-loaded nodes in the system, the user has to survey the available nodes and construct the appropriate list of hostnames. Users who do not want to construct their own resource lists can use a system-wide resource list. However, most software packages will start at the beginning of this resource list and allocate processes to machines in a round-robin fashion. This method can cause an unequal load distribution when multiple users run jobs simultaneously.

It is also the user's responsibility to respond to changes in the system configuration. For example, when machines fail, they have to be removed from any resource lists. This change can impact any users that have their own configuration lists. Similarly, any nodes added to the cluster must be added to each user's `.rhosts` file as well as any user configuration lists. While none of these responsibilities is overwhelming for a typical user, it clearly demonstrates that the user is responsible for determining how the available compute, network, and disk resources in the system are utilized. While these mechanisms work for clusters of a hundred nodes, they will become more difficult and tedious on clusters with a thousand nodes. Currently, research is being directed towards resource allocators, gang schedulers, and load-balancing software to try to alleviate these problems.

From a system administration perspective, each node in a Beowulf cluster is managed individually. Each workstation runs a complete, full-featured operating system. Since users can log into any machine, a complete set of daemons, shells, and services has to be available. In some cases, a serial network is provided for console access to each node to support system administration activities. Booting a node is done either through the console, or by initiating a reset at the computer itself. The operating system is booted off of the local hard drive. Upgrading the OS or any of the system software requires each node to be addressed individually.

While Beowulf-class clusters have shown that it is possible to develop systems that are able to compete with small MPP systems and deliver adequate performance for extremely low cost, they have not yet shown that they can scale or perform to the level of large MPP systems.

### 2.3 HPVM

The HPVM (High-Performance Virtual Machines) project at the University of Illinois is "an attempt to enable high-performance computing on distributed computational resources with support for traditional supercomputing applications as well as emerging high-performance distributed applications"[20]. HPVM tries to address the critical software challenges that arise when trying to integrate high-performance commodity networking technology with gigaFLOPS microprocessors. These challenges include delivering high-performance communication with standard APIs, coordinating scheduling and resource management, and managing heterogeneity.

The first version of HPVM software supports clusters of Pentium Pros running Windows NT 4.0 or Linux, connected by either Myrinet or Ethernet. The high-performance version that uses Windows NT also requires Platform Computing's Load Sharing Facility (LSF). HPVM's high-performance message passing layer, FM[13], supports a variety of application interfaces, including MPI, SHMEM (a la the Cray T3), and Global Arrays. Details of HPVM are not well documented in the literature,

but the following attempts to describe the interworkings of the system[19].

The Global Resource Manager (GRM) process runs on a single Internet-accessible machine within the cluster. It assigns each process within an application a logical node number (rank) that can be used to uniquely address each process within the application. It is also responsible for distributing mapping information so that each process knows the location of all of the other processes.

The Context Manager (CM) process runs on each node in the cluster. It is responsible for managing the Myrinet interface and insuring that processes do not interfere with each other's data. The CM communicates with a newly spawned process to establish a context for the Myrinet interface.

An HPVM front-end server process runs on an Internet-accessible machine. It receives requests from the front-end client process and communicates with LSF to spawn application processes out to the available nodes. The front-end client process is a Java applet that provides the user's interface to the HPVM system.

Several LSF services and processes run on the system. The Load Information Manager supplies information about the activity of nodes. The Remote Execution Server executes programs on the remote nodes. The Slave Batch Daemon handles execution of batch jobs on the nodes in the cluster. These LSF services interact with the front-end client through a set of library functions provided with the LSF software.

The current software distribution does not provide any tools to manage, configure, or maintain an HPVM cluster. The installation guide that comes with the software distribution provides detailed, step-by-step instructions on how to configure each node in the cluster. In order to configure a node in the cluster, one must log on to the console and install the necessary software in the appropriate locations. Since there is nothing provided to automate this process, upgrades and new releases of software must be installed by hand.

Many of the processes in the cluster provide vital services and must be configured at well-known hostnames and addresses. It is not clear whether there are any redundant capabilities that allow the system to function in the event of failure.

## **2.4 Sandia/Intel TeraFLOPS Machine**

The Sandia/Intel TeraFLOPS machine is composed of more than 9000 Pentium Pro processors connected by a network capable of delivering 800 MB/s bidirectional bandwidth (see [23] for details). This machine is part of the Department of Energy's Accelerated Strategic Computing Initiative (ASCI). It has a peak performance of 1.8 teraFLOPS, and is the current world record holder for compute performance, demonstrating over 1.3 teraFLOPS on MPLINPACK. This level of perfor-

mance is required to solve the types of problems that are critical to Sandia and the ASCI program[1]. The TFLOPS machine is the culmination of more than ten years of research and development in massively parallel, high-performance, distributed memory computing. The ability of the TFLOPS machine to scale to more than 4000 nodes has been cited as an important factor in establishing the feasibility of Beowulf clusters[28]. However, we believe that it is the combination of hardware and software design that achieved this capability, and not simply hardware.

The software design for every component in the system had to be scalable. The high-performance operating system on the TFLOPS compute nodes was designed and developed by Sandia and the University of New Mexico[26] and ported to the TFLOPS by Intel. This lightweight kernel design stems from earlier experiences in providing a high-performance operating system optimized for distributed memory, messages passing MPPs[14]. The key component of the system software is a high-performance message passing layer called *portals*[25]. Portals are essentially data structures within an address space that inform the kernel how and where messages should be deposited. Portals were one of the earliest attempts to establish an application bypass mechanism for message passing. Commodity networking technology has only begun to realize the benefits of decoupling the network from the application processor, with emerging technologies such as the Virtual Interface Architecture (VIA)[5] and Scheduled Transfer (ST)[30] <sup>1</sup>.

The TFLOPS machine has a completely separate subsystem for maintaining and diagnosing the machine. The reliability, availability, and serviceability (RAS) system consists of a separate network of processors that monitor the health of the system. This network is used to identify components of the machine that have failed and allows these components to be replaced and integrated back into the machine. This type of subsystem is vital for large-scale systems.

In addition to the high-performance data movement software, the TFLOPS employs a partition model of resources[9]. This model divides the resources of the machine into partitions based on functionality. Each partition provides access to a specialized resource. The minimal set of partitions in the model are the service and compute partitions. The compute partition is the largest portion of the machine and is dedicated to delivering processor cycles and interprocessor communications to applications. The service partition provides a full UNIX environment where users log in to access the compute partition. Nodes in this partition provide the familiar UNIX shells, tools, and utilities. Users launch their parallel jobs from the service partition into the compute partition.

The TFLOPS machine is a space-shared resource. Nodes are allocated to a specific job and user.

---

<sup>1</sup>Portals support an application bypass in the sense that when the portals' code is interpreted on a coprocessor, messages can be delivered to the application without interrupting the execution of the application. However, unlike current OS bypass strategies, the portals implementation requires full address translation support.

Once a node has been allocated, it can only be used by that user until the job terminates. Even though the TFLOPS environment supports multiple processes per node, this feature of the machine is rarely used in practice. Typically a job will need all of the compute, memory, and communication resources that a node can provide. If more compute power is needed, more nodes are added.

*Yod* is the service node tool to launch an application. Arguments to *yod* specify the number of nodes to allocate and the executable image(s) to load on those nodes. All standard I/O for processes in the compute partition is channeled through the *yod* process.

Most applications use MPI for communication between processes in the compute partition. We have implemented a high-performance MPI based on portals[4]. All compute node processes in the same job are in the same `MPLCOMM_WORLD`.

The file system for the compute node processes is the same file system that the user in the service partition sees. That is, any file which is visible from the service partition is also visible to the compute node processes. Low bandwidth file access is easily supported by channeling I/O request through the *yod* process. In addition, the TFLOPS environment supports high bandwidth file system access using nodes in an I/O partition.

Users can view the status of the compute partition and the status of individual jobs with a utility called *showmesh*. Showmesh is a command line tool that gives a text display of the status of each compute node in the machine. The command lines for each job follow the display and provide more detailed information about each running job. Each job is also assigned a unique identifier that can be used by some tools to address all of the processes in a job.

### 3 Limitations

While the cluster-based projects have firmly established a foundation upon which small- and medium-scale clusters can be based, the current state of cluster technology does not support scaling to the level of compute performance, usability, and reliability of large MPP systems. In contrast, large-scale MPP systems have addressed the problems related to scalability, but are limited by their use of custom components.

#### 3.1 Workstation Clusters

If we are to build workstation clusters with thousands of nodes, scalability of **all** system components is critical. Dependence on non-scalable technologies must either be bounded so that the limits of scalability are not approached, or, if bounding is not possible, the dependence must be eliminated.

Network technologies such as Ethernet, wide-area network protocols such as TCP/IP, and tools such as NFS and rsh have inherent scalability limitations that must be addressed. Scalability spans every aspect of a large-scale machine. Scalability must be provided to the application, not only through high-speed networking, but also through a scalable startup mechanism in which compute resources are allocated and processes are started on thousands of nodes.

The machine must also be managed and maintained in a scalable fashion. The complexity of administration should not increase significantly as the size of machine or the number of users increases. For large-scale clusters, efficient maintenance and management of the system is at least as important as efficient use of the system.

Usability of the machine is critical. Users should be able to interact with and use the machine without any specialized or intimate knowledge of the underlying components. That is, users should not be required to know the name of every node in the cluster. Nor should they be responsible for determining the allocation of the machine's resources.

Ideally, clusters should be able to support the same types of applications that MPP systems support. Current cluster systems support a small number of users and run a small subset of applications. In order to approach MPP systems, clusters must address the issues related to making all types of applications run well.

### **3.2 Large-scale MPP Systems**

The Sandia/Intel ASCI-red TFLOPS machine has proven to be one of the more technically successful efforts in massively parallel, high-performance computing. However, large MPP systems, like clusters, also have drawbacks. Despite its success, some shortcomings in developing these types of large-scale systems have been identified. Future high-performance computing platforms of this magnitude will have to address these issues.

Custom hardware components are quickly superseded by commodity components. Even though the TFLOPS machine is based on commodity processor technology, many of the hardware components of the machine were designed and built specifically for the TFLOPS. This significantly increases the cost of the machine, in money as well as time. The performance of the Pentium Pro processors on the TFLOPS machine was quickly overcome by that of the Pentium II and Alpha processors. Commodity networking technology will soon reach that of the custom TFLOPS high-speed network.

Volume vendors are not the best organizations to create niche products. High-performance, massively parallel scientific computing continues to be a small market for volume hardware and

software vendors. Companies that target the mass market are unwilling or unable to address the needs of the scientific community due to the lack of revenue.

Large system scalability requires specialized knowledge and research. The success of the TFLOPS machine was the result of ten years of research and development. Much of the knowledge of how to build systems that can scale to thousands of nodes has not yet permeated the high-performance computing community as a whole, and much research still needs to be done.

Large-scale systems must grow in size and capability over their lifetime. Otherwise, they risk becoming obsolete before they are fully functional. Hardware and software technology in the current arena of high-performance computing is evolving rapidly. Large-scale systems must be able to adapt and evolve with these new technologies in order to remain a valuable resource.

The applications that require high levels of compute performance will continue to grow in size, variety, and complexity. Large-scale systems will have to evolve with the changes that occur in the applications for which they are built. These systems must be able to adapt to whatever environments and capabilities applications require.

## 4 Computational Plant

The Computational Plant[24] project at Sandia National Laboratories addresses all of the above issues relevant to developing, using, and maintaining a massively parallel commodity-based cluster. We are combining the knowledge and research of previous and ongoing commodity cluster projects with our expertise in designing, developing, using, and maintaining large-scale MPP machines. Our goal is to provide a commodity-based, large-scale computing resource that meets the level of compute performance needed by Sandia's critical applications. We have proposed a new model for the creation of high-end capability resources. Cplants, build on the design of the TFLOPS system, from which several key concepts can be derived:

- Large systems should be constructed from independent building blocks.
- Large systems should be partitioned into conceptually separate components that provide specialized functionality.
- Large systems must have significant resources dedicated to system monitoring and system configuration.
- Independent building blocks can be partitioned by changing a small number of system characteristics.

- Partitions can interact through a limited number of capabilities.

## 4.1 Conceptual Design

In order for a Cplant to be integrated from pieces made available from a variety of vendors, it is necessary to plan an architecture that is easily expandable and integratable. The architecture consists of *scalable units* and a *support system*.

The definition of a scalable unit is intended to be as non-specific as possible in order to allow a variety of vendors to supply components that meet the criteria. Scalable units provide resources to the Cplant. The use of the partition model of resource provision[9] allows scalable units to provide a variety of types of resources, such as service, compute, I/O, and network. Most of the Cplant resources will be compute resources that can service distributed memory programs and, at a minimum, run an MPI process. At least one scalable unit must provide a service capability as a direct interface to users from where jobs can be started, monitored, and debugged. Scalable units can also provide specialized resources such as enhanced secondary or tertiary storage and enhanced network capability. A specialized resource may be available to applications running within a compute partition and/or to users through a service partition. Scalable units must also respond to the queries and commands defined for the support system. It is also desirable that a scalable unit allow for remote power-cycling to install or update any software on the system.

The boundary between the support system and the scalable unit is somewhat blurred. If scalable units do not provide sufficient support for control and query, then the support system may be extended to include a system support station (sss) that is “attached” to the scalable unit. In this way, the support system grows in a scalable manner with the number of scalable units, but the scalable units do not have to include support system functions by design.

The primary purpose of the support system is to bond scalable units together. Most of the functionality of the support system consists of querying and controlling scalable units. Typically the support system will encompass a superset of the functionality of any single scalable unit, since it must support all scalable units. The support system has two logical pieces, a system support network and a (usually pre-existing) local infrastructure.

The system support network provides the ability to configure, customize, monitor, maintain, and control the entire Cplant. The simplest hardware realization of a system support network might be a console and keyboard with physical connections to every component of the system. This is, however, not a scalable solution. A minimal hardware realization must include a special component within each scalable unit that serves as a proxy for all interaction with the scalable unit, a system console

that serves as the point of entry for system administrators, and a network connecting scalable unit components to the console. In general, some sort of broadcast medium is preferable. Fault tolerance through multiple paths in the network and multiple consoles is also desirable.

The following is an overview of some of the classes of service that the system support network must provide. Once a scalable unit has been attached to the system, it must be possible to configure it from the system console. It must be possible to monitor the health of all components in the system. It must be possible to collect information about the current state of the system. It must be possible for a scalable unit attached to the system to tell the system support network what resources it provides to the system.

The information collected above may be used in several ways. We consider at least the following items. It must be possible for a system administrator to access the system console and execute commands. The allocation of resources within the system will have to be managed by a distributed management system. System libraries, such as MPI, can be optimized if given detailed information about the system configuration. Tools such as debuggers and performance monitors could make use of some of the functionality of the system support network.

While it is undesirable for ordinary users to have access to the system support network, there are certain parts of the system status of interest to users. For example, the user might wish to see a representation of what the available resources are and who is using them.

In addition to the system support network, a local infrastructure must be available to the Cplant. Any computing resource that is not available at users' desktop is not really available. A Cplant must live within an existing local infrastructure or have one built for it. Some resources in the Cplant must provide services directly to the user, for interactive use or batch queue submission. These command interfaces require a network connection into the Cplant service partition. Most Cplant users will have workstations and access to shared file servers where code and data are stored. It is desirable that the Cplant service partition also have a mechanism to access these files. Files created by applications on the Cplant will most likely need to be examined externally, and data may need to be transferred to external storage systems such as HPSS or visualization servers.

## 5 Cplant Prototype

During the past year, Sandia has designed and constructed a prototype Cplant machine. This prototype consists of 96 nodes (with an additional 32 nodes at the Sandia California site). The following sections provide an analysis of the processes involved in developing this prototype.

## 5.1 Design Goals

The primary goal of the prototype was to construct a commodity cluster capable of scaling to provide a level of compute performance on the order of one hundred gigaFLOPS. The prototype was to eventually become a production-level quality machine that will support on the order of a hundred users. It was not designed to be a resource dedicated to a small number of users or applications. While other cluster projects have emphasized achieving low price/performance levels, this was not a paramount goal for this prototype. An effort was made to keep costs as low as possible, but, in some instances, the solution that was the most scalable or that performed better was chosen over the least expensive. An additional design goal was to replicate the familiar TFLOPS user environment so that application developers could interact with the cluster in the same manner as the TFLOPS machine. Constructing a large-scale prototype involved developing a set of requirements and evaluating possible candidates for different components. In the following sections, we enumerate the considerations we had for different parts of the system and the motivating factors behind the choices.

## 5.2 Hardware

In order to construct the prototype, we purchased several different hardware components from several different vendors. We examined the market for the workstations, networking hardware, remote power controllers, and hardware for remote console access.

The critical step in terms of hardware was to determine the workstation or PC to use as the basis for the cluster. This decision was made after extensive testing of several different configurations of processors, motherboards, clock rates, memory subsystems, and compilers. Both benchmarks and real application codes that attempt to measure integer and floating point performance, processor to memory bandwidth, PCI bus bandwidth, and overall FLOP rates were used. Other hardware issues were also considered, such as the number of available PCI slots, whether the machines were rack or stack mountable, and the amount of heat generated.

Additional important configuration requirements also needed to be met. The machines had to be able to boot via the standard BOOTP network protocol. The machines also had to be able to perform a headless boot. That is, the machines had to boot without a keyboard, monitor, or video card being attached.

After compiling and studying all of this data, we decided on the Personal Workstation 433a from Digital Equipment Corporation (DEC). These machines have the following features: a 433 MHz

21164 Alpha processor, 192 MB ECC SDRAM, a 2 MB L3 cache, integrated Fast Ethernet, and a 2 GB IDE disk. The machines came mounted in cabinets with eight machines per cabinet, and each cabinet was equipped with two internal power distribution units.

Nearly as important as the workstation choice, was the determination of the interconnection fabric for the cluster. At the start of the project, it was decided that Fast Ethernet provided insufficient bandwidth to support the needs of our target applications. For the TFLOPS, the bandwidth requirement was at least one byte per the peak FLOP rate of a single node. Since the nodes were capable of delivering 400 MFLOPS, the bandwidth requirement was 400 MB/s. Clearly this is not possible given the current performance differential of commodity processors and networking hardware. The requirements for the Cplant prototype bandwidth were relaxed to one bit per the peak FLOP rate of a single node, or about 100 MB/s. At the time of the purchase, only Myricom was offering a reasonably priced product capable of not only meeting this performance requirement, but that could also meet our scalability requirements. We purchased 32 Myrinet dual 8-port SAN switches and 128 Myrinet 33 MHz 32-bit PCI network interface cards.

The ability to remote power cycle individual nodes is an important capability for a cluster of this size. After surveying the market of available power controllers, we purchased units from Electronic Energy Control, Inc. These power distribution units can control up to 16 machines through a serial line.

Console access to individual nodes is also an important capability for a cluster of this size. The ability to access the console of a machine to perform diagnostics and testing, or to check that a machine boots properly is critical. The serial port on each node can be accessed through the system support station to enter the console. Each support station contains two 8-port Cyclades serial controller cards. More details about the serial network and support stations are presented below.

### **5.3 Operating System**

Linux was the preferred operating system choice from the start. The value of Linux in building commodity-based clusters has been well established by the Beowulf-class machines. It provides all of the desired functionality at extremely low cost. The loadable module feature facilitates efficient debugging and development. And most importantly, the source code is available, and support from the Linux community is invaluable.

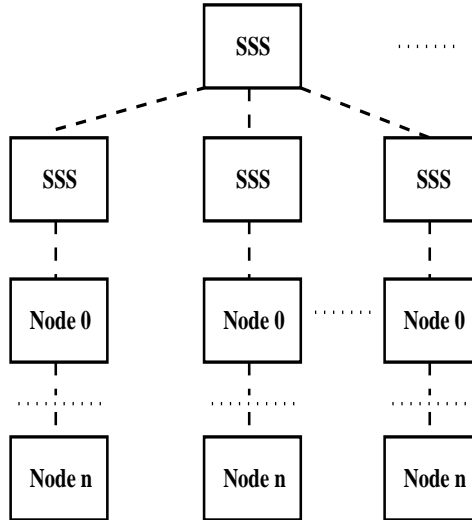


Figure 1: Diagnostic Network

## 5.4 Management Configuration

Each scalable unit for the prototype system consists of 16 nodes, or two cabinets. The number of nodes in a scalable unit is based on the consideration of the performance and scalability of NFS and BOOTP. Each cabinet is controlled by a first level system support station (sss-0). The sss-0 is responsible for serving kernels to the compute nodes via BOOTP over Ethernet, for providing console access to each compute node via serial lines, and for power-cycling each compute node through the serial-controlled power distribution units. Figure 1 shows the prototype configuration of the diagnostic network.

Simple shell scripts were written to perform all of the necessary management, diagnostic, and testing functions for a scalable unit. The only code written from scratch was the code to drive the power controllers. Booting a node is accomplished by a shell script that changes the `/etc/bootptab` entry for the node and then calls the power control program to power cycle the node. A console script that uses the standard UNIX call up utility (`cu`) was written to provide console access to each node.

Each compute node NFS mounts its root directory from the sss-0 machine. This mounting scheme allows the sss-0 to contain a single set of configuration information common to all of the nodes that it controls. Any information specific to each node can be easily separated from the common set.

The sss-0's also contain shell scripts for loading or unloading the communication and run-time environment software on the compute nodes. These scripts essentially perform operations via the remote shell utility from the sss-0 to each individual compute node.

The six first level system support stations are all connected to a second level system support sta-

tion, sss-1. The sss-1 machine is the single point of entry into the diagnostic network. Constructing the support network in this fashion allows for scalability in the support system. If more scalable units are added, an additional second level support station can be added. As more second level support stations are added, it may be desirable to have a third level support station. The support system tree is designed to grow taller as it grows wider.

## 5.5 System Administration Tools

Administration may be decomposed into three parts: discovery of new hardware, generation of configuration files and operations performed upon devices in the system. Our configuration management software handles all three of these aspects of the administration with a minimum of user intervention. It also consolidates all information regarding the layout and composition of the parallel machine into a single database for ease of updating and a consistent interface. All aspects of the configuration are reflected in this central database.

When a device is added to the system or new unit is swapped in place of a failed device, the configurator must probe the new device to discover its MAC address and any other salient details of the unit. After adding this information to the central database, certain configuration files must be updated to reflect this change. For instance, the `/etc/bootptab` on the device's boot host must be updated with the MAC address of the new device. `/etc/hosts` on all hosts should have the new device added and properly qualified as well. Myrinet maps may also need to be updated. This is tedious to do by hand and error prone due to the necessary duplication of information in disparate locations across the system.

The new device must also be configured to become aware of its location in the system and work with the other units. Terminal servers, for instance, require at least four parameters to be changed per port. SSS0 devices are even more challenging to do by hand, since an entire Linux installation is required. All of this would be most difficult to remember how to do for every single device type in the system and impossible to do in a reasonable length of time should the entire machine require a reconfiguration. This aspect is also automated by the configurator.

Lastly, the database must supply the configuration information to tools that will operate on the device in the system. The power control and console tools that must be able to query the database to retrieve information regarding the physical rack and unit of a device to determine which power controller or terminal server handles a given node.

Our implementation is a collection of Perl scripts that build the configuration database in memory and operate on devices, groups and racks represented as Perl objects. Each object has methods

related to system management: `discover()`, `configure()`, `power()`, `console()`, and so on. The rules for generating hostnames, IP address assignments and such are handled by user-supplied templates. The system is as generic as possible to allow for design changes in the computational plant to be easily tested without requiring rewriting the configuration code. It is also fairly easy to move to new hardware systems or layouts with out requiring much rewriting – new power controllers or different terminal servers could be used side-by-side with the old hardware as soon as configuration scripts for the new devices were written. Flexibility is a key aspect of the configurator.

## 5.6 High-Performance Message Passing Layer

An initial assumption for this project was that the data movement layer used on the TFLOPS machine was a key component in scaling a system to thousands of nodes. The entire design of the lightweight kernel for the TFLOPS machine was built on this low latency, high performance, predictable message passing layer.

An implementation of portals in Linux was already under development as part of the Unified Kernel[12] project in which Linux was being run on nodes of the TFLOPS machine. However, this implementation of portals was designed to run over Ethernet rather than Myrinet.

Several choices for low-level message passing layers on top of Myrinet were available from commercial vendors as well as research institutions[17, 16, 15, 13, 21, 11]. Many of these were removed from consideration due to factors such as the inability to support multiple processes per node, the lack of MPI support, and the lack of support for AXP Linux.

We chose to continue the development of portals in Linux for several reasons. First, it gives us a chance to continue our research with a communication methodology that had been successful on the TFLOPS machine. It also allowed us to re-use much of the code that had been developed for the TFLOPS machine and the Unified Kernel project, including a high-performance MPI library[4, 3] that had been validated by Intel. We were also able to begin code development on portals over Ethernet before obtaining Myrinet hardware. Portals also provides us with a familiar low-level message passing interface to which all of our system tools and libraries can be written, allowing a layer of abstraction that can adapt to emerging network technologies[5, 30].

## 5.7 TFLOPS Environment

A key feature of the Cplant prototype machine is that the user environment is nearly identical to the TFLOPS user environment. From the user’s standpoint, we wanted to the machine to be familiar to

those who had experience using the Paragon and TFLOPS machines. The following describes some the characteristics of the Paragon and TFLOPS machines that we provided.

The function of *yod* on the prototype Cplant machine is essentially identical to that of the TFLOPS machine. Yod accepts arguments that tell it how many nodes are being requested and the executable(s) to be launched. It handles UNIX standard I/O in the same manner as its TFLOPS counterpart. It also redirects UNIX signals sent to it to the compute node processes. Sending a SIGKILL to yod will kill the compute node processes.

The Process Control Thread, or *PCT*, runs on each compute node. This heavyweight daemon process is responsible for controlling the resources on a compute node<sup>2</sup>. It handles starting and terminating the user process, redirection of UNIX signals to the user process, and provides the user process with the environment needed to be part of an application.

*Bebopd* is a daemon process that runs in the service partition. It handles the allocation and status of the available compute nodes. It keeps track of which nodes are available, which nodes are allocated, and to whom the nodes are allocated. Bebopd is essentially the resource manager for all of the compute nodes in the system.

*Pingd* is analogous to the TFLOPS showmesh utility. It displays the status of the machine, including which nodes are free, which nodes are allocated, which job is running on each node, how long the job has been running, and who owns the job . Typically users will run pingd to discover the number of available nodes and to verify that a job has been launched and is running.

An additional component of the runtime environment gets linked into an application through a set of system libraries. These libraries redirect most UNIX system calls back to the controlling yod process. This mechanism allows the environment in which the yod process is running to be reflected to the compute node processes. For example, I/O calls, such as `open()`, and standard library calls, such as `getpid()`, are linked into the application as RPC calls to yod. In this way, the transparency of the remote processes is not compromised, and the appearance of a single environment is given to all processes in the application.

In addition to the familiarity of this environment, it is also designed to be scalable. Communication between PCT's participating in loading a job or disseminating signals can be implemented as a spanning tree to significantly decrease the amount of time needed to broadcast information to all nodes. Since all information concerning the user's environment is given to the PCT's, and since yod handles most system calls, there is no need to have a complete environment for every user on every

---

<sup>2</sup>The PCT is somewhat of a misnomer due to the fact that the equivalent entity for the operating system on the TFLOPS machine is a thread rather than a heavyweight process

node. Only service nodes require a complete user environment. This scheme significantly decreases the number of nodes that NFS mount user home directories and significantly decreases the amount of work needed to administer the machine.

The compile environment for the prototype machine is a “cross” compile environment. AXP Linux is almost binary compatible with executables produced by Digital UNIX compilers. Since Digital UNIX compilers are more mature than the GNU AXP Linux compilers, it was highly desirable to be able to compile compute node applications with the high-performance Digital UNIX compilers and libraries. The compile environment for the prototype system builds statically-linked Digital UNIX ECOFF executables that then run under AXP Linux. In addition to the high-performance compilers and libraries, using the Digital UNIX compilers helps to avoid requiring application developers to construct a new build environment specifically for GNU compilers and Linux. A large percentage of current Sandia applications already have build environments for the Digital UNIX compilers. Most all of the FORTRAN codes require the Cray pointer extensions that the GNU FORTRAN compiler does not yet support.

The prototype compilers are shell scripts that call the corresponding Digital UNIX compiler with the appropriate paths to find the system libraries and header files. The compilers are available in a central location on the LAN so that any user with a Digital UNIX machine can access them. Additionally, there is a dedicated compile server that all users can access. The compile environment has also been distributed to users who do not have direct access to the compile server so that they can build executables on their own Digital UNIX machine and then move these executables to the prototype machine. This results in a high-performance, flexible compile environment to which the TFLOPS users have become accustomed.

## 6 Cplant Experience

In constructing the 96-node Cplant prototype, we successfully met three of the significant challenges facing developers of large-scale clusters. We built a production-quality, multi-user environment. We implemented a traditional MPP programming environment on a cluster of workstations. We constructed a scalable support structure that enables system administration activities by providing a single point of control. To date, we have not achieved our performance goals. However, based on our experience in developing system software for large-scale MPP systems, we are confident that we will be able to bring the current performance up to meet application’s needs.

## 6.1 Production Quality Multi-user Environment

The Cplant prototype machine has been available for use by application developers since March of 1998. In that time, more than 20 developers have run a wide variety of applications on the machine. Despite the availability of several larger and more mainstream platforms at Sandia, we have had as many as six different users and 50 jobs run on a given day.

In order to support this user community, we have provided man pages, an FAQ, and a mail alias for support. The man pages are available both through the UNIX man command and on our web pages. These pages describe how to use specific tools, e.g., the compilers, yod, and pingd. The FAQ provides more global context for using the Cplant and comparisons to the TFLOPS environment. The Cplant FAQ answers questions that were common for the TFLOPS machine. The cplant-help mail alias is available to answer questions that are not covered in the man pages or FAQ, and includes members of the Cplant development team.

Like the TFLOPS system, the Cplant system is integrated into Sandia's extensive computational facilities. The Cplant hardware is accessible throughout Sandia at both the New Mexico and California campuses. Users can log in to Cplant service nodes, access files in their home directories, compile their programs, launch jobs, and monitor progress from their desktop workstations. Users who have access to a Digital UNIX system and who require frequent compilations can install a full-functional development environment on their local workstations.

## 6.2 Porting Applications to Cplant

Despite the large decrease in both actual and relative network performance versus the TFLOPS machine, almost all of Sandia's production and research codes have been ported by the applications developers to the Cplant. Most application developers understand that Cplant-like machines are the inevitable future. Many of Sandia's codes, such as CTH (a high-speed shock-physics package, based on solving Lagrangian equations) and Salvo (a wave-equation-based, 3D, prestack, seismic imaging code) are explicitly designed to run on a wide variety of platforms, including the TFLOPS machine and networks of workstations. These codes, as one would expect, were quickly and easily ported to the Cplant system. Other codes, such as MPQuest (a materials structures code) and MP-SALSA (a chemically reacting flows code) have been highly optimized for the TFLOPS machine and have been finalists for the Gordon Bell prize for innovative use of MPPs. These codes also ported quickly to the Cplant system, since it mirrors the programming environment provided by the TFLOPS machine. Work continues to examine how these codes can best be tuned for the Cplant processor/network

imbalance.

A newer class of codes at Sandia involves the optimization of physical designs. Sometimes this means the iterative calling of simulation packages with varied parameters in order to map out the space of possible safety problems (e.g., the RSM/TEMPRA code) and other times this means the application of sophisticated techniques such as genetic algorithms or mixed local/global search schemes (e.g., the DAKOTA system). New codes, such as PICO, are being developed to apply Mixed-Integer-Programming to scheduling and design problems. All of these codes appear to be suitable for use on Computational Plants. As noted above, the coupling to parallel visualization codes will be essential to large-system success. The PMESA team has implemented their parallel version of the popular MESA package on the Cplant system. A parallel volume rendering project is also in progress.

### 6.3 System Administration

As part of our system administration support, we have developed tools that make it easy to manage user accounts, integrate new hardware, upgrade system software, isolate faulty hardware, and reserve a portion of the machine for special purposes.

Managing user accounts for the entire system is possible by modifying a single password entry in a master file. Changes to this file are automatically propagated to all of the service nodes in the cluster. Adding a new user takes less than one minute.

New hardware can be integrated using the discovery process described in the previous section. When a scalable unit is added to the cluster, the system support station (sss0) of the scalable unit obtains the necessary administrative environment from the sss-1 using remote distribution (rdist). Once the administrative environment has been built, the sss-0 discovers and configures all of its components. Integrating a new scalable unit takes approximately five minutes.

Upgrading the system software is analogous to adding new scalable units. The system administrator updates the system software on all six sss-0's and then executes the command to modify the `/etc/bootptab` and power-cycle the nodes. Note that in the 400-node system with 25 scalable units, we are enhancing the support software to automatically distribute the new version of the system software.

Upon report of faulty hardware, the system administrator can logically remove faulty nodes from the system by modifying a single resource file. The `fping` utility[27] makes it possible to ping all nodes in parallel (96 nodes in less than 5 seconds), thereby identifying faulty nodes.

Like the NOW developers, We have found that it is essential to be able to reserve a portion of the

system for special purposes such as development of system software, benchmarking, and servicing high priority users. Reserving nodes is another feature of our tools for isolating nodes and upgrading system software.

## 7 Lessons Learned and Future Directions

While our prototype system addressed many of the essential goals facing the development of large-scale clusters, this system also uncovered a number of smaller problems. In addition, we were presented with new requirements as the system was being developed. In July of 1998, Sandia purchased 400 more machines to integrate into the existing prototype Cplant system. In the design of the new portion of the system, we addressed the shortcomings and inefficiencies identified by the prototype.

### 7.1 Problems Identified

Experiences with the prototype machine identified several problems with the system. Most are related to the fragility of the individual hardware components of the system and the redundancy needed to keep the system running in the event of hardware failures. Myrinet switches and power controllers required careful handling to avoid potential problems. Portions of the system, for example serial lines, provided insufficient redundancy. Integration was hindered by the large number of cables exiting each cabinet. In general, while it was possible for the Cplant team to integrate the system, it was determined that simpler, better documented integration procedures were necessary.

The dual 8-port switches required special care because they were not rack mountable and the cable connectors could not be locked on the ports of the switch. These factors limited the amount of movement that a cabinet could withstand without disconnecting some of the cables. The Myrinet cables were also very delicate. The cables are fragile and do not have the flexibility or durability of Ethernet cables. SAN cables cannot exceed ten feet in length. This restriction limited the inter-cabinet topology.

Since they were also not rack mountable, the power controllers were placed on the floor in the back of each cabinet. This placement made it difficult to visually inspect the power status of individual nodes. It was also difficult to access the breaker switches. In some instances, the breaker switches were accidentally triggered by moving the power controllers.

The use of the sss-0 as an attachment point for console serial lines and power controller serial lines prevented redundant control of these basic functions. Loss of an sss-0 resulted in the loss

of control over an entire scalable unit. This configuration also limited the distance between each scalable unit and its controlling sss-0.

This method of attaching serial cables to sss-0's contributed to the difficulty of wiring the machine. Each cabinet had an average of 25 cables leaving the chassis (9 serial, 8 Ethernet, up to 8 Myrinet, 2 power). Reducing the number of cables would make it easier to diagnose wiring problems and to maintain cabinets.

Early in the integration process, it became clear that we were not well suited to this task. Self-integration may make sense for small systems. However, the common use of systems of this sort will require that integration be performed by people with the appropriate expertise. Sandia has encouraged the establishment of companies who specialize in this type of integration.

## 7.2 Additional Requirements

Some new considerations needed to be addressed for a larger system. While it was relatively easy to find floor space for thirteen cabinets, finding space for a larger number required additional planning and evaluation of available space. Thus, more effort was dedicated to logistics and cabinet placement. An increase in the number of nodes also would increase the amount of maintenance for the machine. We wanted to take any steps possible to increase the stability and reliability of the system. Also, we wanted to increase the ability to diagnose and test individual cabinets.

We were also asked to consider how to provide a machine that could be switched between classified and unclassified computing. The hope was to replicate the capability of the TFLOPS machine which can be switched between these modes in about 20 minutes. Part of the TFLOPS machine is dedicated to classified applications and data, another part is dedicated to unclassified, and another can be switched to dedicate more compute resources to either part. DOE has strict guidelines regarding the use of machines for classified computing. For example, machines that are performing classified computations must not only be physically disconnected from any unclassified machines, but there is also a minimum distance requirement. Additionally, any machine that had been executing in a classified mode must be scrubbed before being used in an unclassified mode. This means that disks as well as memory must be erased and overwritten. Because compute nodes of the TFLOPS machine are diskless, the time needed to switch between classified and unclassified modes is significantly reduced. We wanted to replicate this design as much as possible in the design of the new machine.

### 7.3 Second Generation Design

We believe our original design of a scalable unit was sound. However, the implementation with the prototype hardware was inadequate for several hundreds of nodes. For this new portion of the machine, we designed the contents of a single cabinet to facilitate integration by Digital Equipment Corporation.

The Myrinet switches for the new portion are rack mountable and contain 8 ports for LAN connectors and 8 ports for SAN connectors. The delicate SAN cables are used within the cabinet, while the more rugged LAN cables are used between cabinets. The length limit on the LAN cables is 35 feet, which significantly increases the distance allowed between cabinets. All of the connectors on the switch are locking. The new switch also contains an Ethernet port for diagnostics and control through familiar SNMP mechanisms. The features of this new switch allowed us to design a new network topology which has increased scalability and performance characteristics.

The power controllers for this new piece of the machine are rack mountable and are controlled via Ethernet rather than serially. Ethernet allows for control from longer distances and eliminates the need for multiple serial lines entering each cabinet.

The serial lines from each machine within a cabinet for console access are now connected to an Ethernet terminal server mounted inside the cabinet. This eliminates the need for serial lines entering the cabinet and allows for greater distances between the controlling sss-0 machine and the scalable unit.

Since the Myrinet switch, the power controllers, and the terminal server all have Ethernet capability, a rack mounted Ethernet hub was placed in every cabinet.

The individual nodes were also upgraded. The new nodes are 500 MHz Miatas with an upgraded PCI chipset. This new chipset fixes some of the problems with the older chipset and significantly improves PCI performance. The compute node machines are also diskless, to increase reliability and to better enable switching between classified and unclassified computing modes.

There are many benefits of the new hardware and cabinet layout. Cabinet layout is now more flexible, with increased distance limitations and reduced restrictions on the position of the sss-0 relative to the nodes it controls. This increased flexibility allows for a better network topology and avoids many of the problems that made the prototype system sensitive to failures.

The new internal cabinet layout results in significantly fewer cables coming into and leaving a cabinet. This new layout has a single Ethernet cable for the cabinet hub, a single power cable for the cabinet power distribution unit, and eight Myrinet LAN cables for the Myrinet switch. Reducing

the number of cables increases the stability of the cabinets and makes testing and diagnostics easier. The internal workings of each cabinet can be tested by connecting the Ethernet line to a laptop computer running the sss-0 maintenance and diagnostic software. This method of evaluating a cabinet is convenient for installation and off-line testing. When integrated into the system, these same functions are available through the controlling system support station.

The new internal cabinet layout also attempts to increase the reliability of the system, since the dependence on serial lines has been reduced. The terminal server that controls the console for the nodes in a cabinet can be reached by multiple machines. If the sss-0 responsible for a particular cabinet goes down, the terminal server can still be reached from other sss-0 nodes. The same is true for the power control units. Only the failure of the relatively well studied, robust, common components—the Ethernet hub, the terminal server, or the power control units, results in loss of management control for a cabinet.

## 7.4 Future Directions

Over the last year, we have concentrated much effort into designing a system that can scale to thousands of nodes. Less effort has been devoted to developing the system software to meet the high standards of performance and predictability to which Sandia users are accustomed.

We are currently defining a new set of requirements for a system-level message passing interface specification. Our initial experience with portals for the prototype machine identified areas where performance could be improved and functionality could be extended. The current interface to portals does not allow for a true OS bypass mechanism. This new specification will allow for more flexibility in the placement of message passing data structures, that in turn will help to decouple the compute node processor from the network. We expect to have a complete set of requirements as well as an initial API completed soon.

We are also working on modifications to the Linux kernel to increase communication and computation performance. Specifically, we are investigating the utility of running an application process in physically contiguous regions of memory. Currently in Linux, application message buffers can span several regions of virtual memory, whereas physically contiguous message buffers can be identified by a single address-length pair. We are also investigating modifications to the Linux scheduler to increase the amount of processor resources dedicated to the application process.

We are also working on a distributed resources architecture for the runtime tools. This will allow the different software parts of the runtime system to work more effectively and scalably. For example, the current prototype has a single resource allocation daemon. If this process faults,

the entire machine is unusable. In the new architecture, the resource allocation processes will be distributed throughout the service partition so that a failure will only affect a small portion of the machine.

## 8 Summary

In this paper, we have presented our approach to building a large-scale, massively parallel computing resource from commodity components. We have identified several areas in which current cluster technology falls short of attaining the level of performance, scalability, and maintainability necessary to compete directly with large MPP systems. Conversely, we have also identified several areas where large MPP systems fall short in adapting to evolving commodity technologies. The Cplant concept that we have presented is a hybrid approach that merges the knowledge and research of commodity cluster projects with Sandia's experience and expertise in large-scale MPP systems. We have designed and built a prototype system that has demonstrated the ability to provide both performance and scalability, and the implementation of the prototype machine has been refined to produce a more reliable large-scale system that can provide Sandia with a production-quality computing resource.

## Acknowledgments

The authors would like to acknowledge the contributions of the entire system software development team at Sandia, including Bill Davidson, Jim Otto, John van Dyke, and David van Dresser. Gratitude is also expressed to the distributed computing group at SNL/California, especially Rob Armstrong, Robert Clay, Joe Durant, David Evensky, John Laroco, Alan Williams, and Pete Wyck-off. We would like to thank Brian Bray, Doug Doerfler, Steve Gossage, Luis Martinez, and Tom Pratt for their help with Cplant networking. We also acknowledge the help of many application developers, including Mark Sears, Guylaine Pollock, Steve Plimpton, Joe Kotulski, James Peery, Mark Christon, and Eliot Fang. We also would like to recognize the contributions of Mike McConkey and Len Stans. Finally, we would like to express appreciation to Art Hale for his vision and to Bill Camp and Dona Crawford for their support in this effort.

## References

- [1] ASCI. *ASCI PathForward Program Description*, 1997. <http://www.llnl.gov/asci-pathforward/pf-desc.html>.
- [2] N. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su. Myrinet-a gigabit-per-second local-area network. *IEEE Micro*, 15(1):29–36, February 1995.
- [3] R. Brightwell and D. S. Greenberg. Experiences Implementating the MPI Standard on Sandia's Lightweight Kernels. Technical Report SAND97-2519, Sandia National Laboratories, August 1997.
- [4] R. Brightwell and L. Shuler. Design and implementation of MPI on Puma portals. In *Proceedings of the Second MPI Developer's Conference*, pages 18–25, July 1996.
- [5] Compaq, Microsoft, and Intel. Virtual Interface Architecture Specification Version 1.0. Technical report, Compaq, Microsoft, and Intel, December 1997.
- [6] D. Culler, A. Dusseau, S. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Parallel Programming in Split-C. In *Proceedings of SC'93*, 1993.
- [7] D. E. Culler, A. Arpaci-Dusseau, R. Arpaci-Dusseau, B. Chun, S. Lumetta, A. Mainwaring, R. Martin, C. Yoshikawa, and F. Wong. Parallel Computing on the Berkeley NOW. In *Proceedings of the 9th Joint Symposium on Parallel Processing*, 1998.
- [8] D. P. Ghormley, D. Petrou, S. H. Rodrigues, A. M. Vahdat, and T. E. Anderson. GLUnix: a Global Layer Unix for a Network of Workstations. *Software: Practice and Experience*, 1997.
- [9] D. S. Greenberg, R. Brightwell, L. A. Fisk, A. B. Maccabe, and R. Riesen. A System Software Architecture for High-End Computing. In *Proceedings of SC'97*, 1997.
- [10] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, September 1996.
- [11] G. Henley, N. Doss, T. McMahon, and A. Skjellum. BDM: A Multiprotocol Myrinet Control Program and Host Application Programmer Interface. Technical Report MSSU-EIRS-ERC-97-3, Mississippi State University, May 1997.
- [12] D. Katramatos, S. Chapin, P. Hillman, L. A. Fisk, and D. van Dresser. Cross-Operating System Process Migration on a Massively Parallel Processor. Technical report, University of Virginia, January 1997.
- [13] M. Lauria, S. Pakin, and A. Chien. Efficient Layering for High Speed Communication: Fast Messages 2.x. In *Proceedings of the IEEE International Symposium on High Performance Distributed Computing*, 1998.
- [14] A. B. Maccabe, K. S. McCurley, R. Riesen, and S. R. Wheat. SUNMOS for the Intel Paragon: A brief user's guide. In *Proceedings of the Intel Supercomputer Users' Group. 1994 Annual North America Users' Conference.*, pages 245–251, June 1994.
- [15] A. Mainwaring and D. Culler. Active Message Applications Programming Interface and Communication Subsystem Organization. Technical report, Computer Science Division, University of California at Berkeley, September 1996. <http://www.cs.berkeley.edu/AM/am-spec-2.0.ps>.
- [16] Myricom, Inc. The GM Message Passing System. Technical report, Myricom, Inc., 1997.
- [17] Myricom, Inc. The Myrinet API. Technical report, Myricom, Inc., 1997.
- [18] Ohio Supercomputing Center. *LAM/MPI Parallel Computing*, 1998. <http://www.mpi.nd.edu/lam>.
- [19] S. Pakin. Personal communication, July 1998.
- [20] S. Pakin, M. Buchanan, K. Connelly, A. Lavery, G. Koenig, L. Giannini, J. Prusakova, G. Hermann, and A. Chien. *High-Performance Virtual Machines 1.0 User Documentation*, August 1997. <http://www-csag.cs.uiuc.edu/projects/hpvm/doc/hpvm.doc.html>.
- [21] L. Prylli. BIP Messages User Manual for BIP 0.94. Technical report, LHPC, June 1998.
- [22] S. H. Rodrigues, T. E. Anderson, and D. E. Culler. High-Performance Local Area Communication With Fast Sockets. In *Proceedings of USENIX'97*, 1997.
- [23] Sandia National Laboratories. *ASCI Red*, 1996. [http://www.sandia.gov/ASCI/TFLOP/Home\\_Page.html](http://www.sandia.gov/ASCI/TFLOP/Home_Page.html).
- [24] Sandia National Laboratories. *Computational Plant*, 1997. <http://www.cs.sandia.gov/cplant>.
- [25] Sandia National Laboratories. *Puma Portals*, 1997. <http://www.cs.sandia.gov/puma/portals>.
- [26] L. Shuler, C. Jong, R. Riesen, D. van Dresser, A. B. Maccabe, L. A. Fisk, and T. M. Stallcup. The Puma operating system for massively parallel computers. In *Proceeding of the 1995 Intel Supercomputer User's Group Conference*. Intel Supercomputer User's Group, 1995.
- [27] Stanford University. *Fping: A Program to Ping Hosts in Parallel*, 1998. <http://www-leland.stanford.edu/schemers/docs/fping/fping.html>.
- [28] T. Sterling, D. Becker, M. Warren, T. Cwik, J. Salmon, and B. Nitzberg. An Assessment of Beowulf-class Computing for NASA Requirements: Initial Findings from the First NASA Workshop on Beowulf-class Clustered Computing. In *Proceedings of IEEE Aerospace*, 1998.
- [29] V. Sunderam. PVM: A Framework for Parallel Distributed Computing. *Concurrency: Practice and Experience*, 2(4):315–339, December 1990.
- [30] Task Group of Technical Committee T11. Information Technology - Scheduled Transfer Protocol - Working Draft 2.0. Technical report, Accredited Standards Committee NCITS, July 1998.

- [31] University of California, Berkeley. *UC Berkeley Millenium Project*, 1997.  
<http://www.millennium.berkeley.edu>.
- [32] M. S. Warren, M. P. Goda, D. J. Becker, J. K. Salmon, and T. Sterling. Parallel Supercomputing with Commodity Components. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, 1997.