Cluster Computing and Applications

Mark Baker (University of Portsmouth, UK),
Amy Apon (University of Arkansas, USA),
Rajkumar Buyya (Monash University, Australia),
Hai Jin (University of Southern California, USA)

18th September 2000

1. Introduction

The needs and expectations of modern-day applications are changing in the sense that they not only need computing resources (be they processing power, memory or disk space), but also the ability to remain available to service user requests almost constantly 24 hours a day and 365 days a year. These needs and expectations of today's applications result in challenging research and development efforts in both the areas of computer hardware and software.

It seems that as applications evolve they inevitably consume more and more computing resources. To some extent we can overcome these limitations. For example, we can create faster processors and install larger memories. But future improvements are constrained by a number of factors, including physical ones, such as the speed of light and the constraints imposed by various thermodynamic laws, as well as financial ones, such as the huge investment needed to fabricate new processors and integrated circuits. The obvious solution to overcoming these problems is to connect multiple processors and systems together and coordinate their efforts. The resulting systems are popularly known as parallel computers and they allow the sharing of a computational task among multiple processors.

Parallel supercomputers have been in the mainstream of high-performance computing for the last ten years. However, their popularity is waning. The reasons for this decline are many, but include being expensive to purchase and run, potentially difficult to program, slow to evolve in the face of emerging hardware technologies, and difficult to upgrade without, generally, replacing the whole system. The decline of the dedicated parallel supercomputer has been compounded by the emergence of commodity-off-the-shelf clusters of PCs and workstations. The idea of the cluster is not new, but certain recent technical capabilities, particularly in the area of networking, have brought this class of machine to the vanguard as a platform to run all types of parallel and distributed applications.

The emergence of cluster platforms was driven by a number of academic projects, such as Beowulf [1], Berkeley NOW [2], and HPVM [3]. These projects helped to prove the advantage of clusters over other traditional platforms. Some of these advantages included, low-entry costs to access supercomputing-level performance, the ability to track technologies, an incrementally upgradeable system, an open source development platform, and not being locked into particular vendor products. Today, the overwhelming price/performance advantage of this type of platform over other proprietary ones, as well as the other key benefits mentioned earlier, means that clusters have infiltrated not only the traditional science and engineering marketplaces for research and development, but also the huge commercial marketplaces of commerce and industry. It should be noted that this class of machine is not only being used as for high-performance computation, but increasingly as a

platform to provide highly available services, for applications such Web and database servers.

A cluster is a type of parallel or distributed computer system, which consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource [4], [45]. The typical architecture of a cluster computer is shown in Figure 1. The key components of a cluster include, multiple standalone computers (PCs, Workstations, or SMPs), an operating systems, a high performance interconnect, communication software, middleware, and applications. All these components, their functionality along with representative examples are discussed in the remaining sections of this chapter.

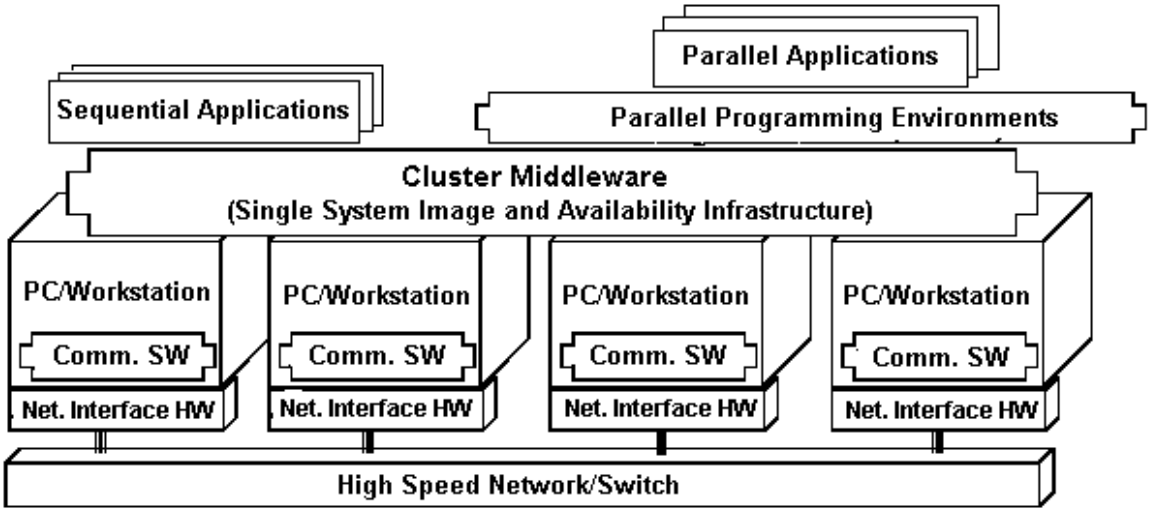

Figure 1. A Cluster Architecture. (Source [45]).

This chapter of the encyclopaedia is divided broadly into two parts. In the first part (sections 2-8) we explore and discuss the hardware and software components that make up a cluster. In the second part (section 9) of the chapter we discuss five cluster-based applications. Sections 2 and 3 look at networking and operating system solutions. In section 4 we discuss and explain why the concept of having a Single System Image is so important to clusters. In section 5 we review a range of possible middleware technologies that may be used to provide a range of services to applications. In section 6 we briefly outline the current parallel I/O solutions that are being used with clusters. Section 7 deals with the important area of high availability and how this is becoming an increasingly popular role for clusters. In section 8 we sketch out how tools and libraries for clusters are being adapted and developed for these heterogeneous platforms. Finally in section 9 we discuss a number of cluster applications, including a Web server, a speech-to-email service, a data mining service, an ad hoc network simulation, and an image rendering system.

2. Interconnection Technologies

A key component in cluster architecture is the choice of interconnection technology. Interconnection technologies may be classified into four categories, depending on whether the internal connection is from the I/O bus or the memory bus, and depending on whether the communication between the computers is performed primarily using messages or using shared storage [4]. Table 1 illustrates the four types of interconnections.

	Message Based	*Shared Storage*
I/O Attached	Most common type, includes most high-speed networks; VIA, TCP/IP	Shared disk subsystems
Memory Attached	Usually implemented in software as optimizations of I/O attached message-based	Global shared memory, Distributed shared memory

Table 1: Categories of Cluster Interconnection Hardware.

Of the four interconnect categories, I/O attached message-based systems are by far the most common. This category includes all commonly-used wide-area and local-area network technologies, and includes several recent products that are specifically designed for cluster computing. I/O attached shared storage systems include computers that share a common disk sub-system. Memory attached systems are less common, since the memory bus of an individual computer generally has a design that is unique to that type of computer. However, many memory-attached systems are implemented in software or with memory-mapped I/O, such as Reflective Memory [5]. Hybrid systems that combine the features of more than one category also exist, such as the Infiniband standard [6], which is an I/O attached interconnection that can be used to send data to a shared disk sub-system as well as to send messages to another computer. Similarly, SCI [7] may also be used for both shared memory and message passing.

The choice of interconnect technology for a cluster depends upon a number of factors, including compatibility with the cluster hardware and operating system, price, and performance. Performance is usually measured in terms of latency and bandwidth. Latency is the time to send data from one computer to another, and includes overhead for the software to construct the message as well as the time to transfer the bits from one computer to another. Bandwidth is the number of bits per second that can be transmitted over the interconnect hardware. Ideally, distributed applications are designed to minimise the amount of remote data that need to be accessed. However, in general, applications that utilise small messages will have better performance particularly as the latency is reduced and those that send large messages will have better performance particularly as the bandwidth increases. The latency in message passing is a function of both the communication software and network hardware. The remainder of this section describes protocols for cluster communication and provides a survey of available interconnection hardware products.

2.1 Protocols for cluster communication

Traditional networks use the standard Transmission Control Protocol/Internet Protocol (TCP/IP) [8]. The TCP/IP protocol is adequate for many cluster applications, and is the easiest to use when building a cluster with inexpensive Fast Ethernet products [8]. However, TCP/IP was designed for a wide-area network environment where errors are common. TCP/IP includes mechanisms to measure the round-trip time and attempts to optimise the number of bits that are sent in a single message, or before an acknowledgement is received. While these optimisations can reduce the number of times that a message must be resent in an error prone wide-area network, they increase the overall latency in cluster applications where errors are uncommon and the physical distance is small. For good performance, clusters generally utilize interconnection technologies that are specifically designed for cluster computing and that generally use specialized software and extra hardware for reducing the latency in messages.

Several academic research projects during the 1990's developed low-latency messaging protocols, including Active Messages [9], Fast Messages [10], VMMC [11], BIP [12], and Unet [13]. Active Messages (later extended to Generic Active Messages, or GAM) was the low-latency protocol used in the Berkeley Network of Workstations (NOW) project [2]. Active Messages allows messages to be sent from one computer to another without intermediate system buffering. Network hardware transfers the data directly from a pinned location in user memory to the network, and then to a pinned user location in user memory on the receiving side. Since no copies from user to system memory are required, this type of protocol is called a "zero-copy" protocol. In GAM, a copy sometimes occurs to a buffer in system memory on the receiving side so that user buffers can be reused more efficiently. Fast Messages extends Active Messages by guaranteeing that all messages arrive in order and reliably, even if the underlying network hardware is not reliable.

Virtual Memory-Mapped Communication (VMMC) is the communication protocol used in the Princeton SHRIMP system [11] (Scalable High-performance Really Inexpensive Multiprocessor system). SHRIMP is based on commodity processors and special communication hardware. VMMC works by mapping a page of user virtual memory to physical memory in the nodes of the cluster. The specially-designed hardware allows the network interface to snoop writes to local host memory and allows updates to take place on remote host memory. VMMC is an example of Distributed Shared Memory (DSM), in which memory is physically distributed among the nodes in the system, and falls in the category of memory-attached shared storage.

Basic Interface for Parallelism (BIP) is a low-level message layer for Myrinet [15] and was developed at the University of Lyon. In addition to blocking and non-blocking low-latency messaging, BIP provides multiplexing between Myrinet and shared memory under a single API, for users of clusters of symmetric multiprocessors.

The Unet network interface architecture allows zero-copy messaging where possible, and extends Active Messages by adding the concept of a virtual network interface. Each communication endpoint is viewed as a virtual network interface that is mapped to a real set of network buffers and queues on demand. The mapping requires system operations, but once the mappings are defined the communication can take place without requests to the operating system. The result is that communication can take place with very low latency.

The Virtual Interface Architecture [16] (VIA) standard combines the best features of various academic projects for low-latency messaging. VIA is a standard for low-latency communication that was developed with consortium that includes Compaq, Microsoft, IBM, and other industrial and academic institutions. The VIA standard has been adopted for use by all major cluster application vendors. In addition to products offered by hardware vendors, VIA implementations that are available include M-VIA [17], which is a modular VIA that runs in Linux, and Berkeley VIA [18], which is available for x86PC and Sparc processors, and Linux, Windows, and Solaris operating systems. Although VIA can be used directly for application programming, it is considered to be too low-level for most applications, and so most users write applications in a messaging passing language such as MPI [19] (Message Passing Interface) that runs over VIA.

A large consortium of industrial partners, including Compaq, Dell, Hewlett-Packard, IBM, Intel, Microsoft and Sun Microsystems support the Infiniband standard. The InfiniBand architecture replaces the shared bus that is standard for I/O on current computers with a high-speed serial, channel-based, message-passing, scalable-switched fabric. All systems and devices attach to the fabric through channel adaptors. A single InfiniBand link operates at 2.5 Gbps point-to-point in a single direction. Data is sent in packets, and six types of

transfer methods are available, including: reliable and unreliable connections, reliable and unreliable datagrams, multicast connections, and raw packets. In addition, InfiniBand supports remote direct memory access (RDMA) read or write operations, which allows one processor to read or write the contents of memory at another processor, and also directly supports IPv6 messaging for the Internet. Since InfiniBand replaces the standard shared bus, new equipment is needed to introduce InfiniBand into the network, although this can be done incrementally. Infiniband requires that new system software be implemented in the operating system of the host platforms, and that embedded logic be included within the enclosures and devices that attach to the fabric.

2.2 Network Topologies

In general, switched networks are preferred for cluster applications over shared implementations of the same type of network. Ethernet, Fast Ethernet, and some Gigabit Ethernet [20] products are based on the concept of a collision domain. When the network is a shared hub, all nodes connected to the hub share the same collision domain and only one computer on the network can send a message at a time. With a switched hub each computer is in a different collision domain. In this case, two sets of nodes may be communicating at the same time as long as they are not sending to the same receiver. For cluster computing, switched networks are preferred since it allows multiple simultaneous messages to be sent, which can improve overall application performance.

The topology of the network is generally determined by the selection of the specific network hardware. Some networks, such as Ethernet types of networks, only allow a configuration that can be viewed as a tree because of the hardware-implemented transparent routing algorithm. Other networks, such as Myrinet [26], allow a configuration with multiple redundant paths between pairs of nodes. In general, the primary factor with respect to topology that may affect the performance of the cluster is the maximum bandwidth that can be sustained between pairs of nodes in the cluster simultaneously. This is a function of both the topology and the capability of the switches in the network and depends on the specific network product that is chosen.

2.3 Network Hardware Products

Most clusters are built using a network technology that is attached to the I/O bus since this interface to the computer is well understood. The I/O bus provides, at least, a hardware interrupt that can inform the processor that data is waiting for it. How this interrupt is handled may vary, depending on the firmware or software stack that is used to receive the data. Networks that are built specifically for cluster computing have additional hardware support for lowering the latency in communication.

Networks that are traditionally used for local area and campus area networks, including Ethernet, Fast Ethernet, Gigabit Ethernet [20], and ATM [21], can provide basic functionality for small clusters. Higher-performance products in this category can be competitive for high-performance clusters. Networks that are designed for traditional applications generally have minimal additional hardware support specifically for cluster computing, but can be less expensive. The trade off is that the messaging latency may be somewhat higher.

Table 2 compares several commonly used networks products for high-performance clusters as of fall, 2000. Products are listed in alphabetic order across the table. Because new products are continually becoming available, this list will be incomplete, but is representative of the capabilities typically found for network products of this type.

	Gigabit Ethernet	Giganet	Myrinet	Qsnet	ServerNet2	SCI
Bandwidth (MBytes/s)	85	105	140-33MHz 215-66MHz	208	65	80
MPI Latency (us)	30-200	20-40	16.5-33MHz 11-66MHz	5	20.2	6
List price per port	$1.5K	$1.5K	$1.5K	$6.5K	$1.5K	$1.5K
Hardware Availability	Now	Now	Now	Now	Q2'00	Now
Linux Support	Now	Now	Now	Late'00	Q2'00	Now
Maximum #nodes	1000's	1000's	1000's	1000's	64K	1000's
Protocol Implementation	Implemented in hardware	Firmware on adaptor	Firmware on adaptor	Firmware on adaptor	Implemented in hardware	Firmware on adaptor
VIA support	NT/Linux	NT/Linux	Soon	None	In hardware	Software
MPI support	MPICH over MVIA, TCP	3rd party	3rd party	Quadrics/ Compaq	Compaq/3rd party	3rd party

Table 2: Comparison of Several Interconnection Products.

Gigabit Ethernet products are available for low-cost Linux and Microsoft Windows platforms.. Although Gigabit Ethernet does not have hardware support for VIA, M-VIA provides an emulated implementation of VIA for several Gigabit Ethernet adaptors and MPI is available with the MVICH [22] implementation over M-VIA.

Giganet [20] cLAN products were developed with the goal of supporting VIA in hardware. It has hot-pluggable links, automatic network mapping and node detection. Software support includes drivers for VIA and management tools for NT and Linux.

Myrinet [26] is one of the first networks developed for cluster computing. Myrinet technology has several features that make it ideal for cluster computing. Myrinet consists of high-speed, low-latency switches that can be connected in a hierarchical fashion. When connected using a fat-tree configuration [24], multiple paths between nodes can help to reduce contention on the links. Myrinet uses wormhole routing [23], which ensures that a message is delivered without buffering at intermediate nodes, and thus reduces latency across the cluster. Myrinet uses a slack buffer for flow control on each link. Unlike the flow control between links in ATM, bits are not dropped. Since most cluster applications require that data be delivered reliably between nodes, this improvement in reliability means that messages do not have to be resent, and the overall performance of the cluster is improved. Myrinet products have been available commercially since the mid-1990's and are a well-established cluster interconnection technology.

QsNet [27] is a high bandwidth, low latency interconnect for commodity symmetric multiprocessing computers. The network switches include a network processor with memory management unit, cache and local memory interface, and are connected in a fat tree. QSnet supports a true zero-copy (virtual-to-virtual memory) protocol, and has excellent performance.

ServerNet [28] has been a product from Tandem (now a part of Compaq) since 1995. ServerNet II offers direct support for VIA in hardware, 12-port non-blocking switches, and fat links between switches. Up to four physical links can be mapped into one logical link. Software support includes VIA drivers for NT and Linux. Although ServerNet II is a well-established project, it is only available from Compaq as packaged cluster solution, not as single components, which may limit its use in general-purpose clusters.

Scalable Coherent Interface (SCI) [29] was the first interconnection technology to be developed as a standard specifically for the purposes of cluster computing. The IEEE 1596 standard for SCI was published in 1992 and specifies the physical and data link layers of the network as well as higher layers for distributed shared memory across a network. Many commercial products are developed using the lower layers only. At higher layers, SCI defines a directory-based cache coherence scheme that implements distributed shared memory.

The Atoll [30], or Atomic Low Latency network, is one of the newest networks for cluster computing. Atoll has four independent network interfaces (NIs), an 8x8 crossbar switch and four link interfaces in a single chip. Message latency is as low as 4 μs, and bandwidth between two nodes approaches 200 Mbytes/s. Atoll is available for Linux and Solaris operating systems and supports MPI over its own low-latency protocol. The price per node is expected to be around $700, with product release sometime in 2001.

Shared storage systems are an important cluster interconnection for many commercial systems. The IBM Sysplex interconnection between mainframes is an example of an I/O attached shared storage system [1]. Also, many commercial clusters are interconnected via shared disk subsystems. These subsystems, often called Storage Area Networks (SANs), are often constructed with a combination of SCSI and Fibre Channel [31] interconnections. Fibre Channel is a set of standards that defines a high performance data transport connection technology that can transport many kinds of data at speeds up to 1 Gbps, through copper wire or fiber optic cables. Like SCI, the Fibre Channel standard is defined in layers, beginning at the physical layer, and many commercial products are developed using the lower layers only.

Except for what are really large symmetric multiprocessors, true memory-attached architectures are rare. The illusion of memory-attached interconnections can be created through software or using memory-mapped I/O on I/O attached interconnections. Reflective memory [32] is an example of a memory attached interconnection of this type in which nodes write to a local memory locations on the NIC, and have the data broadcast on the network so that it is available to all other nodes in the network. The broadcast of the data happens automatically, so that all nodes in the network have a shared view of the reflective memory. Reflective memory products have been available since the 1980's and are available from several vendors. Reflective memory is often used in a real-time fault-tolerant environment. In these environments, the replication of the data in the reflective memory gives a higher level of fault tolerance. Because each memory location has to be physically replicated on each node in the system, reflective memory products tend to be more expensive than other types of networks, and their cost grows as the size of the shared memory and the number of nodes in the system grows.

3. Operating Systems

The operating system in the individual nodes of the cluster provides the fundamental system support for cluster operations. Whether the user is opening files, sending messages, or starting additional processes, the operating system is always present. The primary role of an

operating system is to multiplex multiple processes onto hardware components that comprise a system (resource management and scheduling), as well as provide a high-level software interface for user applications. These services include protection boundaries, process/thread co-ordination, inter-process communication and device handling.

3.1. Operating System Features for Clusters

The desirable features of a cluster operating system include:

- *Manageability* – Intuitive facilities for local and remote administration; this is often associated with a Single System Image (SSI). These facilities can be realized at different levels, ranging from a high-level set of `Perl` management scripts, down to inter-node state sharing via operating system level extensions.
- *Stability* – An important characteristic is robustness against system crashes, failure recovery by dynamic reconfiguration, and usability under heavy load.
- *Performance* – Performance is critical to all facets of an operating system. Important parts of the operating system, such as memory management, process and thread scheduler, file I/O and communication protocols should work as efficiently as possible. A user, such as a programmer or system administrator, should be able to transparently optimise the relevant parameters to fine-tune the operating system for the varying demands of specific applications.
- *Extensibility* – An operating system should allow the easy integration of cluster-specific extensions, for example user-loadable device drivers.
- *Scalability* – The scalability of a cluster is primarily determined by the properties of the node interconnect. This may include support by the operating system for low-level high-performance communications protocols to access the network.
- *Support* – User and system administrator support is an underrated but an essential element to the success or failure of any computer system. Fast and timely support is crucial for the successful deployment of a cluster operating system.
- *Heterogeneity* – By its very nature a cluster consists of heterogeneous hardware components. An operating system is much more likely to be useful in a cluster if the same operating system can run across multiple architectures, or at least the operating systems can support a set of standardized APIs that simplify the development of middleware layers. Middleware layers enable the seamless usage of heterogeneous components across the cluster.

To date there has been little work on operating systems specifically for clusters. Most turnkey clusters are provided with versions of the companies' mainline products [4], usually with little or no modification. Generally, however, there may be some form of SSI integrated into a conventional operating system. Two variants are typically encountered, that for system administration and/or job scheduling purposes and that found at the system call level. The former is usually some type of middleware and enables each node to deliver the required services, whereas the latter may offer features like transparent remote device usage or to use a distributed storage facility that is seen by users as a single standard file system.

Another approach to sharing computational resources is via software-enabled distributed shared memory (DSM) [183]. This programming paradigm allows the distributed resources of the cluster be used in a similar fashion to a shared memory system, such as an SGI Origin. DSM is normally realised via user-level libraries [33] and not as a service offered by the operating system. Integration of DSM into an operating system has rarely been done [34], [35] as the performance for general-purpose use (requiring strict sequential consistency) is often to low. The use of a relaxed consistency models improves performance, but presents the possibility that data consistency may not be strictly correct.

Alternatively, Network RAM [36] is an approach to using memory on remote node. Here the operating systems manage the remote memory and take advance of current-day high-speed and low-latency interconnect such as Myrinet or SCI. Technologies such as these allow access to remote RAM faster then for local secondary storage. Network RAM offers the use of remote memory instead of the local hard disk for purposes like swapping or paging.

3.2. Heterogeneity

Clusters, by their very nature, are heterogeneous. A cluster may originally be only slightly heterogeneous, where, for example, memory or disk speeds may vary. As new nodes are added or old ones are replaced, the cluster may become moderately heterogeneous, when for example different generations of components, such as CPUs, are added. In the longer term, a cluster may be potentially highly heterogeneous, when for example it consists of mixture of different hardware and software components.

Heterogeneity in a cluster can cause problems. The lowest level on which heterogeneity causes problems is the data representation – big-endian vs. little-endian. If such systems are to be connected, the adaptation of the different representations should be done on the lowest level possible to gain suitable performance. Obviously the fastest approach to do endian conversion would be in hardware, but currently no hardware support exists. Middleware approaches to masking heterogeneous systems are currently the most common and successful.

Heterogeneous clusters may not perform as well as homogeneous clusters. In general, a parallel system performs best when all nodes are kept busy an equal amount of the time and communication is minimised. If one node finishes its work before the others, either because it was given less work to do, or because it runs at a faster speed, then the time that it spends waiting for the others to complete is time that it does not spend computing. Balancing the load of an application across a cluster that has homogeneous nodes is difficult, and becomes even more difficult when the nodes are heterogeneous.

3.3. Current State-of-the-Art

The single most popular cluster operating system for clusters is Linux [37]. This is primarily for three reasons:

- It is free;
- It is an open source. The source code is available and anyone is free to customize the kernel to suit ones needs;
- It is easy. A large user community of Linux users and developers have created an abundant number of tools, web sites, and documentation, so that Linux installation and administration is straightforward enough for a typical cluster user.

MOSIX [40] is a set of extensions for the Linux kernel that provides support for transparent process migration in a cluster environment. In this system a user can launch jobs on their home node, and the MOSIX will automatically load balance by potentially migrating jobs to lightly loaded nodes. Under MIOSIX a user sees a single process space, making the tracking of migrated jobs easy. Like Linux, MOSIX is free and is distributed under the GNU Public License.

For users who desire to use an operating system that has been developed commercially, several products are available. Commercial clusters often run proprietary operating systems, such as IBM's AIX, Sun's Solaris, SGI's IRIX, Tandem's Non-Stop Unix and Unixware 7,

Compaq's Tru64 and Windows NT/2000. Sun Microsystems has developed a multi-computer version of their Solaris operating system called Solaris MC [38]. Solaris MC incorporates some advances made by Sun, including an object-oriented methodology and the use of CORBA IDL in the kernel. Solaris MC consists of a small set of kernel extensions and a middleware library. Solaris MC provides a SSI to the level of the device. That is, processes running on one node can access remote devices as if they were local. Solaris MC also provides a global file system and a global process space.

3.4. Future

Symmetric multiprocessors (SMPs) are becoming more popular as cluster nodes. High-performance SMP clusters are generally connected via a low-latency and high-bandwidth interconnect. Federated Clusters of SMPs, connected by gigabit-speed wide-area networks, are also emerging with some recent Grid computing testbeds [41]. The nature of these Federated Clusters will have its impact on scheduling, both in terms of task placement and in terms of selecting a process to run from a ready queue. In this environment, scheduling, which is traditionally seen as an operating system activity, is carried out in middleware.

New operating systems for clusters may offer a minimalist approach by using micro-kernels. The Exokernel operating system is an example of a micro-kernel operating system [181]. With this approach to operating systems, only the minimal amount of system functionality is in the kernel. This allows the user to only load services when, and if, they are needed. In addition to maximizing the available physical memory by removing undesirable functionality, the user can alter the characteristics of the service. For example, a scheduler specific to the cluster application may be loaded that helps the application run more efficiently. Operating system micro-kernels bring up the issue of operating system configuration. In particular why provide a node operating system with the ability to provide more services to applications than they are ever likely to use? For example, a user may want to alter the personality of the local operating system, e.g. "strip down" to a minimalist kernel to maximize the available physical memory and remove undesired functionality. Possible mechanisms to achieve this range from a use of a new kernel to dynamically linking service modules into the kernel.

4. Single System Image (SSI)

The view of a distributed system as a single unified computing resource is often known as a Single System Image (SSI). This property hides the distributed and heterogeneous nature of the available resources and presents them to users as a single, powerful, unified computing resource [41]. The ability to realize SSI can be achieved through one or several mechanisms implemented in hardware or software at various levels of abstraction, from the kernel to application layers.

A system with SSI means that users have a system view of the resources available to them irrespective of the node to which they are physically associated. These resources can range from access and manipulation of remote processes to the use of a global file-system. Furthermore, SSI can provide a system with other features, including the ability to continue to operate after some failure (high availability), or the ability to ensure that the nodes are evenly loaded. Load balancing is provided through resource management and scheduling.

The design goals for SSI cluster-based systems are mainly focused on complete transparency of resource management, scalable performance, and system availability in supporting user applications [41], [45], [46], [47], [48]. The following are among some of the key SSI attributes that are generally considered desirable: point of entry, user interface, process

space, I/O and memory space, job-management system, and point of management and control.

The most important benefits of SSI [41] include:

- It frees the end-user from having to know where in the cluster an application will run;
- It allows the use of resources in a transparent way irrespective of their physical location;
- It offers the same command syntax as in other systems and thus reduces the risk of operator errors, with the result that end-users see an improved performance, reliability and higher availability of the system;
- It greatly simplifies system management and thus reduced cost of ownership;.
- It promotes the development of standard tools and utilities.

SSI can be realized in one or more of the following levels:

- Hardware – Systems such as Digital/Compaq Memory Channel [49] and hardware Distributed Shared Memory (DSM) [50] allow a user to view a cluster as a shared-memory system;
- Operating System – Achieved by either modifying/extending the operating system kernel (i.e., SCO UnixWare [47] and Sun Solaris-MC [38]) or building a layer (i.e., GLUnix [51]) that integrates the operating systems running on each node;
- Middleware – Use of this layer to provide SSI is common in present-day clusters. Middleware solutions include runtime and programming environments such as PVM [52] and job-management and scheduling systems such as CODINE [53] and Condor [54];
- Application – Application-level SSI is the highest and, in a sense, most important because this is what the end-user sees. At this level, multiple co-operative components of an application are presented to the user as a single application. For instance, a GUI based tool such as PARMON [55] offers a single window representing all the resources or services available. Another solution for providing SSI at this level is application-specific Problem Solving Environments (PSEs). These PSEs consist of all the tools and utilities required to develop, implement and run scientific and commercial applications, bound into one environment. The distributed nature of the runtime-environment is masked from the application developer [184].

It is important to remember that a good SSI is usually obtained by a co-operation between all these levels as a lower level can simplify the implementation of a higher one. The level at which SSI is developed, the interaction of SSI layers, and the degree of sophistication and complexity of SSI all affect its performance, flexibility, scalability and usability.

The use of SSI can greatly enhance the acceptability and usability of clusters by hiding the physical distribution and potential complex interaction of multiple independent computers by presenting them as a single, unified resource. SSI can be realised using hardware and/or software techniques, each of them have their own advantages and disadvantages. The designers of software for clusters must always consider SSI (transparency) as one of their important design goals in addition to scalable performance and enhanced availability.

5. Middleware

Middleware is generally considered the layer of software sandwiched between the operating system and applications. Middleware has been around since the 1960's and provides various services to applications. More recently, middleware has re-emerged as a means of integrating software applications that run in a heterogeneous environment. There is large overlap between the infrastructure that is provided to a cluster by high-level Single System Image (SSI) services and those provided by the traditional view of middleware.

Middleware has the ability to help a developer overcome three potential problems with developing applications on a heterogeneous cluster:

- The integration of software from different sources;
- Access to software inside or outside their site;
- Rapid application development.

In addition, the services that middleware provides are not restricted to application development. Middleware also provides services for the management and administration of a heterogeneous system.

5.1 Message-based Middleware

Message-based middleware is a technology that uses a common communications protocol to exchange data between applications. The communications protocol hides many of the low-level message passing primitives from the application developer. Message-based middleware software can pass messages directly between applications, send messages via software that queues waiting messages, or use some combination of the two. Examples of this type of middleware are the three upper layers of the OSI model [57], the session, presentation and applications layers. Other examples are DECmessageQ [58] from Digital, MQSeries [59] from IBM, and TopEnd [60] from NCR.

5.2 RPC-based Middleware

For many applications, the overwhelming numbers of interactions between processes in a distributed system are remote operations, often with a return value. For these applications, the implementation of the client/server model in terms of Remote Procedure Call (RPC) allows the code of the application to remain the same whether the procedures are the same or not. Inter-process communication mechanisms serve four important functions [61]:

- They allow communications between separate processes over a computer network;
- They offer mechanisms against failure, and provides the means to cross administrative boundaries;
- They enforce clean and simple interfaces, thus providing a natural aid for the modular structure of large distributed applications;
- They hide the distinction between local and remote communication, thus allowing static or dynamic reconfiguration.

Client and server do not execute in the same address space, so there are no global variables and pointers cannot be used across the interface. Marshalling is the term used for transferring data structures used in RPC [62] from one address space to another. Marshalling is required, as two goals need to be achieved: the data must be serialized for transport in messages, and the data structures must be converted from the data representation on the sending end and reconstructed on the receiving end.

Middleware tools built over RPC include Network Information Services [63] (NIS) and Network File Services [64] (NFS). Both NIS and NFS were originally developed by Sun Microsystems, but versions of each of them have been released into the public domain. NIS is a network naming and administrative tool for smaller networks that allows users to access files or applications on the network with a single login name and password. NFS allows files and directories to be exported from the server computer to client computers on the network. With NFS, users can have the same view of the file system from any computer in the network. With NFS and NIS, users can login; access an application that resides on the file

server, and save files into a home directory on the file server from any computer on the network.

5.3 Object Request Broker

An Object Request Broker (ORB) is a middleware that supports the remote execution of objects. CORBA (Common Object Request Broker Architecture) is an international ORB standard supported by more than 700 groups and managed by the *Object Management Group* [65] (OMG). The OMG is a non profit-making organization whose objective is to define and promote standards for object orientation in order to integrate applications based on existing technologies.

The *Object Management Architecture* (OMA) is characterized by the following:

- The Object Request Broker (ORB). The broker forms the controlling element of the architecture because it supports the portability of objects and their interoperability in a network of heterogeneous systems.
- Object services. These are specific system services for the manipulation of objects. Their goal is to simplify the process of constructing applications.
- Application services. These offer a set of facilities for allowing applications access databases, to printing services, to synchronize with other application, and so on.
- Application objects. These allow the rapid development of applications. A new application can be formed from objects in a combined library of application services. Adding objects belonging to the application can extend the library itself.

5.4 OLE/COM

The term COM has two meanings. It stands for *Component Object Model* [66], which form the object model underpinning Object Linking and Embedding (OLE) version 2.0. It also stands for the *Common Object Model* after an agreement between Microsoft and Digital. The first version of OLE was designed to allow composite documents (text and images) to be handled. The second version of OLE introduced a highly generic object model whose use can be extended well beyond the handling of composite documents. OLE2 offers a set of interfaces (Object Oriented) that allows applications to intercommunicate. The first version of OLE2 required applications to run on the same machine. OLE2 was later extended to run in a distributed fashion on Windows and UNIX platforms.

The COM model defines mechanisms for the creation of objects as well as for the communication between clients and objects that are distributed across a distributed environment. These mechanisms are independent of the programming languages used to implement objects. COM defines an inter-operability standard at the binary level in order to make it independent of the operating system and machine.

5.5 Internet Middleware

The Internet is based on the Internet Protocol (IP), which provides datagram services between devices that have an IP address, and the Transmission Control Protocol (TCP), which provides a connection-oriented reliable service over IP. While many applications use TCP/IP directly, a number of middleware-like technologies are built over TCP/IP. HyperText Transport Protocol (HTTP) allows text, graphics, audio, and video files to be mixed together and accessed through a web browser. The Common Gateway Interface (CGI) standard enables retrieved files to be executed as a program, which allows web pages to be created dynamically. For example, a CGI program can be used to incorporate user information into a

database query, and then display the query results on a web page. CGI programs can be written in any programming language.

Since users access some web pages and applications frequently, some web servers will send a small amount of information back to the web client to be saved between sessions. This information is stored as a “cookie”, and saved on the local disk of the client machine. The next time that the web page is accessed, the client will send the cookie back to the server. Cookies allow users to avoid retyping identification information for repeated sessions to the same server, and allow sessions to be restarted from a saved state.

5.6 Java Technologies

Java Remote Method Invocation [67] (RMI) allows communications between two or more Java entities that are located in distinct Java Virtual Machines (JVM). Java RMI is similar to RPC, except that Java RMI allows a Java applet or application access another remote object and invokes one of its methods. In order to access the remote object the calling application must obtain its address. This is obtained by access to a Registry where the object's name was registered. The Registry acts as a name server for all objects using RMI. The Registry contains a table where each object is associated with a reference – this is the object's interface and unique address of the object.

Jini [68] from Sun Microsystems, is an attempt to resolve the inter-operability of different types of computer-based devices. These devices, which come from many different vendors, may need to interact over a network. Jini is designed for a network in which devices and services are added and removed regularly. Jini provides mechanisms to enable the addition, removal, and discovery of devices and services attached to network. Jini also provides a programming model that is meant to make it easier for programmers to enable devices to interact.

Jini objects move around the network from virtual machine to virtual machine. Built on top of Java, object serialization, and RMI, Jini is a set of APIs and network protocols that can be used to create and deploy distributed systems that are organized as federations of services. A Jini service can be anything that is connected to the network and is ready to perform some useful task. The services can consists of hardware devices, software, communications channels, and even interactive users. A federation of services, then, is a set of services, currently available on the network that a client (meaning some program, service, or user) can bring together to help it accomplish some task.

In addition to Java RMI and Jini, Sun Microsystems has produced a plethora of Java-based technologies that can be considered as middleware [69]. These technologies range from the Java Development Kit (JDK) product family that consists of the essential tools and APIs for all developers writing in the Java programming language to APIs such as for telephony (JTAPI), database connectivity (JDBC), 2D and 3D graphics, security as well as electronic commerce. These technologies enable Java to interoperate with many other devices, technologies, and software standards.

5.7 Cluster Management Software

Cluster Management Software (CMS) is primarily designed to administer and manage application jobs submitted to workstation clusters. More recently CMS has also be extended to manage other underlying resources as well as software licenses (e.g. CODINE/GRD [53]). The job can be a parallel or sequential application that needs to run interactively or in the

background. This software encompasses the traditional batch and queuing systems. CMS can be used to help manage clusters in a variety of ways:

- Optimise the use of the available resources for parallel and sequential jobs;
- Prioritise the usage of the available resources;
- Manage mechanisms to "steal" CPU cycles from cluster machines;
- Enable check-pointing and task migration;
- Provide mechanisms to ensure that tasks complete successfully.

CMS software is widely available in both commercial and public domain offerings. There are several comprehensive reviews of their functionality and usefulness [71].

5.8. Factors in Choosing Middleware

The purpose of middleware is to provide services to applications running in distributed heterogeneous environments. The choice of which middleware may best meet an organization's needs is difficult as the technology is still being developed and may be some time before it reaches maturity. The risk of choosing one solution over another can be reduced if [72]:

- The approach is based on the concept of a high-level interface;
- The concept of a service is associated with each interface;
- The product conforms to a standard and supports its evolution;

In addition to these, a key criterion is ensuring that the middleware is interoperable with the other current and emerging middleware standards. Older software for providing middleware services, such as DCE, is being replaced by a new generation of object-oriented software such as CORBA/COM and Java, and these are being developed to be interoperable with each other. Jini, Sun Microsystems' latest distributed environment, is a promising middleware as it has been developed with the goal of interoperability.

6. Parallel I/O

The scalability of an application refers to its ability to effectively use additional resources as the size of the cluster and the application grow. In many cases, the scalability of a high performance application is more limited by the performance of the I/O system than by the performance of the CPUs in a cluster. Scaling such applications is not a case of merely increasing the number of processors, but also increasing the available bandwidth of the I/O sub-system. A cluster can potentially provide a large data storage capacity since each node in the cluster will typically have at least one disk attached. These disks can produce a large, common storage subsystem. This storage subsystem can be more effectively accessed using a parallel I/O system.

The general design goals of a parallel I/O system are [78]:

- Maximize the use of available parallel I/O devices to increase the bandwidth;
- Minimize the number of disk read/write operations per device;
- Minimize the number of I/O-specific operations between processes to avoid unnecessary and costly communication;
- Maximize the hit-ratio (ratio between accessed data to requested data), to avoid unnecessary data accesses.

6.1 Types of Parallel I/O Systems

Three types of parallel I/O systems can be identified [89]:

1. Application level;
2. I/O level;

3. Access anticipation methods

Application-level methods (based around so called buffering algorithms) try to organize the main memory and mapping the disk space (e.g., buffer) to make disk accesses efficient. Typically these methods are realized by runtime libraries, which are linked to the application programs. Thus, the application program performs the data accesses itself without the need for dedicated I/O server programs. Examples for this group are the Two-Phase method [74], the Jovian framework [56], and the Extended Two-Phase method [88].

The I/O level methods try to reorganize the disk access requests of the application programs to achieve better performance. This is done by independent I/O node servers, which collect the requests and perform the accesses. Therefore, the disk requests (of the application) are separated from the disk accesses (of the I/O server). A typical representative of this group is the Disk-directed I/O method [80].

Access anticipation methods extend the I/O operations into the time dimension, and are based around data pre-fetching. These methods anticipate data access patterns by using hints from the code in advance of its execution. Hints can be embedded by the programmer into the application or can be delivered automatically by appropriate tools (e.g., compiler). Examples for this group are informed pre-fetching [84], the PANDA project [75], and the Two-phase data administration [85].

6.3 Technological Issues

Several technological issues that arise in a cluster can have an impact in the performance of a parallel I/O system. These issues include interconnection latency, heterogeneity of the cluster, parameters of the I/O system, and the user need for optimisation.

The latency of the interconnection is not as much of a problem for I/O intensive applications as it is for computational bound applications. In general, a high-speed, low-latency network can turn the network bottleneck for clusters into the known I/O bottleneck [86]. In that case,, the already developed methods and paradigms to reduce this I/O bottleneck on massively-parallel systems can similarly be used on cluster systems.

A heterogeneous environment leads to the following issues for cluster I/O.

- Hardware Architecture – Each node can exhibit a unique hardware configuration (e.g., various types and numbers of disks, I/O bus and controller characteristics). To understand the behaviour of such systems requires complex analytical models and consequently makes understanding factors like performance prediction and system optimisation very difficult.
- Data Representation – Heterogeneous operating systems and processors can result in different physical representation of the primitive data types. This is not a problem so far if data is stored without semantics, but modern applications use this property (e.g. database tasks). Thus, respective conversion routines and/or a standardized I/O formats have to be implemented.
- Load balancing – Clusters support a multi-user, multi-tasking environments. These factors accompanied by the heterogeneous nature of the cluster hardware and software leads to the situation that the overall distribution of the workload of a cluster is hard to predict at any particular moment. The static approach to I/O planning is consequently almost useless.

The most crucial parameters for a parallel I/O system are:

- Distribution of data on disks;

- Data layout on disks;
- Latencies and transfer rates of media (disks);
- Communication network latency and transfer rate;
- Sizes of disk blocks, memory and communication buffers;
- Transfer rates for memory to memory copy;
- I/O scheduling;
- Caching, pre-fetching and write behind policies;
- Contention.

These parameters are essentially the same for clusters as for traditional computers. The (possible) heterogeneity in cluster systems, however, tends to increase the number of parameters dramatically. For example the characteristics of different disks may vary considerably. Further, the speed of communication may vary dependent on which nodes are communicating. Many of the parameter values (like latencies and transfer rates) are strongly influenced by the underlying hardware. Nonetheless, these parameters are not constant but can change significantly if other parameters are slightly modified. Given the huge number of possible parameter combinations, finding a near optimal I/O strategy for a specific application on a given hardware configuration is a difficult task.

In order to gain broad acceptance by the users (application programmers) a parallel I/O system must not only to offer a simple API but also deliver a high I/O throughput with little or no user intervention. The I/O system should therefore automatically optimise I/O operations based on information about the underlying hardware and the client application. The need of automatic optimisations implies building an I/O model that sufficiently describes the interdependencies of all the I/O parameters in order to evaluate different characteristics exhibited by applications. Several such models exist that primarily focus on special aspects (e.g., checkpointing) and/or only use a subset of the parameters and consequently have limited use.

Due to the complex nature of I/O and even if a complete I/O model was produced that optimisation of the I/O parameters in itself would be a demanding task. The huge number of possible parameter settings will inhibits an exhaustive search. So, intelligent algorithms are needed that only generate parameter sets that seem promising in respect with I/O performance and select the best one based on the model evaluation. These algorithms should produce the optimal (or a near optimal) result and they should not take longer to run than the time that can be gained by the achieved.

Some potential candidates to help solve the I/O optimisation problem are artificial intelligence algorithms and neural networks. Artificial intelligence algorithms are capable of finding near-optimum configurations in large search spaces and have already been successfully used to find partial solutions of the problem (using simulated annealing and genetic algorithms). Neural networks have been used to spot regular patterns in I/O accesses to help understand the optimisation problem.

It should also be remembered that I/O requests are dynamic and consequently the optimum I/O configuration will be changing with time due to factors like contention or maybe failure of a network connection or a disk. I/O optimisation in this case must be done autonomously (i.e., the system detects the changes itself) and in transparent manner that does not impact the client applications.

6.4 Future Aspects for Parallel I/O

The availability of data storage systems capable of storing huge data sets is rapidly increasing. The doubling in the sales of storage system each year can be seen as evidence of this. The support of fast and efficient access of data on these systems is crucial. The demand of new applications, such as multimedia, knowledge engineering, and large-scale scientific computing, is increasing at an astonishing rate.

Cluster-based platforms can provide a practical solution for I/O intensive applications. Past experience of massively parallel processing systems, however, does not provide all the answers of how to configure and optimise clusters for the new generation of I/O intensive applications. It is evident that new approaches and I/O models will need to be studied to cope with the demands of these applications. Cluster I/O will be a stimulating research area research for some time.

7. High Availability

Highly Available (HA) systems for traditional mission critical application have been around for almost as long as modern computers have been around. More recently, the need for highly available systems has increased as electronic commerce and other internet-based applications have become widely used with the burgeoning Web usage. Traditionally, HA systems consist of proprietary hardware and software components. However, the price/performance advantages of commodity-off-the-shelf (COTS) based clusters have had a compelling affect on HA vendors and their marketplace. The move to COTS-based HA components has driven down the entry price to buy a HA system, and as such the HA market is expanding and more users, both commercial (and many academic) demand minimal downtime for their systems.

In general, systems can provide different types of availability, including:

- *High availability* [90] – the provision of a cheaper alternative to fault tolerance by providing software solutions using COTS hardware components;
- *Continuous availability* – the provision of a non-stop service, representing an ideal state;
- *Fault tolerance* – the provision of high levels of availability by employing redundant, hardware for various system components.

Today, a typical HA solution consists of software modules that run on a system consisting of two or nodes comprising off-the-shelf computers. The failure of a node or its components results in all applications that were running on that node to be migrated over to another node in the system.

The current generation of HA solutions range from two to eight nodes-clusters, although products for larger clusters are available. A typical HA solution consists of:

1. A software infrastructure that enables the cluster to appear as a single system;
2. Cluster monitoring software;
3. Applications that use the services provided by the infrastructure that provide HA.

HA solutions are generally implemented at one of two levels, above or integrated within the operating system. HA solutions that are implemented above the operating system (middleware) often provide SSI and are critical to a system ease-of-use. The important elements of SSI include global file systems, devices and networking. SSI also impacts the functionality and usage of cluster management tools.

Integrating HA functionality into the underlying operating system has a number of advantages, such as better performance and ease of use. The main advantage the middleware HA solution includes independence with respect to the underlying operating

system. A middleware solution leads to better portability across heterogeneous platforms, and often a smaller application development cycle for the HA functionality in the system.

There are an increasing number of commonly used and commercial highly available applications. The most common ones are the various databases (Oracle, Informix, Sybase), file systems, web servers, and mail servers. Many of these come pre-packaged with HA solutions and others can be developed with the HA infrastructure in mind. In addition, a custom application may use an HA-API to convert an application into an HA version. While each particular HA-API has its own specific functionality, most have methods to start, monitor, or stop an application. Highly available cluster systems are an active area of research [91].

8. Numerical Libraries and Tools for Scalable Parallel Cluster Computing

An essential element for a successful cluster is an effective and efficient suite of numerical libraries and programming tools that is available to application developers. Clusters, especially heterogeneous ones, present tool and library developers a range of problems to solve in order to achieve adequate levels of performance. Tools and libraries that have been developed for existing parallel supercomputers, such as MPI [19] or PVM, will, in most cases, operate adequately on clusters. However, acceptable levels of efficiency or effectiveness may not be achieved. This problem will typically be exasperated if the cluster consists of SMP-nodes. Currently there is little software exists that offers the mixed-mode parallelism. Nevertheless, there is a huge interested in this form of mixed-mode programming.

Currently, message passing, using MPI, is the most popular paradigm for developing parallel and distributed application. MPI provides the communication layer of the library or package, which may or may not be revealed to the user. Both free and vendor supported versions of MPI ensures its widespread availability. More importantly, MPI ensures portability of application codes across all distributed and parallel platforms.

OpenMP [94] is an emerging standard that provides a portable API for the development of applications that used a shared memory-programming paradigm. This standard has obvious important implication for Cluster SMP nodes. It is clear that a mixed programming paradigm based on MPI and OpenMP will have an important impact on SMP-based clusters in the near future.

Clusters that consist of small and large SMPs, sometimes called constellations, represent a popular and fast growing area of the computing market. The depth of the memory hierarchy with its different access primitives and costs at each level makes clusters of SMPs more challenging to design and use effectively than their single SMP and cluster predecessors. It is envisaged that to take advantage of clusters of SMPs some abstract programming model that is not dependent on implementation details of the underlying machine would be needed. It is probable that some layered programming model that allows the underlying machine to be exposed at varying levels of detail will be used. Such a model would consist of a communication library, various data structure libraries, and numerical algorithms. An important aspect is to build libraries that are parameterised and can take advantage of the architectural features that most influence performance. Such features may include cache sizes, network latency, and bandwidth, for example. Identifying the parallelization strategies for combining distributed and shared-memory programming paradigms is an active area of research [108].

8.2 Numerical Libraries

Numerical libraries are an essential tool for scientific application programmers. The categories of libraries that are generally available include linear algebra, iterative solvers, optimization and partial differential equation solvers. The majority of the available numerical packages in this area are for linear algebra. Both direct solvers and iterative are well represented. Direct solver packages include ScaLAPACK [96] and PLAPACK [97] both of which are based on the parallelised version of LAPACK [97], but using different approaches.

The majority of iterative solvers packages use MPI as the underlying communications layer and include Aztec [99], Blocksolve [100] and PSPARSLIB [101]. Packages for eigenvalue problems include PARPACK [102] and PeIGS [103].

Optimatization and Partial Differential Equation (PDE) Solvers include freely available packages from Netlib [45]. Several machine vendors produce commercial packages and NAG [113] provides a commercial, supported, a general Parallel Library [104] based on MPI and also an SMP library [105] based on OpenMP.

PETSc [105] provides a set of tools for solving PDE problems including iterative methods. PETSc is an object-based interface to suite of C algorithms. The underlying implementation of algorithms or communications does not need to be known by the application developer. PETSc works well with C++-based packages such as ISIS++ and ELLPACK [107].

In general, cluster applications can take advantage of a legacy of numerical libraries of tools that were originally developed for parallel supercomputer, including both publicly available and commercially supported packages. These legacy packages may not in general provide the most effective software for clusters, but provide a good starting point. Abstract frameworks for developing cluster applications that can exploit complex memory hierarchies as well as heterogeneous clusters of SMPs are still some way in the future. Certainly the impetuous behind all types of scientific and engineering applications will drive the development of fast and efficient numerical libraries and tools for clusters.

9. Scientific and Commercial Applications of Cluster Computing

9.1 Introduction

The availability of commodity high-performance microprocessors and high-speed networks has made the use of clusters an appealing vehicle for low-cost commodity supercomputing [45]. Clusters provide a computational platform for all types of resource hungry parallel and distributed applications whether they are scientific or commercial.

One of the classes of applications that a cluster can typically cope with would be considered grand challenge or supercomputing applications. GCAs (Grand Challenge Applications) [178] are fundamental problems in science and engineering with broad economic and scientific impact. They are generally considered intractable without the use of state-of-the-art supercomputers. The scale of their resource requirements, such as processing time, memory, and communication needs distinguishes GCAs from other applications. For example, the execution of scientific applications used in predicting life-threatening situations such as earthquakes or hurricanes requires enormous computational power and storage resources. If we try to forecast an earthquake using a single PC, we would probably end up predicting it only after it had occurred, with all the inevitable consequences. In the past, these applications would be run on vector or parallel computing supercomputers costing millions of

dollars in order to perform predictions well in advance of actual events. Such applications can be migrated to run on commodity off-the-shelf-based clusters of computers and deliver comparable performance at low cost. In fact, in a few occasions expensive supercomputers have been replaced by low-cost commodity clusters such as Linux clusters in order to reduce maintenance costs and increase overall computing resources. [148].

A typical example of a grand challenge problem is the simulation of some phenomena that cannot be measured through physical experiments. GCAs [139] include complex crystallographic and microtomographic structural problems, protein dynamics and biocatalysis [185], relativistic quantum chemistry of actinides, virtual materials design and processing including crash simulations [186], and global climate modeling. This section describes commercial and Internet applications that demand both high availability and performance. This section also describes the use of clusters in image rendering applications, from television and movie animations, to architectural and web page design.

Clusters are increasingly being used for running commercial applications. In a business environment, for example in a bank, many of its activities are automated. However, a problem arises if the server that is handling customer transactions fails. The bank's activities could come to halt and customers would not be able to deposit or withdraw money from their account. Such situations can cause a great deal of inconvenience and result in loss of business and confidence in a bank. This is where clusters can be useful. A bank could continue to operate even after the failure of a server by automatically isolating failed components and migrating activities to alternative resources as a means of offering an uninterrupted service.

With the escalating popularity of the Web, computer system availability is becoming critical, especially in e-commerce application. Popular and free email service providers such as Hotmail [146] and search engines such as Hotbot [148] – that use Inktomi solutions [146] are hosted by clusters. Cluster-based systems can be used to execute many Internet applications: Web servers, search engines, email, security, proxy, and database servers. In the commercial environment these servers can be consolidated (see Figure 2) to create what is known as an enterprise server. They can be optimized, tuned, and managed for increased efficiency and responsiveness depending on the workload through load balancing techniques. A large number of low-end machines (PCs) can be clustered along with storage and applications for scalability, high availability, and performance. The leading companies building these types of systems are Compaq (ProLiant, AlphaServer, and NonStop Himalaya systems) [140], Hewlett-Packard (HyperFabric technology) [141], IBM (IBM S/390 server) [143], Microsoft (Application Center) [145] and Sun (Data Centre) [144].

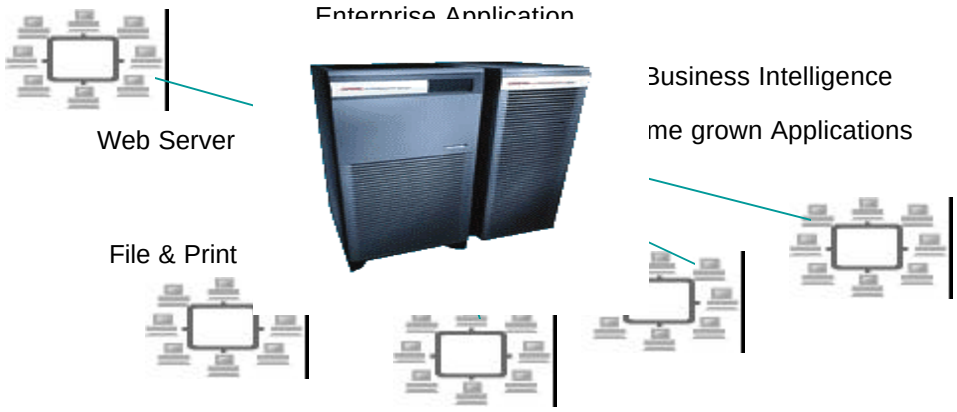

Figure 2. Enterprise Cluster (server consolidation). (Source [140]).

In the following sub-sections we discuss the use of cluster technologies in the following application areas:

- Web serving;
- Audio processing (voice based email);
- Data mining;
- Network simulation;
- Image processing.

9.2 Linux Virtual Server: A cluster-based Web Server

A cluster of servers, connected by a fast network, provides a viable platform for building highly scalable and available Internet services. This type of loosely coupled architecture is highly scalable, cost-effective and more reliable than a tightly coupled multiprocessor system [45] since failed components can be easily isolated and the system can continue to operate without any disruption.

The Linux Virtual Server [150] directs clients' network connection requests to multiple servers that share their workload, which can be used to build scalable and highly available Internet services. Prototypes of the Linux Virtual Server have already been used to build many sites that cope with heavy Internet loads, such as Linux portal [151] and UK National JANET Web Cache Service [150].

The Linux Virtual Server directs clients' network connection requests to the different servers according to scheduling algorithms and makes the parallel services of the cluster to appear as a single virtual service with a a single IP address. The Linux Virtual Server extends the TCP/IP stack of Linux kernel to support three IP load-balancing techniques:

1) NAT (Network Address Translation): Maps IP addresses from one group to another. NAT is used when hosts in internal networks want to access the Internet and be accessed in the Internet.
2) IP tunneling: Encapsulates IP datagram within IP datagrams. This allows datagrams destined for one IP address to be wrapped and redirected to another IP address.
3) Direct routing: Allows route response to the actual user machine instead of the load balancer.

The Linux Virtual Server also provides four scheduling algorithms for selecting servers from cluster for new connections:

1) Round robin: directs the network connections to the different server in a round-robin manner;
2) Weighted round robin: treats the real servers of different processing capacities. A scheduling sequence will be generated according to the server weights. Clients' requests are directed to the different real servers based on the scheduling sequence in a round-robin manner;
3) Least-connection: directs clients' network connection requests to the server with the least number of established connections;
4) Weighted least-connection: A performance weight can be assigned to each real server. The servers with a higher weight value will receive a larger percentage of live connections at any time.

Client applications interact with the cluster as if it were a single server. The clients are not affected by the interaction with the cluster and do not need modification. The application performance scalability is achieved by adding one or more nodes to the cluster. Automatically detecting node or daemon failures and reconfiguring the system appropriately achieve high availability.

The Linux Virtual Server that follows a three-tier architecture is shown in Figure 3. The functionality of each tier is:

- *Load Balancer*: the front end to the service as viewed by connecting clients. The load balancer directs network connections from clients who access a single IP address for a particular service, to a set of servers that actually provide the service.
- *Server Pool*: consists of a cluster of servers that implement the actual services, such as Web, Ftp, mail, DNS, and so on.
- *Backend Storage*: provides the shared storage for the servers, so that it is easy for servers to keep the same content and provide the same services.

The load balancer handles incoming connections using IP load balancing techniques. The Load balancer selects servers from the server pool, maintains the state of concurrent connections and forwards packets, and all the work is performed inside the kernel, so that the handling overhead of the load balancer is low. The load balancer can handle much larger numbers of connections than a general server, therefore the load balancer can schedule a large number of servers and it will not be a potential bottleneck in the system.

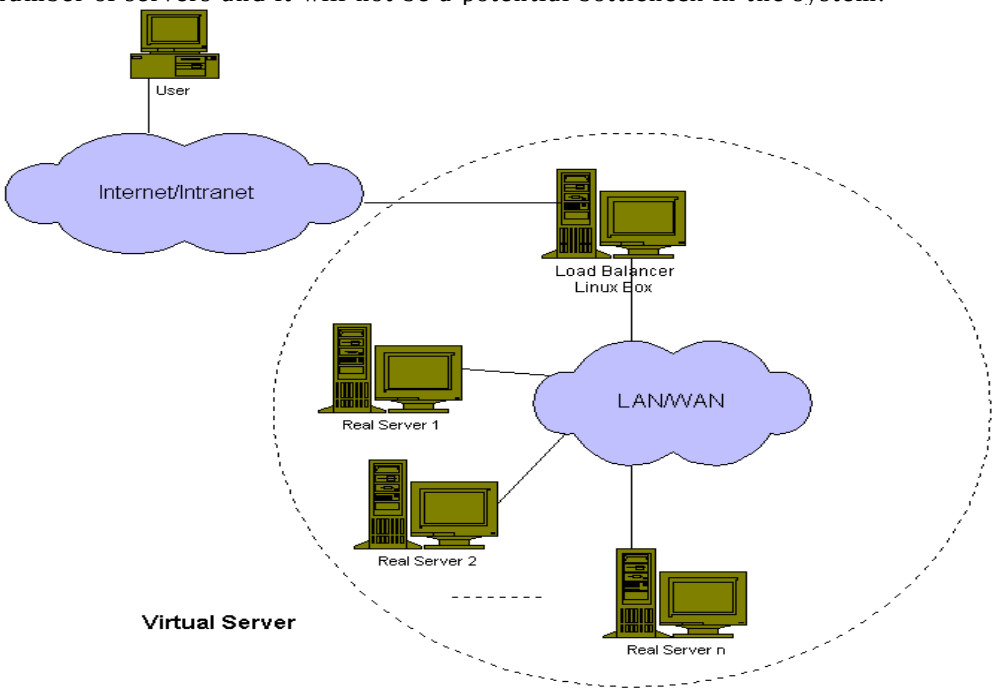

Figure 3. Linux Virtual Server (Source: W. Zhang [150]).

The server nodes may be replicated for either scalability or high availability. When the load on the system saturates the capacity of the current server nodes, more server nodes can be added to handle the increasing workload. One of the advantages of a clustered system is that it can be built with hardware and software redundancy. Detecting a node or daemon failure and then reconfiguring the system appropriately so that its functionality can be taken over by the remaining nodes in the cluster is a means of providing high system availability. A cluster monitor daemon can run on the load balancer and monitor the health of server nodes. If a server node cannot be reached by ICMP (Internet Control Message Protocol) ping or there is no response of the service in the specified period, the monitor will remove or disable the server in the scheduling table, so that the load balancer will not schedule new connections to the failed one and the failure of a server node can be masked.

The backend storage for this system is usually provided by distributed and fault tolerant file system. Such a system also takes care of the availability and scalability issue of file system accesses. The server nodes access the distributed file system in a similar fashion to that of accessing a local file system. However, multiple identical applications running on different server nodes may access a shared data concurrently. Any conflicts between applications must be reconciled so that the data remains in a consistent state.

9.3 Talking Email: Evoke's Reliable and Scalable Speech-to-email Service

Evoke Communications [153] is one of the service providers for Internet communication. They have developed a broad range of services to meet the diverse communication needs of businesses and consumers. One of Evoke's most popular applications is a speech-to-email system, called *talking email*. Evoke Talking Email [156] allows users to record a free voice message of up to 30 seconds in duration and send it as an email message. The service has the capability to deliver the message to thousands of users simultaneously. Additionally, users can post their voice message as a link embedded on a Web page. As the popularity of this service grew, meeting the demand require a scalable computing platform architecture. The resulting architecture utilizes both pipelined and multiple levels of parallel processing to meet the requirements of processing speed, reliability and scalability. At the core of the system is a very large (200 CPUs) cluster of Linux systems, integrated with a very large disk storage array.

Figure 4 shows the simplified message flow pipeline in speech-to-email service. The basic steps to record a speech message from a user and deliver it to the recipient are as follows.

- A user's call is answered;
- The user keys a series of pre-assigned DTMF (Dual-Tone Multi-Frequency)digits to match the message to appropriate database records;
- Then the message is digitally recorded, compressed and stored;
- An Email is sent to the recipient, along with a URL to retrieve the stored voice message.

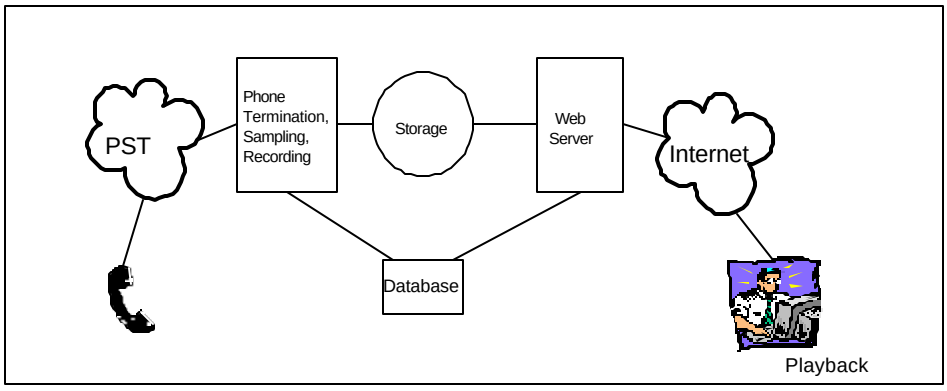

Figure 4. Simplified message flow pipeline in speech-to-email service
(Source: R. Brumbaugh, and T. Vernon [156]).

Evoke's speech-to-email system uses two separate servers with an intermediate short-term storage in-between. In the initial step, the email message is sampled and stored digitally by a dedicated telephony processor. The selection of number of processors in this stage largely determines the number of simultaneous calls to handle. This stage of the process does not require a great deal of computational power; however, it does require specialized functions and a continuous connection to the user and real-time capture of the voice message. To accomplish this, a large number of small proprietary telephony processors are assigned the

task of answering and recording the incoming calls. The recorded messages are stored as uncompressed digital audio files on a shared disk array.

The heart of the system is the transcoding server. This takes the digital audio files and performs processing and compression to create a format suited for long-term storage and streaming playback over an Internet server. This step uses the cluster of Linux systems to perform transcoding to meet the real time requirement. The number of parallel processors needed for this stage is primarily determined by the workload generation capability of the telephony processors, and overall throughput and insuring excess capacity. The current transcoding application is able to convert a message in approximately one-tenth of the time taken to record it, but other transcoding and processing may be added in the future without requiring additional transcoding processors. This asynchronous and faster processing allows the number of Linux nodes to be significantly less than the number of incoming voice channels.

A key element of the system is the storage of the compressed audio files for streaming playback on demand, when a recipient selects the URL sent to them in email. The storage system must be capable of holding a large amount of data, be reliable and upgradeable. The system chosen by Evoke was a commercial network file server that appears as a shared disk to each of the transcoding nodes.

The speech-to-email service is designed as an engine that may be integrated and used to enhance other services. An example of this is Blue Mountain Arts [154] who use Evoke's talking email engine to allow senders to add speech messages to their electronic greeting cards.

9.4 Parallel & Distributed Data Mining: Terabyte Challenge

Clusters have proved themselves to be very effective for a variety of data mining applications. The data mining process involves both compute and data intensive operations. Clusters provide two fundamental roles:

- Data-clusters provide the storage and data management services for the data sets being mined.
- Compute-clusters provide the computational services required by the data filtering, preparation and mining tasks [167].

The Terabyte Challenge [161] is an open, distributed testbed for experimental work related to managing, mining and modelling large, massive and distributed data sets. The Terabyte Challenge is sponsored by the National Scalable Cluster Project (NSCP) [160], the National Center for Data Mining (NCDM) [159], and the Data Mining Group (DMG) [158]. Figure 5 shows the NSCP architecture that connects clusters at multiple sites in different organizations using ATM networks to create a cluster of clusters (also know as a Hypercluster). The Terabyte Challenge's testbed is organized into workgroup clusters connected with a mixture of traditional and high performance, broadband networks. A meta-cluster is a cluster of work-group clusters connected over the Internet. A super-cluster is a cluster of workstations connected with a high performance network such as Myrinet.

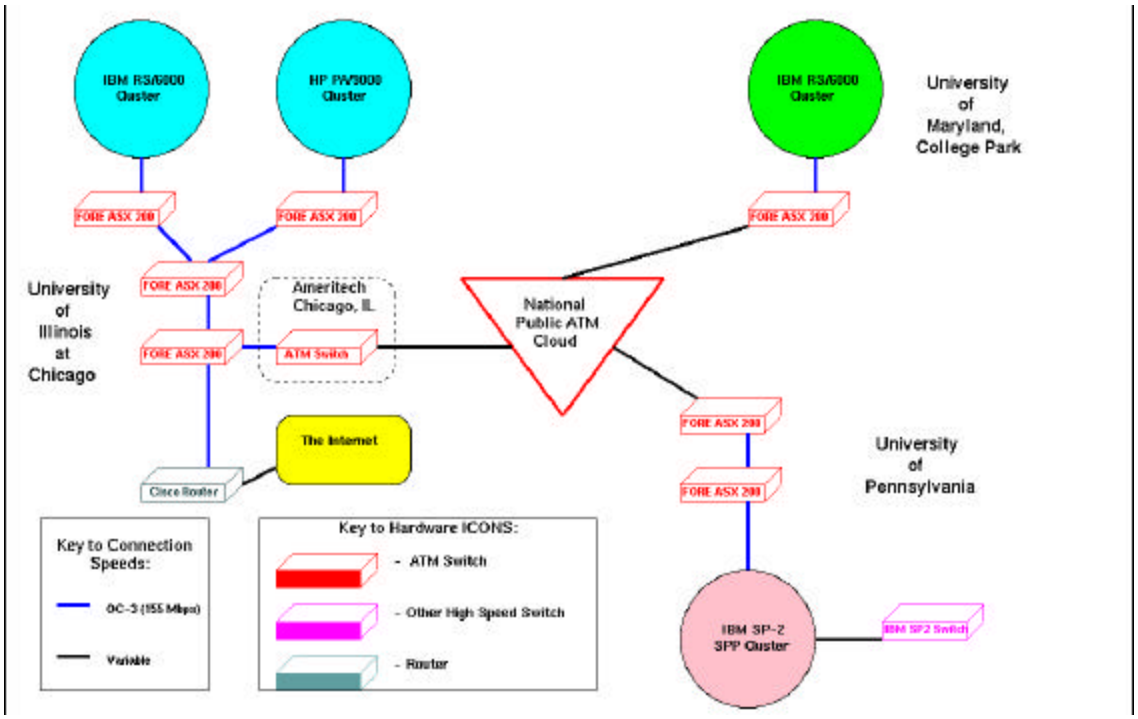

Figure 5. National Scalable Cluster Project – ATM connectivity (Source [162]).

The NSCP philosophy is to use commodity components with high performance networking to build a virtual platform with supercomputing potential. Currently, the NSCP meta-cluster consists of approximately 100 nodes and two TBytes of disk, with two robotic tape libraries. The clusters at University of Illinois at Chicago (UIC), University of Pennsylvania (Upenn), and Univeristy of Maryland (UMD) are linked by an ATM OC-3 (155 Mbps) network to form a single Super-Cluster. The NSCP has demonstrated the effectiveness of local and campus clusters of workstations linked using ATM and Ethernet connections. The NSCP has used the vBNS (very-high-performance Backbone Network Service) [162] to create wide area clusters of workstations for a variety of applications. The main applications of Terabyte Challenge include:

- EventStore – Data mining for high energy physics data with 30 nodes distributed at 6 sites. The data size is 480 GBytes and 79 million events. The underlying architectures are Papyrus distributed data clusters [157], distributed compute clusters [157], and Cluster Resource Markup Language [159] and aglet agents [164] to manage data and compute clusters. The performance of EventStore can achieve 30 MBytes/s in moving data between clusters.
- MedStore – Data mining for distributed health care data with 4 US sites and 2 international sites. The data size is less than 1 GByte and 600,000 records. This is the first distributed data mining application over international network. It uses Predictive Model Markup Language [165] built independently and combined with Papyrus, Cluster Resource Markup Language and aglets used to manage distributed computation.
- Alexa Crawler – Data mining for web documents with 2.1MBytes of reachable Web sites [166]. Figure 6 shows the system architecture of Alexa It composed of four layers The first layer gathers more than 80 GBytes of information per day, from more than 2000 different Web sites at a time. Alexa uses many commodity off-the-shelf disks that work together in terabyte clusters in second layer. To May 1999, Alexa had collected more than 13 TBytes of information from the Web [166]. Alexa use a number of data mning

techniques in third layer to find information about Web sites. Alexa displays the metadata and clusters Web sites in Related Links through the last layer of Alexa service.

- TrajectoryStore – data mining for simulation data with 4 nodes, 3 GBytes data size and 800,000 trajectories. Trajectories can be returned in constant time, independent of store size. It uses Papyrus to store complex data, and dyadic decomposition of phase space for scalability.
- Other applications include distributed BLAST search, textural data mining, economic data mining, etc.

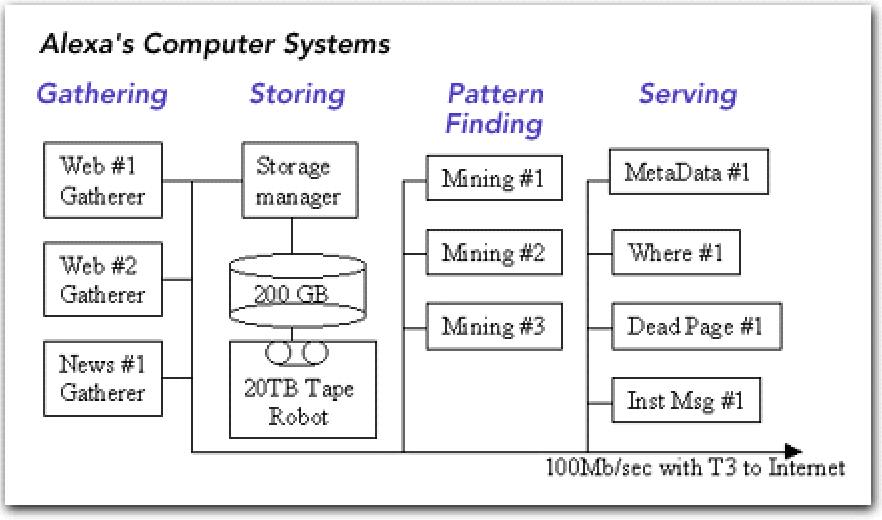

Figure 6. Alexa System Architecture (Source [166]).

An underlying technology of NSCP is Papyrus [157]. Papyrus is a layered infrastructure for high performance, wide area data mining and predictive modelling. Papyrus is specifically designed to support various clusters configurations. Papyrus is designed to support different data, task, and model strategies. These data strategies include local learning, centralized learning and hybrid learning. Task strategies refer to independent learning or coordinated learning. Model strategies can be meta-learning or knowledge probing. Papyrus is built over a data-warehousing layer, which can move data over both commodity and proprietary networks.

Papyrus applications can be designed to access any of the following four layers:

a. Data management layer called Osiris;
b. Data mining layer;
c. Predictive modelling layer;
d. Agent layer called Bast.

Currently, Papyrus has two modes:

- It can use Osiris to move segments of data from node to node;
- It can also use Bast to move predictive models from node to node.

Papyrus is the first distributed data mining system to be designed with the flexibility of moving data, moving predictive, or moving the results of local computations.

9.5 Ad Hoc Network Simulation: Parametric Computing

Parametric computational experiments are becoming increasingly important in science and engineering as a means of exploring the behaviour of complex systems. For example, an engineer may explore the behaviour of an aircraft wing by running a computational model of the airfoil multiple times while varying key parameters such as angle of attack or air speed. The results of these multiple experiments yield a picture of how the wing behaves in different parts of parametric space. Parametric computing is suitable for problems where a single program must be executed repeatedly with different initial conditions [168]. Such applications can be easily modelled and executed on clusters using software tools such as Nimrod [169]. Nimrod (or its commercial version, Clustor [172]) allows specifying an experiment using a simple GUI, and then performing the work on the nodes of the cluster. It provides automatic generation of input parameters and distribution of jobs, as well as the collection of results. The next generation of Nimrod, called Nimrod/G, supports parametric processing over computational Grids [174] with the addition of factors such as computational economy and deadline based scheduling [170].

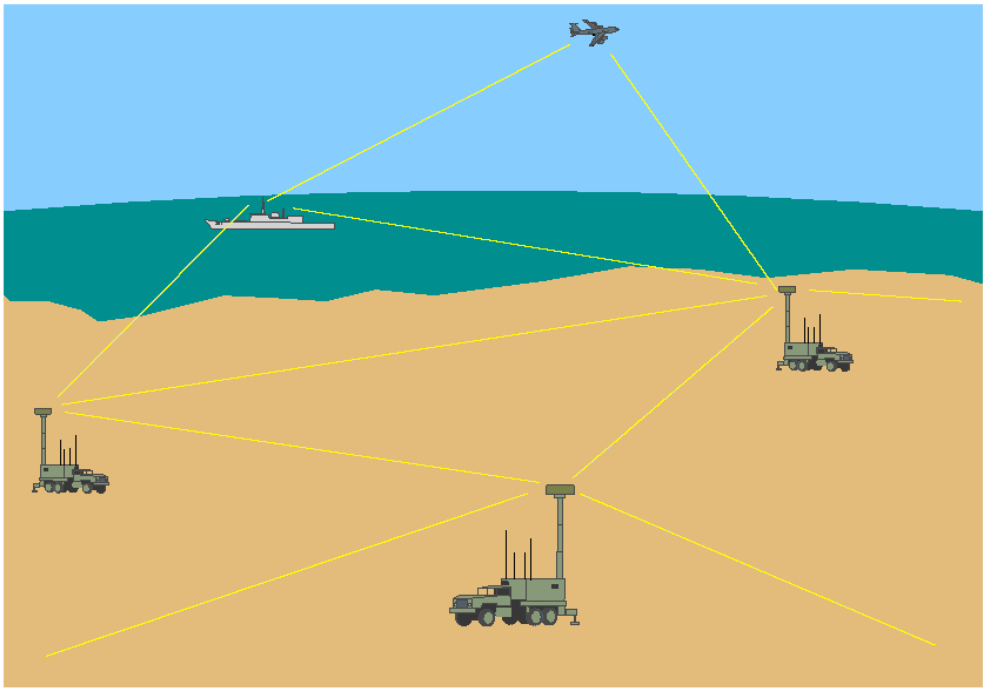

Figure 7. An example of the Ad Hoc Network (Courtesy of C. Kopp [172]).

One case for parametric computing on clusters is ad hoc network simulations. An ad hoc network can be considered an autonomous system of routers (nodes), interconnected via wireless links, which are free to roam in an arbitrary fashion. Figure 7 illustrates an example of an ad hoc network. The simulation models a multicast protocol known as the Ad Hoc Multicast Protocol (AMP) [173]. The purposes of the simulation are to explore achievable network performance at different microwave frequencies under different weather conditions, consider local tropospheric refraction and propagation impairments, and analyze global network performance. For example, network simulation under 100 different microwave frequencies and four types of weather conditions produce 400 different scenarios. Nimrod allows parameterisation of these two parameters (in input file) and automatically creates 400 jobs and manages their processing cluster nodes. If each of these jobs takes 2 hours of CPU time, then completing all of them on a single node takes 800 hours. Instead, if we run these jobs on clusters of 25 nodes, all jobs can be completed by 16 hours with some overhead time.

That means, clusters offer (almost) linear performance scalability when used to run such embarrassingly parallel applications.

9.6 Persistence of Vision: Parallel Image Rendering

Rendering is a technique for generating a graphical image from a mathematical model of a two or three-dimensional object or scene. A common method of rendering is ray tracing [175]. Ray tracing is a technique used in computer graphics to create realistic images by calculating the paths taken by rays of light entering the observer's eye at different angles. The paths are traced backwards from the viewpoint, through a point (a pixel) in the image plane until they hit some object in the scene or go off to infinity. Objects are modelled as collections of abutting surfaces, which may be rectangles, triangles or more complicated shapes. The optical properties of different surfaces (colour, reflectance, transmittance, refraction, and texture) also affect how it will contribute to the colour and brightness of the ray. The position, colour and brightness of light sources, including ambient lighting, is also taken into account.

Ray tracing is an ideal application for parallel processing since there are many pixels, each of whose values are independent and can be calculated in parallel. The Persistence of Vision Ray Tracer (POV-Ray) is an all-round 3-dimensional ray tracing software package [177]. It takes input information and simulates the way light interacts with the objects defined to create 3D pictures and animations. In addition to the ray tracing process, newer versions of POV can also use a variant of the process known as radiosity (sophisticated lighting) to add greater realism to scenes, particularly those that use diffuse lightPOVRay can simulate many atmospheric and volumetric effects (such as smoke and haze).

Given a number of computers and a demanding POVRay scene to render, there are a number of techniques to distribute the rendering among the available resources [180]. If one is rendering an animation then obviously each computer can render a subset of the total number of frames. The frames can be sent to each computer in contiguous chunks or in an interleaved order, in either case a preview (every N^{th} frame) of the animation can generally be viewed as the frames are being computed.

In many cases even single frames can take a significant time to render. This can occur for all sorts of reasons such as a complicated geometry, a sophisticated lighting (e.g., radiosity), high anti-aliasing rates, or simply a large image size. The usual way to render such scenes on a collection of computers is to split the final image up into chunks, rendering each chunk on a different computer and reconstructing the chunks together at the end. POVRAY supports this rendering by the initial file directives.

There are several ways an image may be split up into chunks, by row, column, or in a checker pattern. The basic procedure to render a scene on an N-node cluster is to create N initial files, each one rendering the appropriate row chunk. Processing of each initial file (image chunk) creates one output image file. When all the row chunks are rendered the image files are reconstructed together.

With regard to performance and efficiency, if each row chunk takes about the same time to render then one gets a linear improvement with the number of processors available. (This assumes that the rendering time is much longer compared to the scene loading time). Unfortunately this is not always the case, the rows across the sky might take a trivial length of time to render while the rows that intersect the interesting part of the scene might take a lot longer. In this case the machines that rendered the fast portions will stand idle for most of the time. This can be overcome by load balancing techniques.

One way around this is to split the scene into many more row chunks than there are computers and write a utility that submits jobs to machines as they become free. Another and perhaps neater way is to create row chunks of different heights according to the estimated rendering time. The script used to render the final image can be modified to render a much smaller image with the same number of row chunks. An estimate of the time for the different rows can be used to create narrow row chunks in the complex regions and wide row chunks in the fast rendering regions. Figure 8 gives an example of composing ray-tracing image in movie production by using software MegaPov [176] and MPI [179] by executing them on clusters.

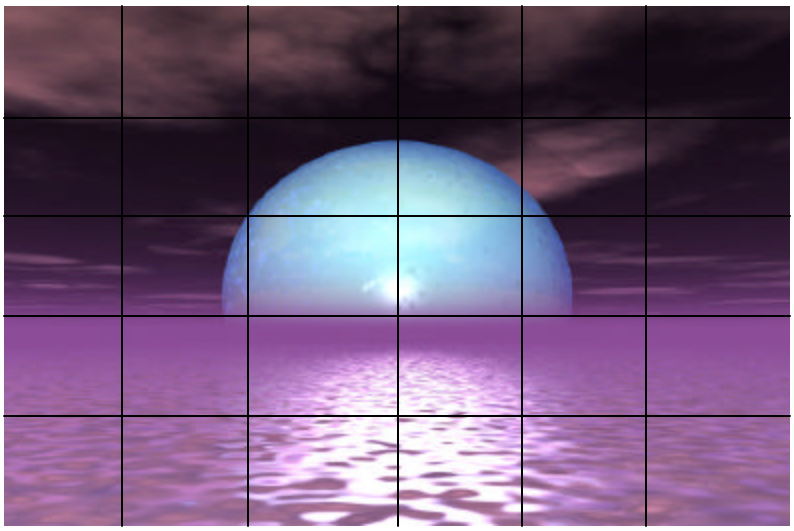

Figure 8. An image that is divided into multiple parts for quick rendering by processing them concurrently on a cluster. (Courtesy of D. Miller [181]: Water Sun was used as part of a multimedia concert event in which animations were projected onto dancers during the performance.)

10. Summary and Conclusions

In this chapter we have discussed the motivation for using clusters as well as the technologies available for building cluster-based systems. There is much emphasis placed on using the commodity-based hardware and software components to achieve high performance and scalability and at the same time keep the ratio of price versus performance low. A number of emerging basic building blocks, such as network technologies, operating systems, and middleware have been discussed. A number of software solutions for providing a single system image for clusters are emerging, yet there are still many opportunities for integrating various cluster tools and techniques and making them work together and so help create more usable and better unified clusters.

Clusters are being used to solve many scientific, engineering, and commercial applications. We have discussed a sample of these application areas and how they benefit from the use of clusters. The applications studied include, a Web server, an audio processing system (voice-based email), data mining, network simulations, and image processing. Many large international Web portals and e-commerce sites use clusters to process customer requests quickly and maintain high availability for 24 hours a day and throughout the year. The capability of clusters to deliver high performance and availability on a single platform is

empowering many existing and emerging applications and making clusters the platform of choice!

11. References

[1] Beowulf – http://www.beowulf.org

[2] T. Anderson, D. Culler, and D. Patterson. A Case for Network of Workstations. IEEE Micro, 15(1) pp 54-64, Feb. 95. http://now.cs.berkeley.edu/

[3] High Performance Virtual Machine – http://www-csag.ucsd.edu/projects/hpvm.html

[4] G.F. Pfister, In Search of Clusters, 2nd Edition, Prentice Hall PTR, pp 29, 1998, ISBN 0-13-899709-8

[5] M.G. Jacunski et al. Low-Latency Message Passing for Reflective Memory Networks. In Workshop on Communication, Architecture and Applications for Network-Based Parallel Computing (CANPC), 1999.

[6] Infiniband Architecture, http://www.infinibandta.org

[7] D.B. Gustavson and Q. Li, The Scalable Coherent Interface (SCI), IEEE Communications Magazine, Vol. 34, No. 8, August 1996, p. 52-63.

[8] L. L. Peterson and B. S. Davie, Computer Networks, A Systems Approach, Second Edition, Morgan Kaufmann Publishers, 2000.

[9] A.M. Mainwaring annd D.E. Culler, Active Messages: Organization and Applications Programming Interface, Technical Report, UC Berkeley, 1995. http://now.cs.berkeley.edu/

[10] S. Pakin, M. Lauria, and A. Chien, High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet, Supercomputing '95, http://www.supercomp.org/sc95/proceedings/567_SPAK/SC95.HTM

[11] M.A. Blumrich, C. Dubnicki, E. W. Felten, K. Li, and M. R. Mesarine, Virtual Memory Mapped Network Interfaces, IEEE Micro, February 1995.

[12] L. Prylli and B. Tourancheau, BIP: a new protocol designed for high performance networking on myrinet, in Workshop PC-NOW, IPPS/SPDP98, Orlando, USA, 1998.

[13] A. Basu, M. Welsh, and T. von Eicken, Incorporating Memory Management into User-Level Network Interfaces, presented at Hot Interconnects V, August 1997, Stanford University. Also available as Department of Computer Science, Cornell University, Technical Report, TR97-1620. http://www2.cs.cornell.edu/U-Net/papers.html

[14] M.A. Blumrich, et al, Design Choices in the SHRIMP System: An Empirical Study, Proceedings of the 25th Annual ACM/IEEE International Symposium on Computer Architecture, June 1998.

[15] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W-K. Su, Myrinet - A Gigabit-per-Second Local-Area Network. IEEE Micro, Vol. 15.

[16] Virtual Interface Architecture, http://www.viarch.org/

[17] M-VIA: A High Performance Modular VIA for Linux, http://www.nersc.gov/research/FTG/via/

[18] P. Buonadonna, A. Geweke, and D. Culler, An Implementation and Analysis of the Virtual Interface Architecture, Proceedings of SC98, Orlando, Florida, November 1998

[19] Message Passing Interface, MPI Forum. http://www.mpi-forum.org/docs/docs.html

[20] The Gigabit Ethernet Alliance, http://www.gigabit-ethernet.org/

[21] ATM Forum. ATM User-Network Interface Specification. Prentice Hall, September 1993.

[22] MVICH – MPI for VIA, http://www.nersc.gov/research/FTG/mvich/

[23] E. Leonardi, F. Neri, M. Gerla, and P. Palnati, Congestion Control in Asynchronous, High-Speed Wormhole Routing Networks, IEEE Communications Magazine, Vol. 34, No. 11, November, 1996, p. 58-69.

[24] A. DeHon, Fat-Tree Routing for Transit, Technical Report, MIT, http://www.ai.mit.edu/people/andre/sb.html

[25] Giganet, Inc., http://www.giganet.com/index.asp
[26] Myricom, Inc., http://www.myri.com/
[27] Qsnet, http://www.quadrics.com/web/public/fliers/qsnet.html
[28] Server Net II, http://www.servernet.com/
[29] SCI Association, http://www.SCIzzL.com/
[30] Atoll: A network on a chip, http://www.atoll-net.de/
[31] Fibre Channel Industry Association, http://www.fibrechannel.com/
[32] VMIC Reflective Memory, http://reflectivememory.vmic.com/
[33] K. Scholtyssik and M. Dormanns, Simplifying the use of SCI shared memory by using software SVM techniques, *2nd Workshop Cluster-Computing*, Published in: W. Rehm, and T. Ungerer (Eds.), Cluster-Computing, Proc. 2nd Workshop, March 1999, Universität Karlsruhe (CSR-99-02) – http://www.tu-chemnitz.de/informatik/RA/CC99/
[34] C. Amza, A.L. Cox, S.D. Warkadas, P. Kelehr, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, TreadMarks: Shared Memory Computing on Networks of Workstations, *IEEE Computer*, Vol. 29, No. 2, 1996 – http://standards.ieee.org/regauth/oui/tutorials/sci.html
[35] G. Cabillic, T. Priol, I. Puaut, MYOAN: an implementation of the KOAN shared virtual memory on the Intel paragon, *Technical Report RR-2258*, Institute National de Recherche en Informatique et en Automatique, 1994
[36] E. Anderson, A. Mainwaring, J. Neefe, C. Yoshikawa, T. Anderson, D. Culler, D. Patterson, Experience with Two Implementations of Network RAM, *Internal Report, Computer Science Division*, University of California at Berkeley – http://http.cs.berkeley.edu/~eanders/projects/netram/
[37] *Linux – How Good Is It?* Executive Summary, D.H. Brown Associates Inc. - http://www.dhbrown.com, 1999
[38] Solaris MC – http://www.sunlabs.com/research/solaris-mc
[39] Puma – http://www.cs.sandia.gov/puma
[40] MOSIX – http://www.mosix.cs.huji.ac.il.
[41] DS-Online Grid Computing – http://computer.org/channels/ds/gc/
[42] Extreme Linux – http://www.extremelinux.org/
[43] RedHat – http://www.redhat.org/
[44] TOP500 – http://www.top500.org
[45] R. Buyya (editor), *High Performance Cluster Computing: Architectures and Systems*, Vol. 1, Prentice Hall PTR, NJ, USA, 1999.
[46] K. Hwang, H. Jin, E. Chow, C.-L. Wang, and Z. Xu, *Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space*, IEEE Concurrency, vol.7(1), pp.60-69, Jan – March, 1999
[47] B. Walker and D. Steel, *Implementing a Full Single System Image UnixWare Cluster: Middleware vs. Underware*, In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas, USA, 1999.
[48] G. Popek and B.J. Walker (Ed.), The Locus Distributed System Architecture, *MIT Press*, 1996
[49] Memory Channel, http://www.digital.com/info/hpc/systems/symc.html
[50] Distributed Shared Memory Systems, http://www.cs.umd.edu/~keleher/dsm.html
[51] GLUnix: A Global Layer Unix for a Network of Workstations, http://now.cs.berkeley.edu/Glunix/glunix.html
[52] Parallel Virtual Machine (PVM), http://www.epm.ornl.gov/pvm/
[53] CODINE Resource-Management System, http://www.genias.de/products/codine/
[54] Condor, http://www.cs.wisc.edu/condor/
[55] PARMON: A Portable and Scalable Monitoring System for Clusters, http://www.csse.monash.edu.au/~rajkumar/parmon/
[56] Linux Virtual Server Project, http://www.LinuxVirtualServer.org/

[57] International Organization for Standardization, http://www.iso.ch/
[58] DECmesssageQ Documentation – http://www.digital.com/info/decmessageg
[59] MQseries – http://www.software.ibm.com/ts/
[60] TopEnd – http://www.ncr.com/
[61] Mullendar. S. (Ed), Distributed Systems, 2nd Edition, *Adison-Wesley*, 1993, ISDN 0-201-62427-3
[62] The Open Group, http://www.opengroup.org/
[63] Using Name Services, *Sun Microsystems*, http://www.sun.com/smcc/solaris-migration/docs/transition-guide_2.5/nis.html
[64] The NFS Distributed File Service, *NFS White Paper, Sun Microsystems*, March, 1995, http://www.sun.com/software/white-papers/wp-nfs/
[65] Object Management Group, http://www.omg.org/
[66] Microsoft – http://www.microsoft.com/com/tech/com.asp
[67] Java RMI – http://www.javasoft.com/products/jdk/rmi/
[68] Jini – http://www.sun.com/jini/
[69] Javasoft – http://www.javasoft.com/products/
[70] CODINE/GRD – http://www.genias.de
[71] M.A. Baker, G.C. Fox and H.W. Yau, Review of Cluster Management Software, NHSE Review, Vol 1, No. 1, May 1996, http://www.nhse.org/NHSEreview/96-1.html
[72] D. Serain, Middleware, *Practitioner Series, Springer-Verlag*, 1999, ISBN 1-852330012
[73] R. Bennett, K. Bryant, A. Sussman, R. Das, and Joel Saltz. Jovian: A Framework for Optimizing Parallel I/O. In Scalable Parallel Libraries Conference, IEEE Computer Society Press, MS, October 1994.
[74] R. Bordawekar, J. del Rosario, and A. Choudhary. Design and Evaluation of Primitives for Parallel I/O. In Supercomputing '93, *IEEE Computer Society Press*, pages 452 – 461, Portland, OR, 1993.
[75] Y. Chen, et al, Performance Modeling for the Panda Array I/O Library. In Supercomputing '96, ACM Press and IEEE Computer Society Press, November 1996.
[76] A. Choudhary and D. Kotz. Large-Scale File Systems with the Flexibility of Databases. *ACM Computing Surveys*, vol. 28A(4), December 1996.
[77] D.G. Feitelson, P.F. Corbett, Y. Hsu, and J. Prost, Parallel I/O Systems and Interfaces for Parallel Computers. In Topics in Modern Operating Systems, IEEE Computer Society Press, 1997.
[78] N. Galbreath, W. Gropp, and D. Levine, Applications-Driven Parallel I/O. In Supercomputing '93, IEEE Computer Society Press, Portland, Oregon, 1993.
[79] G.A. Gibson, J.S. Vitter, and J. Wilkes, Strategic Directions in Storage I/O Issues in Large-Scale Computing. ACM Computing Surveys, vol. 28(4), December 1996.
[80] D. Kotz, Disk-Directed I/O for MIMD Multiprocessors. ACM Transactions on Computer Systems, vol. 15(1), February 1997.
[81] MPI-IO: A Parallel File I/O Interface for MPI. The MPI-IO Committee, April 1996.
[82] B. Nitzberg and Samuel A. Fineberg, Parallel I/O on Highly Parallel Systems, Supercomputing '95 Tutorial M6 Notes. Technical Report NAS-95-022, NASA Ames Research Center, December 1995.
[83] J.L. Hennessy, and D.A. Patterson, Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers Inc., 1995.
[84] R.H. Patterson, G.A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed Prefetching and Caching. In Fifteenth ACM Symposium on Operating Systems Principles, ACM Press, pp.79-95, CO, December 1995.
[85] E. Schikuta, T. Fuerle, and H. Wanek, ViPIOS: The Vienna Parallel I/O System. In Euro-Par'98, Southampton, England, Springer-Verlag, September 1998.
[86] T. Sterling, D.J. Becker, D. Savarese, M.R. Berry, and C. Res. Achieving a Balanced Low-cost Architecture for Mass Storage Management Through Multiple Fast Ethernet

Channels on the Beowulf Parallel Workstation. In International Parallel Processing Symposium, 1996.

[87] H. Stockinger, Classification of Parallel I/O Products. In Parallel And Distributed Processing Techniques and Applications Conference, July 1998.

[88] R. Thakur and A. Choudhary, An Extended Two-Phase Method for Accessing Sections of Out-of-Core Arrays. Scientific Programming, Vol. 5(4), 1996.

[89] E. Schikuta and H. Stockinger, Parallel I/O for Clusters: Methodologies and Systems In High Performance Cluster Computing: Architectures and Systems, Ed. R. Buyya, Prentice Hall, 1999.

[90] The D.H. Brown Reports, http://www.dhbrown.com

[91] I. Pramanick, High Availability, in A Whitepaper on Cluster Computing, submitted to the International Journal of High-Performance Applications and Supercomputing, April 2000.

[92] FirstWatch, http://www.veritas.com

[93] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, MPI: The Complete Reference, *MIT Press*, Boston 1996.

[94] L. Dagum, and R. Menon, OpenMP: An Industry-Standard API for Shared-Memory programming, in *IEEE Computational Science & Engineering*, Vol. 5, No. 1, January/March 1998.

[95] R.J. Allan Y.F. Hu and P. Lockey, Parallel Application Software on High Performance Computers. Survey of Parallel Numerical Analysis Software, http://www.cse.clrc.ac.uk/Activity/HPCI

[96] L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, D. Walker, and R.C. Whaley, ScaLAPACK Users' Guide, *SIAM*, Philadelphia, 1997 - http://www.netlib.org/scalapack

[97] P. Alpatov, et al, PLAPACK: Parallel Linear Algebra Libraries Design Overview, *SC97* - http://www.cs.utexas.edu/users/rvdg/plapack

[98] E. Anderson, et al, Lapack Users' Guide, Release 2.0. SIAM, Philadelphia, 1995.

[99] S.A. Hutchinson, L.V. Prevost, Tuminaro and J.N. Shadid, AZTEC Users' Guide: Version 2.0. Sandia National Labs, 1998, http://www.cs.sandia.gov/CRF/aztec1.html

[100] M.T. Jones and P.E. Plassman, Blocksolve V1.1: Scalable library software for parallel solution of sparse linear systems. ANL Report 92/46, Argonne National Laboratory, December 1992 – http://www-unix.mcs.anl.gov/sumaa3d/BlockSolve

[101] Y. Saad and M. Sosonkina, Solution of distributed sparse linear systems using psparslib. In Applied Parallel Computing, Proc. *PARA'98*, pages 501-9. Springer Verlag, Lecture Notes in Computer Science, Vol. 1541, 1998 – http://www.cs.umn.edu/Research/arpa/p_sparslib/psp-abs.html

[102] R.B. Lehoucq, D.C. Sorensen and C. Yang, ARPACK Users' Guide: Solution of Large-scale Eigenvalue Problems with implicitly-restarted Arnoldi Methods. *SIAM*, Philadelphia, 1998 – http://www.caam.rice.edu/software/ARPACK

[103] D. Elwood, G. Fann and R. Littlefield, Parallel Eigensolver System User Manual. *Batelle Pacific Northwest Laboratory*. (Available from anonymous@ftp://pnl.gov)

[104] The NAG Parallel Library Manual, Release 2, *Numerical Algorithms Group Ltd*, Oxford (1997).

[105] S. Salvini and J. Waniewski, Linear Algebra Subprograms on Shared Memory Computers: Beyond LAPACK. In J. Waniewski, J. Dongarra, K. Madsen, D. Olesen (eds.), Applied Parallel Computing Industrial Computation and Optimization, 3rd International Workshop, *PARA'96*, Lyngby, Denmark, August 18--21, 1996, Proceedings. *Springer-Verlag Lecture Notes in Computer Science*, Vol. 1184, 1996.

[106] W. Gropp and B. Smith, Scalable, extensible, and portable numerical libraries. *Technical report*, Argonne National Laboratory.

[107] S. Weerawarana, E.N. Houstis, R.J. Rice A.C. Catlin, C.L. Crabill, C.C. Chui, Pdelab: an object-oriented framework for building problem solving environments for PDE-

based applications. *Technical Report CSD-TR-94-021*, Purdue University, March 1994 – http://ziggurat.ca.sandia.gov/isis
[108] S. Salvini, B. Smith, and J. Greenfield, Towards Mixed Mode Parallelism on the New Model F50-based IBM SP System. *Albuquerque HPCC Report AHPCC98-003.*
[109] V. Eijkhout, A Survey of Iterative Linear System Solver Packages, http://www.netlib.org/utk/papers/iterative-survey/
[110] J.J. Dongarra, Freely Available Software For Linear Algebra On The Web, http://www.netlib.org/utk/people/JackDongarra/la-sw.html
[111] Netlib – http://www.netlib.org
[112] National High Performance Software Exchange (NHSE) – http://www.nhse.org
[113] Numerical Algorithms Group – http://www.nag.co.uk
[139] R. Buyya (editor), *High Performance Cluster Computing: Programming and Applications*, Vol. 2, ISBN 0-13-013785-5, Prentice Hall PTR, NJ, USA, 1999.
[140] COMPAQ Server Consolidation – http://www.compaq.com/solutions/serverconsolidation/
[141] Hewlett-Packard System Consolidation – http://www.unixsolutions.hp.com/solutions/sc/
[142] IBM RS/6000 Server Consolidation – http://www.rs6000.ibm.com/solutions/sc/
[143] IBM S390 Server Consolidation – http://www.s390.ibm.com/sc/
[144] Sun Data Center Consolidation – http://www.sun.com/datacenter/consolidation/
[145] Microsoft Application Center 2000 – http://www.microsoft.com/applicationcenter/
[146] Hotmail – http://www.hotmail.com
[147] Inktomi Corporation, *A Whitepaper on the INKTOMI Technology behind HOTBOT*, http://www.inktomi.com/products/network/traffic/tech/clustered.html, 2000.
[148] Hotbot – http://www.hotbot.com
[149] Computer World, Linux gushes savings for oil giant: Switch from IBM saves Hess nearly $2M, http://www.computerworld.com/cwi/story/0,1199,NAV47_STO35573,00.html
[150] Linux Virtual Server – http://www.LinuxVirtualServer.org/
[151] Linux Portal – http://www.linux.com
[152] W. Zhang, Linux Virtual Servers for Scalable Network Services, *Proceedings of 2000 Ottawa Linux Symposium*, July 19 – 22, 2000, Ottawa, Canada.
[153] S. Horman, Creating Redundant Linux Servers, *Proceedings of the 4th Annual LinuxExpo Conference*, May 1998.
[154] Blue Mountain – http://www.bluemountain.com
[155] Evoke Communication – http://www.evoke.com/
[156] R. Brumbaugh, and T. Vernon, Design of a Very-large Linux Cluster for Providing Reliable and Scalable Speech-to-email Service, 4th Annual Linux Showcase & Conference, Extreme Linux: Clusters and High Performance Computing Workshop, Oct. 10-14, 2000, Atlanta, Georgia, USA.
[157] S.M. Bailey, R.L. Grossman, H. Sivakumar, A.L. Turinsky, Papyrus: A System for Data Mining over Local and Wide Area Clusters and Super-Clusters, *Proceedings of Supercomputing '99 Conference*, Portland, Oregon, Nov. 1999.
[158] Data Mining Group – http://www.dmg.org
[159] National Center for Data Mining – http://www.ncdm.uic.edu
[160] National Scalable Cluster Project – http://www.ncsp.uic.edu/
[161] The Terabyte Challenge – http://www.ncdm.uic.edu/TC2000.htm
[162] NSCP – http://www.lac.uic.edu/nscp_atm.gif
[163] VBNS – http://www.vbns.net/
[164] D. Lange, and M. Oshima, Programming and Deploying Java Mobile Agents with Aglets, *Addison-Wesley*, USA, 1998.
[165] R.L. Grossman, et al, The Management and Mining of Multiple Predictive Models Using the Predictive Modelling Markup Language (PMML), *Information and Software Technology*, 1999.
[166] Alexa Technology – http://www.alexa.com/support/technology.html

[167] D.B. Skillicorn, Data Mining: Parallelism's Killer Application, Tutorial at the 7th Australian Conference on Parallel and Real-Time Systems (PART 2000), Sydney, Australia, November 2000
[168] Nimrod – http://www.csse.monash.edu.au/~davida/nimrod/
[169] D. Abramson, R. Sosic, J. Giddy and B. Hall, Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations, The 4h IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.
[170] R. Buyya, D. Abramson and J. Giddy, Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), Beijing, China. IEEE Computer Society Press, USA, 2000.
[171] Parallel Parametric Modelling Engine – http://hathor.csse.monash.edu.au/
[172] Active Tools Clustor – http://www.activetools.com
[173] C. Kopp, Supercomputing on a Shoestring – Practical Experience with the Monash PPME Linux Cluster, Proceedings of Open Source - AUUG'99, Melbourne, Australia, September 1999.
[174] I. Foster, and C. Kesselman (editors), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, USA, 1999.
[175] P. Shirley, Realistic Ray Tracing, A K Peters Ltd, 2000.
[176] MegaPov – http://nathan.kopp.com/patched.htm
[177] POVRay – http://www.povray.org/
[178] Argonne National Laboratory, Grand Challenge Applications, http://www-fp.mcs.anl.gov/grand-challenges/
[179] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, A High-Performance, Portable Implementation of the Message Passing Interface (MPI) Standard, Parallel Computing Journal, Volume 22, Number 6, September 1996.
[180] P. Bourke, Parallel Rendering: Using POVRAY on a Computer Farm, Swinburne Astrophysics and Supercomputing Centre, http://www.swin.edu.au/astronomy/pbourke/povray/parallel/, Australia, 2000.
[181] D.H Miller (http://www.casdn.neu.edu/~dmiller/), Image available at http://www.swin.edu.au/supercomputing/rendering/watersun/
[182] D.R. Engler, M.F. Kaashoek, and J. O'Toole, Jr., Exokernel: an operating system architecture for application-level resource management, In the Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP '95), Copper Mountain Resort, Colorado, December 1995, pages 251-266.
[183] J. Protic, M. Tomaevic, and V. Milutinovic (Eds.), *Distributed Shared Memory: Concepts and Systems*. ISBN 0-8186-7737-6. IEEE Computer Society Press, August 1997.
[184] D. Abramson, I. Foster, J. Giddy, A. Lewis, R. Sosic, R. Sutherst, and N. White, *The Nimrod Computational Workbench: A Case Study in Desktop Metacomputing*, Australian Computer Science Conference (ACSC 97), Macquarie University, Sydney, Feb 1997.
[185] S. Bhandarkar, S. Chirravuri, and S. Machaka, Biomedical Applications Modelling, *High Performance Cluster Computing: Programming and Applications*, Vol. 2, ISBN 0-13-013785-5, Prentice Hall PTR, NJ, USA, 1999.
[186] F. Ferstl, *Global Resource Director (GRD): Customer Scenarious for Large Multiprocessor Environments*, IEEE International Workshop on Cluster Computing, Melbourne, Australia, December 2-3, 1999.