Chapter 16

CLUSTER COMPUTING: HIGH-PERFORMANCE, HIGH-AVAILABILITY, AND HIGH-THROUGHPUT PROCESSING ON A NETWORK OF COMPUTERS

Chee Shin Yeo[1]*, Rajkumar Buyya*[1]*, Hossein Pourreza*[2]*,*
Rasit Eskicioglu[2]*, Peter Graham*[2]*, Frank Sommers*[3]
[1]The University of Melbourne, Australia
[2]The University of Manitoba, Canada
[3]Autospaces, LLC

1 INTRODUCTION

The first inspiration for cluster computing was developed in the 1960s by IBM as an alternative to linking large mainframes in order to provide a more cost-effective form of commercial parallelism [1]. At that time, IBM's Houston Automatic Spooling Priority (HASP) system and its successor, Job Entry System (JES), allowed the distribution of work to a user-constructed mainframe cluster. IBM still supports clustering of mainframes through its Parallel Sysplex system, which allows the hardware, operating system, middleware, and system management software to provide dramatic performance and cost improvements while permitting large mainframe users to continue to run their existing applications.

However, cluster computing did not gain momentum until the convergence of three important trends in the 1980s: high-performance microprocessors, high-speed networks, and standard tools for high-performance distributed computing. A possible fourth trend is the increasing need of computing power for computational science and commercial applications, coupled with the high cost and low accessibility of traditional supercomputers. These four building blocks are also known as *killer-microprocessors, killer-networks, killer-tools*, and *killer-applications*, respectively. The recent advances in these technologies and their availability as cheap and commodity components are making clusters or networks of computers such as Personal Computers (PCs), workstations, and Symmetric Multiple-Processors (SMPs) an appealing solution for cost-effective parallel computing. Clusters, built using commodity-off-the-shelf (COTS) hardware components as well as free, or commonly used, software, are playing a major role in redefining the concept of supercomputing. And consequently, they have emerged as mainstream parallel and distributed platforms for high-performance, high-throughput, and high-availability computing.

The trend in parallel computing is to move away from traditional specialized supercomputing platforms, such as the Cray/SGI T3E, to cheaper and general-purpose systems consisting of loosely coupled components built up from single or multiprocessor PCs or workstations. This approach has a number of advantages, including being able to build a platform for a given budget that is suitable for a large class of applications and workloads.

The emergence of cluster platforms was driven by a number of academic projects, such as Beowulf [2], Berkeley NOW [3], and HPVM [4], that prove the advantage of clusters over other traditional platforms. These advantages include low-entry costs to access supercomputing-level performance, the ability to track technologies, incrementally upgradeable system, open-source development platforms, and vendor independence. Today, clusters are widely used for research and development in science, engineering, commerce, and industry applications that demand high-performance computations. In addition, clusters encompass strengths such as high availability and scalability that motivate wide usage in non-supercomputing applications as well, such as clusters working as web and database servers.

A cluster is a type of parallel or distributed computer system that consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource [1, 5]. The typical architecture of a cluster is shown in Figure 16.1. The key components of a cluster include multiple standalone computers (PCs, workstations, or SMPs), operating systems, high-performance interconnects, middleware, parallel programming environments, and applications.

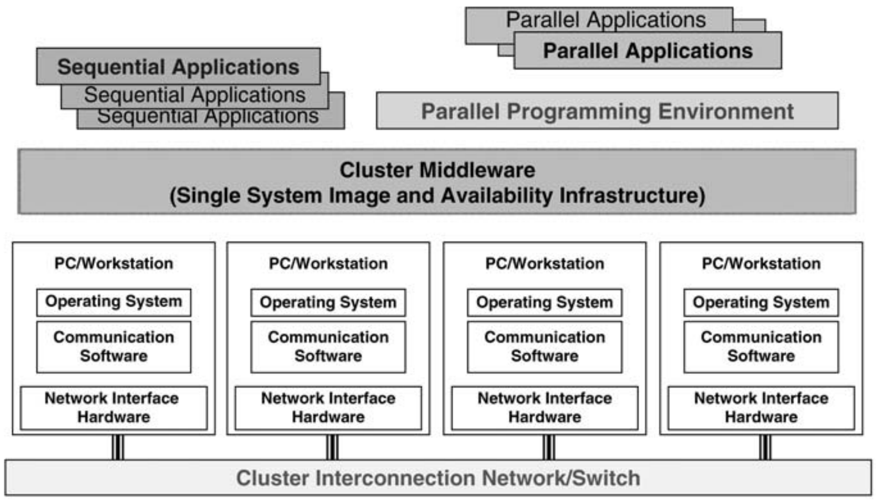

Figure 16.1. Cluster architecture (R. Buyya [1]).

The remaining part of this chapter focuses on cluster-specific components and their functionality, along with representative examples. It assumes that the reader is familiar with the standard commodity hardware and software components such as stand-alone computers, operating systems such as Linux and Windows, and standard communication software such as TCP/IP.

2 INTERCONNECTION TECHNOLOGIES AND COMMUNICATION SOFTWARE

Clusters need to incorporate fast interconnection technologies in order to support high-bandwidth and low-latency interprocessor communication between cluster nodes. Slow interconnection technologies had always been a critical performance bottleneck for cluster computing. Today, improved network technologies help realize the construction of more efficient clusters.

Selecting a cluster interconnection network technology depends on several factors, such as compatibility with the cluster hardware and operating system, price, and performance. There are two metrics to measure performance for interconnects: bandwidth and latency. Bandwidth is the amount of data that can be transmitted over the interconnect hardware in a fixed period of time, while latency is the time needed to prepare and transmit data from a source node to a destination node.

Table 16.1 gives a summary of some interconnection technologies, which are then compared as shown in Table 16.2. The comparisons examine factors that include bandwidth, latency, hardware availability, support for Linux, maximum number of cluster nodes supported, how the protocol is implemented, support for Virtual Interface Architecture (VIA), and support for Message Passing Interface (MPI). VIA [9] is a standard for the low-latency communication software interface that was developed by a consortium of hardware producers and academic institutions and that has been adopted for use by most cluster vendors. MPI [10] provides message passing through a set of libraries that users can use to develop parallel and distributed applications. This means that MPI provides the communication layer for user applications and thus ensures portability of application code across all distributed and parallel platforms.

With the current popularity of cluster computing, it is increasingly important to understand the capabilities and potential performance of various network interconnects for clusters. Furthermore, due to the low cost of clusters and their growing acceptance within the scientific community, many recent cluster builders are not computer scientists or engineers and thus have limited technical computing skills. This new group of cluster builders is less interested in features such as Network Interface Card (NIC) programmability and special messaging libraries. Instead, they are concerned with two primary factors: cost and performance. While cost is easily determined and compared, performance is more difficult to assess, particularly for users who may be new to cluster computing.

Several performance assessments of cluster systems, and of specific interconnects, have been performed [15–19]. Unfortunately, much of the work done has, in some sense, been "comparing apples to oranges." This is because most existing performance analyses have been forced to compare results for interconnects being

Table 16.1. Examples of some interconnection technologies.

Interconnection Technology	Description
Gigabit Ethernet	• Provides a reasonably high bandwidth given its low price, but suffers from relatively high latency, thus restricting Gigabit Ethernet as a good choice. However, the low price of Gigabit Ethernet is appealing to building clusters. • http://www.10gea.org
Giganet cLAN	• Giganet cLAN was developed with the goal of supporting VIA in hardware, and it supports a low latency. But it provides only a low bandwidth of less than 125 MBytes/s, thus making it not a viable choice for implementing fast cluster networks. • http://www.giganet.com
Infiniband [6]	• The latest industry standard based on VIA concepts and released in 2002, Infiniband supports connecting various system components within a system such as interprocessor networks, I/O subsystems, or multiprotocol network switches. This makes Infiniband independent of any particular technology. • http://www.infinibandta.org
Myrinet [7]	• The current most widely used technology for fast cluster networks. The key advantage of Myrinet is that it operates in user space, thus bypassing operating system interferences and delays. • http://www.myrinet.com
QsNet II	• The next generation version of QsNet, based on a high-performance PCI-X interface as compared with QsNet's PCI interface. QsNet II is able to achieve 1064 MBytes/s, support 4096 nodes, and provide 64-bit virtual address architecture. • http://www.quadrics.com
Scalable Coherent Interface (SCI) [8]	• The first interconnection technology standard specified for cluster computing. SCI defines a directory-based cache scheme that can keep the caches of connected processors coherent, and is thus able to implement virtual shared memory. • http://www.scizzl.com

used on different cluster systems (since any given cluster seldom has more than one or two interconnects available). To be as useful as possible, interconnect performance assessments should

- be based on timing runs done on real hardware using real programs, thereby eliminating any issues related to either limitations of simulation or overtuning of synthetic benchmarks;
- use identical cluster node hardware for all runs with all network interconnects so that the results do not have to be carefully "interpreted" to account for possible performance variations due to differences in system components other than the network interconnect;
- concurrently consider as many different network interconnects as possible to avoid possible discrepancies between independent experiments done at different times; and
- include a number of real-world applications in addition to the key benchmark suites commonly used to assess cluster performance in order to provide greater confidence that the results observed are not simply those that

Table 16.2. Comparison of some interconnection technologies (updated version from [11]).

Comparison Criteria	Gigabit Ethernet	Giganet cLAN	Infiniband	Myrinet	QsNet II	Scalable Coherent Interface (SCI)
Bandwidth (MBytes/s)	< 100	< 125	850	230	1064	< 320
Latency (μs)	< 100	7-10	< 7	10	< 3	1–2
Hardware Availability	Now	Now	Now	Now	Now	Now
Linux Support	Now	Now	Now	Now	Now	Now
Max. No. of Nodes	1000s	1000s	> 1000s	1000s	4096	1000s
Protocol Implementation	Hardware	Firmware on adaptor	Hardware	Firmware on adaptor	Firmware on adaptor	Firmware on adaptor
Virtual Interface Architecture (VIA) Support	NT/Linux	NT/Linux	Software	Linux	None	Software
Message Passing Interface (MPI) Support	MVICH [12] over M-VIA [13], TCP	3rd Party	MPI/Pro [14]	3rd Party	Quadrics	3rd Party

can be expected of, in particular, well-tuned benchmark code (results for real applications are likely to be more indicative of what new cluster users can expect from their applications than are results from well-established benchmarks).

A subset of the authors (Pourreza, Eskicioglu, and Graham) have conducted performance assessments of a number of interconnects identified in Table 16.2 by taking timings when running identical applications on identical cluster nodes. Repeated isolated runs were made of a number of standard cluster computing benchmarks (including the NAS parallel benchmarks [20] and the Pallas benchmarks [21]), as well as some real world parallel applications[1] on first- and second-generation Myrinet [7], SCI [8], and Fast (100 Mbps) and Gigabit (1000 Mbps) Ethernet, and the key results are summarized below. For those readers who are interested, additional details are available in [23].

For small-scale compute clusters, most first-time cluster builders tend to choose between a very low-cost commodity interconnect (most commonly, Fast or Gigabit Ethernet) and a more expensive but higher performance interconnect (such as Myrinet or SCI). In many cases, this choice is made prior to any serious investigation of the interconnection needs of the application(s) to be run on the cluster and is often determined by simply asking the question, "Can our budget afford a fast interconnect?" This approach is undesirable and may lead to frustration with the resulting cluster performance, due to two possible reasons: one is that the selected network interconnect is inadequate for the work to be done, and the other is that the money spent on an underutilized interconnect could have been better spent on other useful components, such as faster processors and larger memory. Cluster builders could easily avoid these problems by following some simple guidelines and considering the performance characteristics of the various interconnects as determined by independent analysis.

In general, the cost of commodity interconnects is currently approximately an order of magnitude lower than the cost of high-performance interconnects. The wide price differential means that, whenever possible, it is highly desirable to use a commodity interconnect. The primary difference between commodity and high-performance interconnects is the latency of sending messages and, to a somewhat lesser extent, the bandwidth available for messaging. This naturally means that high-performance interconnects are most desirable when the applications to be run over them communicate frequently (particularly if they exchange many small messages). Applications that communicate only infrequently and that exchange larger messages often perform quite well using commodity interconnects. These general observations are confirmed by the example graphs shown below.

The cost of Fast (100 Mbps) Ethernet is now so low that a common strategy among many first-time cluster builders is to start by using Fast Ethernet for their interconnect and then upgrade when necessary (or when they have a better idea of their application characteristics and hence requirements). Given the extremely low cost of Fast Ethernet, there is little concern if the equipment is used only for

[1]Among these applications were PSTSWM (http://www.csm.ornl.gov/chammp/pstswm), a shallow-water model commonly used as part of larger Global Climate Models (GCMs), and Gromacs [22] (http://www.gromacs.org), a molecular dynamics package.

a short period of time. Furthermore, many cluster builders will keep their Fast Ethernet network to carry maintenance traffic (such as NIS, NFS, and remote logins) even after adding a high-performance interconnect to their cluster. Thus, they avoid “polluting” the high-speed interconnect with unnecessary traffic that might interfere with the actual executing programs. In addition, if the Fast Ethernet equipment is no longer required, it can be easily redeployed for usage elsewhere in the organization. Gigabit Ethernet is also increasingly being treated in a similar fashion, even though it is more expensive. But the advantages of Gigabit Ethernet are a longer lifespan as an active interconnect and the ability to support a much wider range of applications.

All the timings reported are done on an eight-node Linux cluster (with RedHat 9.0, kernel 2.4.18 smp and gcc 3.2.2). Each node has a dual Pentium III, 550-MHz processor with 512 MB of shared SDRAM memory and local IDE disks (all I/O activity in the experiments occurs on local disks to eliminate the effects of NFS access). Each node also has first- and second-generation Myrinet, GigaNet, Fast Ethernet, Gigabit Ethernet, and point-to-point SCI (Dolphin WulfKit) Network Interface Cards (NICs). All NICs except the SCI NICs are connected to dedicated switches. The SCI NICs are interconnected in a 4x2 mesh configuration. The Fast Ethernet network is also used for “maintenance” traffic, but steps were taken to ensure that traffic “maintenance” would be minimal during the experiments. Although each node has a dual processor, only a single processor is used for running most of the applications, since most small-scale cluster nodes are still uniprocessor. The results from the best performing version of MPI that is available for each interconnect are reported. The public-domain GNU compilers (gcc and g77 version 3.2.2) are used to compile all applications because GNU compilers are the most commonly used by most cluster builders. All timings are taken in isolation from other work and logins (in other words, no other applications are running while the timings are being taken, no other users are allowed to log in, and the operating systems on all nodes are only running cluster-essential software daemons). This setup represents the characteristics of a “production” cluster environment. Results for the GigaNet interconnect are not reported due to lack of a suitable MPI implementation. Also, results using the second-generation Myrinet NICs are omitted, since the cluster nodes do not support 64-bit transfers, which is one of the key benefits of second-generation Myrinet. Through experimentation, we discovered that without such wide transfers, there is little difference between the two generations of Myrinet.

The base performance of the various interconnects (in terms of bandwidth and latency) are shown in Figures 16.2 and 16.3, respectively. The relative figures for the four networks are as expected, with Fast Ethernet clearly being inferior to all the other interconnects and Gigabit Ethernet being noticeably below SCI and Myrinet despite advertising a similar raw bandwidth. Even from these results, it is clear that fast Ethernet would likely only be suitable for the most compute-bound applications.

The NPB suite consists of a number of parallel programs implemented using MPI that have a variety of communication patterns and frequencies (some of the programs are compute bound, while others are communication bound). All programs are derived from real-world program codes. The sample results shown below are for the FT and LU benchmarks. The FT-A benchmark (shown in

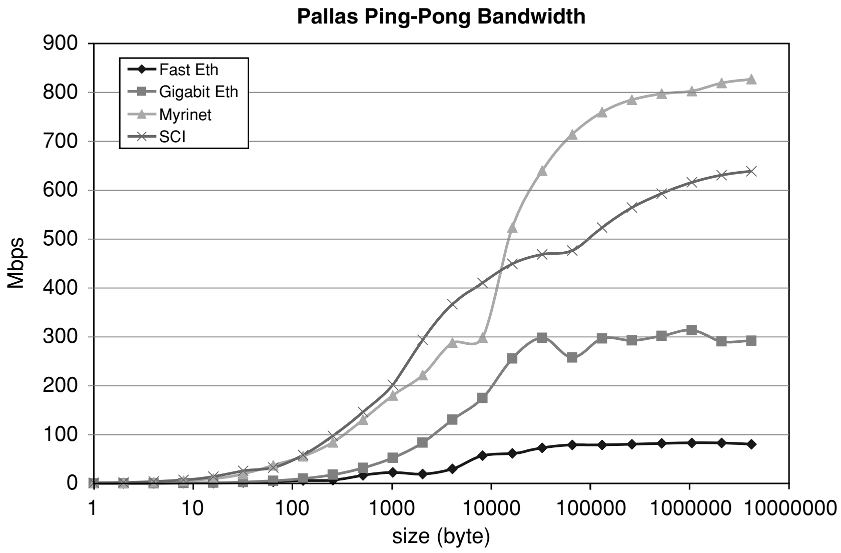

Figure 16.2. Bandwidth of four interconnects (H. Pourreza et al. [23]).

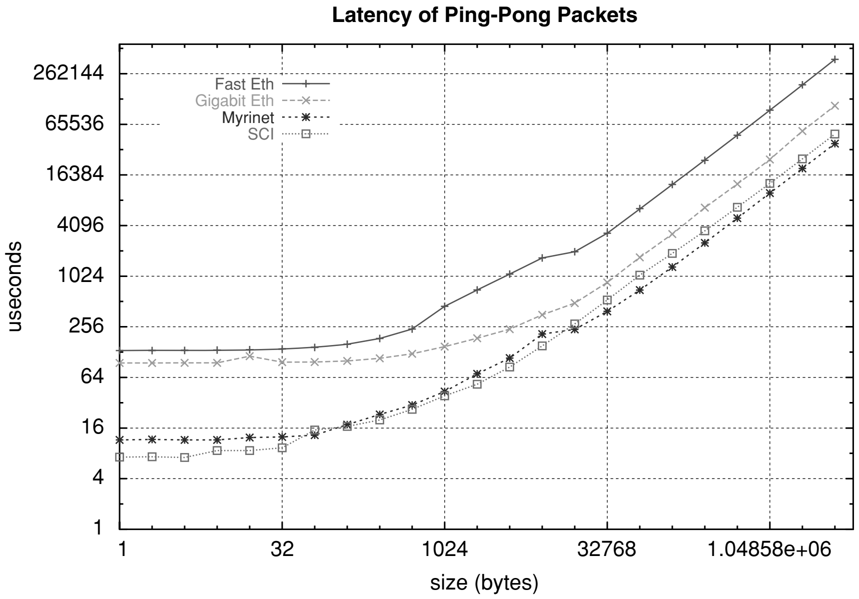

Figure 16.3. Latency of four interconnects (H. Pourreza et al. [23]).

Figure 16.4) implements a Fourier Transform (on relatively small data – the "-A" suffix) and is quite communication bound, while the LU-A benchmark (shown in Figure 16.5) implements LU decomposition (also on small data) and is relatively compute bound. The trends seen for FT are consistent with those described earlier and expected. The fact that LU is more compute bound is reflected by the improved relative performance of the commodity interconnects in Figure 16.5.

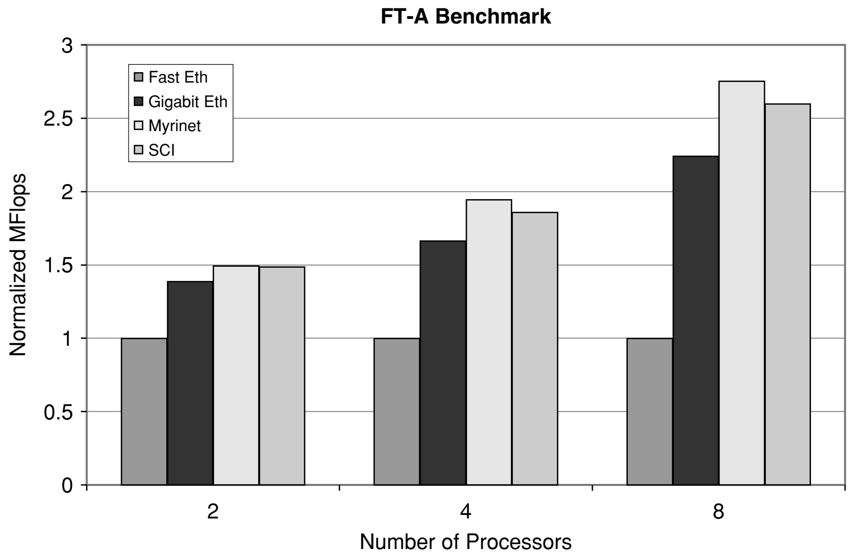

Figure 16.4. The NPB FT-A benchmark (H. Pourreza et al. [23]).

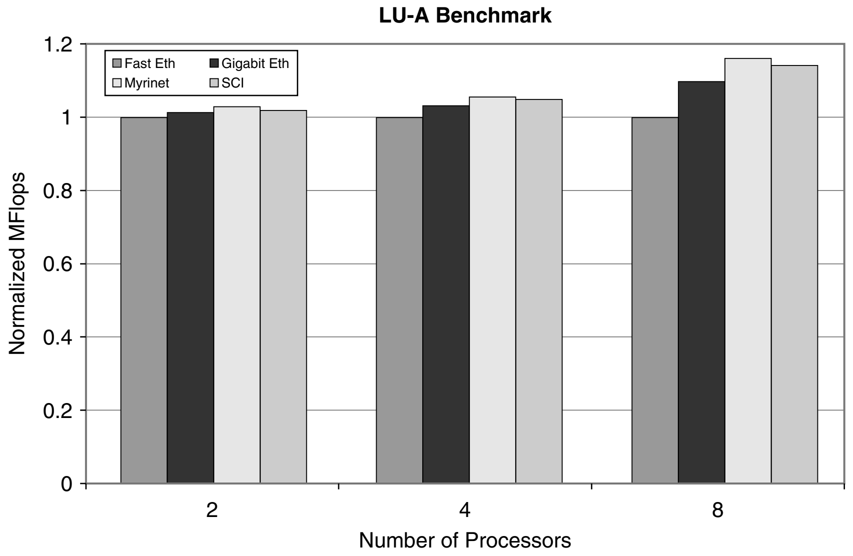

Figure 16.5. The NPB LU-A benchmark (H. Pourreza et al. [23]).

Figures 16.6 and 16.7 show speedup values for FT-A and LU-A on the four interconnects, in this case using all 16 processors in the cluster. Again, the impact of LU's being compute bound is clearly evident, reinforcing the idea that strongly compute-bound code can make good use of cheap, commodity interconnects.

It is useful to note that the overall results obtained for the large, real-world MPI applications are highly consistent with the results for the NPB suite. Both sample applications (PSTSWM and GROMACS) are moderately compute bound

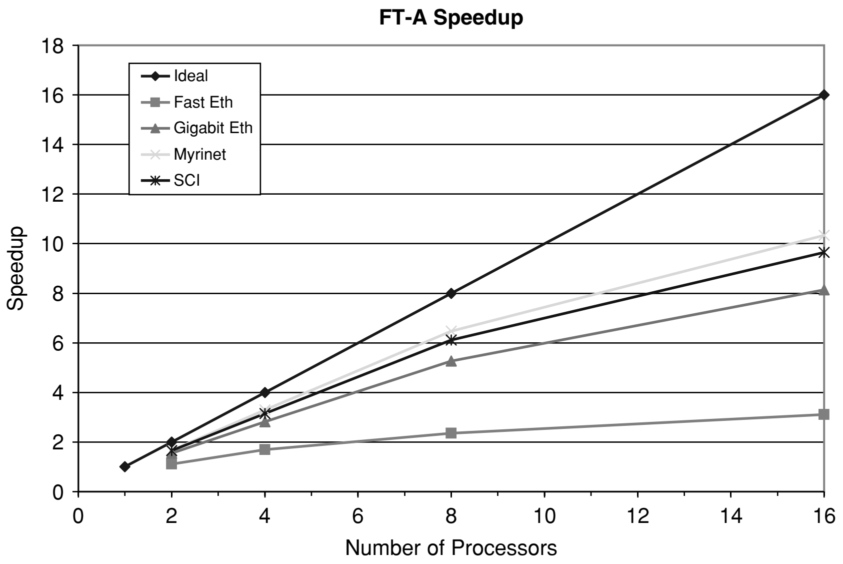

Figure 16.6. Speedup for the FT-A benchmark (H. Pourreza et al. [23]).

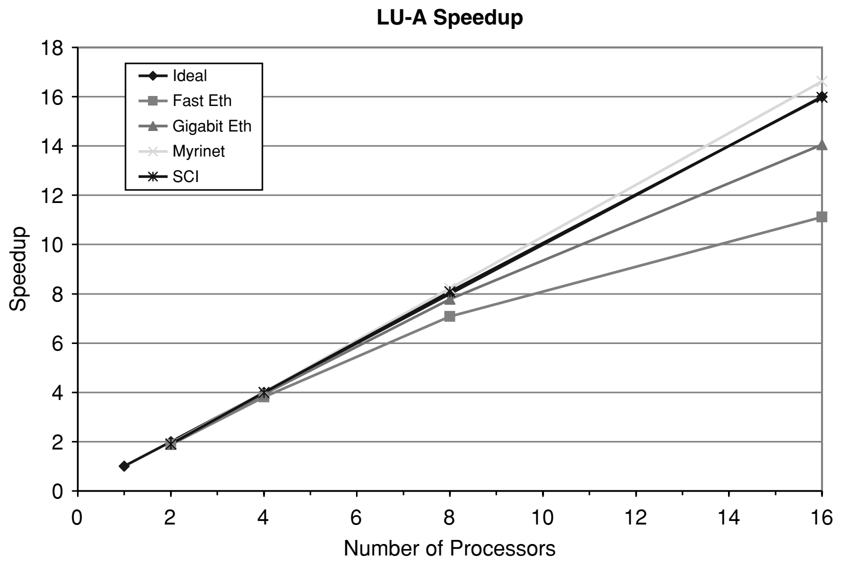

Figure 16.7. Speedup for the LU-A benchmark (H. Pourreza et al. [23]).

(PSTSWM more so than GROMACS). As a result, Fast Ethernet is an undesirable interconnect to use when running them. It is interesting to note that the effect of the point-to-point interconnects in SCI also begin to have a negative impact in GROMACS (presumably since some of the nodes must forward messages on behalf of other nodes).

From the graphs shown (as well as other related work) and considering the cost of the networking equipment, it is clear that fast Ethernet is no longer a desirable choice for a cluster interconnect unless cost is an overriding consideration (in which case, poor cluster performance for all but the most compute-bound applications must be expected). Despite its extremely low raw latency, SCI can experience problems when used in a point-to-point configuration due to the overhead of forwarding "third-party" messages. To take full advantage of SCI's capabilities, the cluster programmer must be prepared to expend significant effort to ensure that the application is structured to minimize nondirect communication. This task is often onerous and should not be taken lightly. Of all interconnects, Myrinet offers the best and most consistent performance but is also (not surprisingly) the highest-cost interconnect. Gigabit Ethernet, a commodity interconnect, offers surprisingly good performance for a fair range of applications (excluding those that are heavily communication bound) and is likely a good interconnect for many cluster builders if they expect their applications to be at least partially compute bound. Gigabit Ethernet is now significantly cheaper than Myrinet and other similar interconnect technologies (e.g., GigaNet and Infiniband). Further, with the appropriate switches, better performance can be obtained with Gigabit Ethernet using "Jumbo" frames. But, of course, the best way to select an interconnect is by benchmarking the application(s) to be run. If this is not possible, the use of the guidelines described and tests based on the NPB suite appear to be reasonable alternatives.

3 SINGLE SYSTEM IMAGE (SSI)

The Single System Image (SSI) [24] represents the view of a distributed system as a single unified computing resource. SSI provides better usability for the users, since it hides from them the complexities of the underlying distributed and heterogeneous nature of clusters. SSI can be established through one or several mechanisms implemented at various levels of abstraction in the cluster architecture: hardware, operating system, middleware, and applications.

The design goals for SSI cluster-based systems focus on complete transparency of resource management, scalable performance, and system availability in supporting user applications. Key SSI attributes that are generally considered desirable include point of entry, user interface, process space, memory space, I/O space, file hierarchy, virtual networking, job management system, and control point and management. Table 16.3 summarizes how SSI can be achieved at different levels of abstraction, with examples. The next section explains the cluster resource management systems.

3.1.1 SSI at the Operating System Level

The operating system in each of the cluster nodes provides the fundamental system support for the combined operation of the cluster. The operating system provides services such as protection boundaries, process/thread coordination, interprocess communication, and device handling, thus creating a high-level software interface for user applications.

Table 16.3. Achieving Single System Image (SSI) at different levels of abstraction.

Level of Abstraction	Description and Examples
Hardware	Implementing SSI at the Hardware layer (lowest level of abstraction) allows the user to view a cluster as a shared-memory system. Some examples are • Memory Channel [25] • Distributed Shared Memory (DSM) [26, 27]
Operating System	Modifying the existing operating system kernel to support SSI. Some examples are • MOSIX [28] • Solaris MC [29] • UnixWare [30, 31] Constructing a new operating system layer that integrates operating systems on each node. Some examples are • GLUnix [32]
Middleware	Implementing SSI at the Middleware layer is most common for clusters. Using a programming environment for development and execution of applications. Some examples are • Parallel Virtual Machine (PVM) [33] Installing resource management systems (RMSs) that manage resources and applications in the cluster. Some examples are • Condor [34] • Loadleveler [35] • Load Share Facility (LSF) [36] • Open Portable Batch System (OpenPBS) [37] • Sun Grid Engine (SGE) [38] • Libra [39]
Application	Implementing SSI at the Application layer (highest level of abstraction) provides an application-specific user interface. Some examples are • PARMON [40] • Linux Virtual Server [41] • Problem Solving Environments [42]

A cluster operating system is desired to have the following features:

- *Manageability*: Ability to manage and administrate local and remote resources.
- *Stability*: Support for robustness against system failures with system recovery.
- *Performance*: All types of operations should be optimized and efficient.
- *Extensibility*: Provide easy integration of cluster-specific extensions.
- *Scalability*: Able to scale without impact on performance.
- *Support*: User and system administrator support is essential.
- *Heterogeneity*: Portability over multiple architectures to support a cluster consisting of heterogeneous hardware components. May be achieved through the use of middleware.

There are two main types of cluster operating systems: free and commercial. Examples of free releases are Linux and MOSIX. Linux [43] is the most widely used cluster operating system, since it is free, open-source, and has a wide user and developer community. MOSIX [28] is a set of extensions built on top of the Linux kernel that enables process migration in a cluster environment to support automatic load balancing.

Commercial releases of cluster operating systems are proprietary and shipped with commercial clusters. Examples include IBM's AIX [44], SGI's IRIX [45], Sun's Solaris MC [29], HP/Compaq's Tru64 [46], SCO's Unixware [30], and Microsoft's Windows NT/2000 [47] and Windows Server family.

4 RESOURCE MANAGEMENT SYSTEM (RMS) MIDDLEWARE

A cluster resource management system (RMS) acts as a cluster middleware that implements the SSI [24] for a cluster of machines. It enables users to execute jobs on the cluster without needing to understand the complexities of the underlying cluster architecture. An RMS manages the cluster through four major branches, namely, *resource management*, *job queuing*, *job scheduling*, and *job management*.

Figure 16.8 shows a generic architecture of a cluster RMS. An RMS manages the collection of resources such as processors and disk storage in the cluster. It maintains status information on resources so as to know what resources are available, and it can thus assign jobs to available machines. The RMS uses job queues that hold submitted jobs until there are available resources to execute the jobs. When resources are available, the RMS invokes a job scheduler to select from the queues what jobs to execute. The RMS then manages the job execution processes and returns the results to the users upon job completion.

The advent of Grid computing [48] further enhances the significance of the RMS in clusters. Grid brokers can discover Grid resources such as clusters and submit the jobs via an RMS. The RMS then manages and executes the jobs before returning the results back to the Grid brokers. To enable effective resource management on clusters, numerous cluster management systems and schedulers have been designed. Table 16.4 gives a summary of some examples of RMSs.

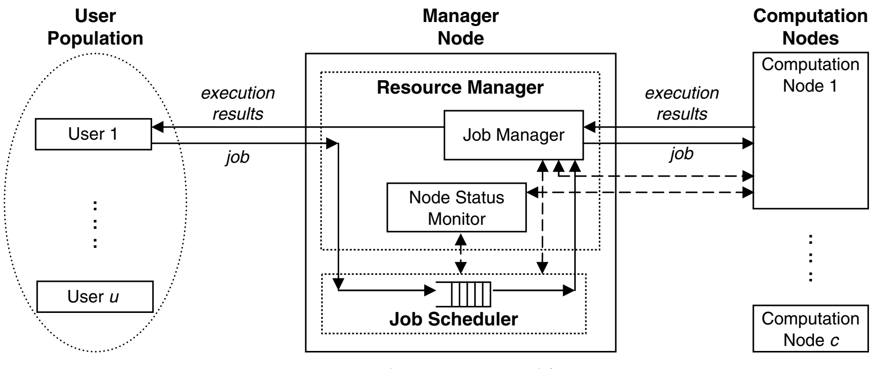

Figure 16.8. Cluster RMS architecture.

Table 16.4. Examples of Resource Management Systems (RMSs) middleware.

RMS	Organization	Brief Description and Website
Condor	University of Wisconsin–Madison	• Able to detect and execute jobs on idle nondedicated machines. • http://www.cs.wisc.edu/condor
Loadleveler	IBM	• Manages resources and jobs for IBM clusters. • http://www.ibm.com/servers/eserver/clusters/software
Load Share Facility (LSF)	Platform Computing	• Adopts a layered architecture that supports many extension utilities. • http://www.platform.com/products/LSF
Open Portable Batch System (OpenPBS)	Altair Grid Technologies	• Supports multiple scheduling policies based on extensible scheduler architecture. • http://www.openpbs.org
Sun Grid Engine (SGE)	Sun Microsystems	• The Enterprise edition supports scheduling of jobs over multiple clusters within an organization. • http://gridengine.sunsource.net
Libra	University of Melbourne	• Supports resource allocation based on computational economy principles and users' quality of service requirements. • http://www.gridbus.org/libra

Condor [34], developed by the University of Wisconsin–Madison, not only is able to manage a cluster of dedicated machines but also allows execution of jobs on nondedicated machines that are otherwise left idle. Condor can automatically detect these idle machines and use them via checkpointing and migration of job processes. Idle machines are placed into a Condor pool so that they are allocated for job execution, and are taken out of the pool when they become busy. Condor also provides extensions for using multiple Condor pools. A technique in Condor called *flocking* allows jobs submitted within a Condor pool to execute on another separate pool of machines. A version of Condor called *Condor-G* also supports the utilization of Globus [49] software that provides the infrastructure for authentication, authorization, and remote job submission of Grid resources.

LoadLeveler [35] is a resource management system developed by IBM to manage resources for IBM cluster products. LoadLeveler schedules jobs and provides functions for building, submitting, and processing jobs. When a user submits a job, LoadLeveler examines the job command file to determine what resources the job requires. Based on the jobs' requirements, it determines which machines are best suited to provide these resources and the best time for the job to be dispatched to the machines, and then dispatches the job at that time. To aid this process, LoadLeveler uses job queues to store the list of jobs that are waiting to be processed. LoadLeveler also uses a classification mechanism called *job classes* to schedule jobs to run on machines. For example, a job class called "short" contains short running jobs, while a job class called "weekend" contains jobs that are only allowed to run on the weekends. Job classes can be defined by the administrator to restrict which users can use a specific job class and what jobs can run on a particular machine.

LSF [36] is a loosely coupled cluster solution for heterogeneous systems, allowing LSF extension modules to be installed to provide advanced services.

This setup is possible due to LSF's design, which is based on a layered architecture. The base layer consisting of *Base System* and *Server Daemon* layers and provides low-level cluster services such as dynamic load balancing and transparent access to the resources available on all participating machines in the cluster. Other LSF utilities are then supported on top of the base layer at the *Utilities* layer. Examples of LSF utilities include *LSF Batch*, which provides a centralized resource management system for the cluster; *LSF MultiCluster*, which enables users to access resources on multiple LSF clusters in different geographic locations; and *LSF Analyzer*, which generates reports about the cluster by processing historical workload data.

OpenPBS [37] is the open-source version of the Portable Batch System (PBS). PBS was developed for NASA to control job execution and to manage resources for Numerical Aerodynamic Simulation (NAS) applications. It aims to be a flexible and extensible batch-processing system that supports multiple scheduling policies and job migration to meet the unique demands of heterogeneous computing networks. Currently, a new commercial version of PBS called PBSPro is available with more advanced features, such as supporting preemption, a backfilling algorithm, and advanced reservations for scheduling. OpenPBS adopts an independent scheduler architecture that enables the administrator to modify the existing default scheduling policies more easily to suit different requirements of the cluster. The administrator can create his own new customized scheduler that defines what types of resources and how much of each resource can be used by each job.

SGE [38] is currently an open-source project by Sun Microsystems which aims to establish community-driven standards that facilitate execution of computationally intensive applications. The user is able to submit batch, interactive, and parallel jobs to SGE. SGE also provides transparent workload distribution within the cluster and supports check pointing that enables jobs to migrate automatically between machines without user intervention based on load demands. The Enterprise Edition of SGE supports resource management and scheduling over multiple clusters within an organization. This setup enables the negotiation of resource and job policies to facilitate cooperation across multiple clusters.

Libra [39] is a computational economy-driven scheduling system that aims to improve the value of utility delivered to the user and the quality of services, as opposed to existing cluster RMSs that focus on a system-centric approach to resource management. Developed as part of the Gridbus Project at the University of Melbourne, Libra is designed to support allocation of resources based on the users' quality of service (QoS) requirements. It is intended to work as an add-on to the existing queuing and resource management system. The first version has been implemented as a plug-in scheduler to PBS. The scheduler offers market-based, economy-driven service for managing batch jobs on clusters by scheduling CPU time according to user-perceived value (utility), determined by the user's budget and deadline rather than by system performance considerations. Libra shows that the deadline and budget-based proportional resource allocation strategy improves both the utility of the system and user satisfaction as compared to system-centric scheduling strategies. We believe that this feature of Libra helps enforce resource allocation based on service level agreements when cluster services are offered as a utility on the Grid.

5 CLUSTER PROGRAMMING MODELS

All of a cluster's subsystems, from I/O to job scheduling to the choice of node operating system, must support the applications the cluster is designed to run. While small clusters are often constructed to support a single class of applications, such as serving Web pages or database applications, larger clusters are often called on to dedicate parts of their resources to different kinds of applications simultaneously [50, 51]. These applications often differ not only in their workload characteristics but also in the programming models they employ. The programming models employed by an application, in turn, determine the key performance characteristics of a cluster application. This section details the most important programming models used to construct cluster-aware applications; the next section provides examples of cluster applications constructed with one or more of these models.

Cluster computing programming models have traditionally been divided into categories based on the relationship of programs to the data the programs operate on [52]. The Single-Instruction, Single-Data (SISD) model defines the traditional von Neumann computer. Multiple-Instruction, Multiple-Data (MIMD) machines include most of today's clusters as well as parallel computers. In the Single-Instruction, Multiple-Data (SIMD) model, each processor executes the same program. Finally, the Multiple-instruction, Single-Data model (MISD) defines systems where multiple programs operate on the same data. MIMD has emerged as the most prevalent programming model on clusters.

In addition to dividing cluster programming models based on how programs relate to data, programming models can also be categorized on how they exploit a cluster's inherent parallelism. On that basis, cluster computing programming models can roughly be divided into two categories. The first category of models allows a serial (nonparallel) application to take advantage of a cluster's parallelism. The second category of models aids in the explicit parallelization of a program. Since cluster users are much more familiar with creating a serial program than with developing explicitly parallel applications, the first category of programming models have become dominant in cluster computing applications.

Pfister [5] coined the term *SPPS (serial program, parallel subsystem)* to describe a common technique of running a serial program on a cluster. In SPPS, many instances of a serial program are distributed on a cluster. A parallel subsystem provides input to each serial program instance and captures output from those programs, delivering that output to users. Because there are multiple programs on the clusters, operating on multiple data, SPPS is a form of MIMD. This section describes the two most common categories of SPPS programming models: distributed shared virtual memory-based systems, and message passing as illustrated by the Message Passing Interface (MPI) standard. Finally, programming models based on virtual machines are explained.

The SPPS model facilitates a division of labor in developing a cluster application: it allows a domain expert to write serial programs and delegates the task of creating an often complex parallel subsystem to highly skilled parallel programming specialists. The parallel subsystem in an SPPS-style cluster application is increasingly provided in the form of off-the-shelf middleware.

For example, Database Management Systems (DBMS) and Transaction Processing Monitors (TPM) routinely apply the SPPS technique to hide the complexity of parallel operations from users. A database query is typically submitted from a serial program to a cluster-aware DBMS subsystem responsible for query processing. That subsystem may process the query in a parallel fashion, possibly involving many cluster nodes. The query results are then returned to the serial program [53]. SPPS is used in scientific applications as well, allowing a scientist to focus on writing a serial program and submit that serial program for parallel execution on a cluster. An example is found in FermiLab's Cooperative Processes Software (CPS) [54].

When many instances of a serial program operate in parallel, those instances must coordinate work through a shared cluster resource, such as distributed shared memory or a message-passing infrastructure. The primitive operations that coordinate the work of concurrently executing serial programs on a cluster define a *coordination language* [55]. A coordination language is often described in terms of an Application Programming Interface (API) to a parallel subsystem. Such an API offers bindings to one or more programming languages. Another way to describe a coordination language is to use declarative scripting, which differs from API since it describes the required conditions and relationships and lets the computer system determine how to satisfy them. A coordination language, together with a programming language, defines the programming model of a cluster parallel application. Table 16.5 shows some examples of cluster programming models.

The *Linda tuplespace* system [56] exploits distributed shared memory [57] to facilitate the parallel execution of a serial program on a cluster. Linda defines primitive operations on a shared memory resource, allowing data items – *tuples* – to be written to that memory, read from shared memory, and deleted from shared

Table 16.5. Examples of cluster programming models.

Programming Environment	Coordination Language	Supported Programming Language	Website
Linda	API	C, Fortran	• http://www.cs.yale.edu/cswwworig/Linda/linda.html
JavaSpaces	API	Java	• http://www.sun.com/jini • http://www.jini.org
Message Queues	API	C, C++, Java	• http://www.microsoft. com/windows2000/technologies/communications/msmq/default.asp • http://www-306.ibm.com/software/integration/mqfamily • http://www.sun.com/software/products/message_queue
Message-Passing Interface (MPI)	API	C, C++, Fortran	• http://www.mpi-forum.org
JavaGroups	API	Java	• http://www.jgroups.org/javagroupsnew/docs
Parallel Virtual Machine (PVM)	API	C, C++, Fortran	• http://www.csm.ornl.gov/pvm/pvm_home.html
Parameter Sweep	Script-based constructs	Declarative programming	• http://www.csse.monash.edu.au/~davida/nimrod

memory. A tuple is similar to a database relation. A serial process with access to the shared memory writes data items to memory, marking each item with an attribute indicating that that item requires processing. Another process awaiting newly arriving tuples removes such an item from the shared memory, performs computations on that data item, and deposits the results into the shared memory. The original submitter of the job then collects all the results from the tuplespace. Each process operating on a tuplespace is typically a serial program, and the Linda system facilitates the concurrent execution of many such programs.

JavaSpaces [58] is an object-oriented Linda system that takes advantage of Java's platform-independent code execution and mobile code facility. Mobile code allows not just data but also code to move from one cluster node to another at run-time. A master node runs a JavaSpace process, providing a shared memory resource to other cluster nodes that act as workers. When a worker removes a job request from the shared JavaSpace, the operating codes for that job dynamically download to that worker. The worker executes that downloaded code and places the output of that execution into the JavaSpace. JavaSpaces, therefore, facilitates the automatic run-time distribution of code to cluster nodes. JavaSpaces also provides optional transactional access to the shared memory resource, which is especially helpful in the case of very large clusters with frequent node failures.

While *message queues* first became popular with SMPs as a load-distribution technique [5], distributed message queues have become increasingly popular with clusters as well. Distributed queues are a form of distributed shared memory with the added property of ordered message delivery. Most message queues in commercial use also provide transactional access to the queue.

While distributed shared memory facilitates communication via a shared cluster resource, the message-passing model coordinates work by sending and receiving messages between processes. The *Message-Passing Interface (MPI)* [10] standard defines a programming model for message passing, along with a series of functions to support that programming model. Language bindings to MPI functions exist for a variety of languages, such as C, C++, and Fortran. In addition, a Java-based version of MPI specifies how this programming model can be used from Java programs [59].

In the MPI model, a set of processes are started at program startup. There is one process per processor, but each processor may execute a different process. Thus, MPI is a message-passing programming model for MIMD systems. During program execution, the number of processes in an MPI program remains fixed.

MPI processes are named, and processes send and receive messages in a point-to-point fashion based on process name. Processes can be grouped, and collective communication functions can be used to perform global operations on a group, such as broadcast and synchronization. Message exchanges in MPI can convey the communication context in which a message exchange occurs. MPI even offers a process the ability to probe its environment and to probe for messages, allowing MPI programs to use both synchronous and asynchronous message exchange.

In addition to defining message exchange semantics, the MPI programming model provides explicit support for the construction of parallel programming libraries suitable for execution on a cluster [60]. Libraries written for the MPI standard are portable and can be reused by higher-level application software. The chief MPI tools for library construction are communication contexts, process groups, virtual topologies, and cached attributes. An MPI construct called a *communicator* encapsulates all these functions in a reusable fashion.

An MPI process group is an ordered collection of processes that defines the scope for process names and for collective communication. Because the system can differentiate between processes sharing a context, communication context allows partitioning of information during message exchange. Separate contexts by MPI libraries insulate communication internal to the library execution from external communication.

An MPI *communicator* can be thought of as a group identifier associated with a context. *Intra*-communicators operate on a single group, whereas *inter*-communicators are used for point-to-point communication between two groups of processes. While intra-communicators let a library developer encapsulate communication internal to a library, inter-communicators bind two groups together, with communication contexts shared by both groups.

The MPI standard limits itself to defining message passing semantics and functions and to defining the primitives required for reusable libraries. MPI does not provide an infrastructure for program construction and task management. Those responsibilities are left to MPI implementations. The most popular implementation of the MPI standard is MPICH [61]. Available as an open-source package, MPICH supports the latest MPI 2 standard [10]. MPI has recently been extended for Grid communication with the MPICH-G library [62].

While MPI provides a comprehensive message-passing library of over 150 functions, several of the key MPI concepts have influenced the design of smaller, special-purpose message-passing libraries. *JavaGroups* [63] is an open-source Java toolkit for reliable multicast communication on a cluster, or even on a wide-area network. Similar to MPI, JavaGroups facilitates the creation of processes groups and also allows the processes to dynamically join or leave groups. An automatic membership-detection infrastructure in JavaGroups handles the removal of non-responsive group members. Communication in JavaGroups can be point-to-point from one group member to another, or group communication (from one group member to an entire group). Several cluster-aware Java applications rely on JavaGroups for group message passing, such as the JBoss application server [64].

In addition to message passing and virtual shared memory, programming models based on virtual machines also facilitate the parallel execution of serial programs on a cluster (SPPS). The *Parallel Virtual Machine (PVM)* [33] consists of daemon processes, to be executed on all cluster nodes, and an application programming library. PVM presents heterogeneous cluster resources as a homogenous environment to a program. The PVM daemon process can run on machines with widely differing computational capabilities, from notebooks to supercomputers. PVM offers language bindings to C and C++ as a set of API functions, and a binding to Fortran as a series of subroutines.

Using PVM is straightforward. First, the user starts up the PVM daemons on the set of cluster nodes he or she wishes to incorporate into the shared cluster resource. Next, the user writes a set of serial programs, includes calls to the PVM routines, and links those programs with the PVM libraries. Finally, the user executes a "master" program on one machine. That program, through the PVM API calls, will spawn other programs, "slaves," on other cluster nodes as needed. Those programs communicate with each other through a simple message-passing mechanism. The run concludes with the termination of the initial serial master program. Code for each slave program must be made available to the PVM daemon on each node prior to executing the master.

Each serial program running on the nodes that make up a PVM instance typically runs as a task on a host's operating system. Therefore, the unit of parallelism in PVM is a *task*. A group of such tasks make up a PVM program. PVM tasks are identified by a task ID, typically an integer. Task IDs are assigned by PVM, and intertask communication takes place in PVM based on task IDs. Currently, PVM does not use MPI for intertask message-passing communication. However, an effort is under way to incorporate the benefits of MPI into PVM [65].

A PVM task can belong to one or more task groups during its execution. Task groups in PVM are dynamic: a task can join or leave a task group during its execution without having to notify other tasks in a given group. PVM also supports group communication primitives: a task can broadcast a message not only to other members of the group to which it currently belongs but also to tasks belonging to other task groups.

Parameter Sweep supports parallelism by executing the same program with different parameters in parallel as individual processes. An example of a tool that supports parameter sweep is Nimrod [66], which performs parameter sweep over a cluster of computers. Nimrod provides a script-based declarative programming language for defining parameter-sweep specification. It allows users to define varying values for key parameters to be studied in a simple script. Using the script, it automatically generates the required data files for each program, depending on the set of parameters. Nimrod then selects a computer for each program, transfers the generated data files and other required files for each program to the selected workstation for execution, and transfers back the execution results.

6 CLUSTER APPLICATIONS

One category of applications where cluster computing is rapidly becoming the architecture of choice is Grand Challenge Applications (GCA). Grand Challenge Applications (GCAs) [67] are defined as fundamental problems in science and engineering with broad economic and scientific impact whose solution can be advanced by applying High Performance Computing and Communications (HPCC) technologies.

The high scale of complexity in GCAs demands an enormous amount of resource needs, such as processing time, memory space, and communication bandwidth. A common characteristic of GCAs is that they involve simulations that are computationally intensive. Examples of GCAs are applied fluid dynamics, environmental modeling, ecosystem simulation, biomedical imaging, biomechanics, molecular biology, molecular design, cognition, and computational sciences.

Other than GCAs, cluster computing is also being applied in other applications that demand high availability, scalability, and performance. Clusters are being used as replicated storage and backup servers that provide the essential fault tolerance and reliability for critical applications. For example, the Internet search engine Google [68] uses cluster computing to provide reliable and efficient Internet search services. There are also many commercial cluster products designed for distributed databases and web servers. In the following subsections, we will discuss some of these applications and examine how cluster computing is used to enable them.

6.1.1 Google Search Engine

Internet search engines enable Internet users to search for information on the Internet by entering specific keywords. A widely used search engine, Google [68] uses cluster computing to meet the huge quantity of worldwide search requests that comprise a peak of thousands of queries per second. A single Google query needs to use at least tens of billions of processing cycles and access a few hundred megabytes of data in order to return satisfactory search results.

Google uses cluster computing as its solution to the high demand of system resources, since clusters have better price–performance ratios than alternative high-performance computing platforms, and also use less electrical power. Google focuses on two important design factors: reliability and request throughput.

Google is able to achieve reliability at the software level so that a reliable computing infrastructure can be constructed on clusters of 15,000 commodity PCs distributed worldwide. The services for Google are also replicated across multiple machines in the clusters to provide the necessary availability. Google maximizes overall request throughput by performing parallel execution of individual search requests. This means that more search requests can be completed within a specific time interval.

A typical Google search consists of the following operations:

1. An Internet user enters a query at the Google webpage.
2. The web browser searches for the Internet Protocol (IP) address via the www.google.com Domain Name Server (DNS).
3. Google uses a DNS-based load-balancing system that maps the query to a cluster that is geographically nearest to the user so as to minimize network communication delay time. The IP address of the selected cluster is returned.
4. The web browser then sends the search request in Hypertext Transport Protocol (HTTP) format to the selected cluster at the specified IP address.
5. The selected cluster then processes the query locally.
6. A hardware-based load balancer in the cluster monitors the available set of Google Web Servers (GWSs) in the cluster and distributes the requests evenly within the cluster.
7. A GWS machine receives the request, coordinates the query execution, and sends the search result back to the user's browser.

Figure 16.9 shows how a GWS operates within a local cluster. The first phase of query execution involves index servers consulting an inverted index that matches each query keyword to a matching list of documents. Relevance scores are also computed for matching documents so that the search result returned to the user is ordered by score. In the second phase, document servers fetch each document from disk to extract the title and the keyword-in-context portion of the document. In addition to the two phases, the GWS also activates the spell checker

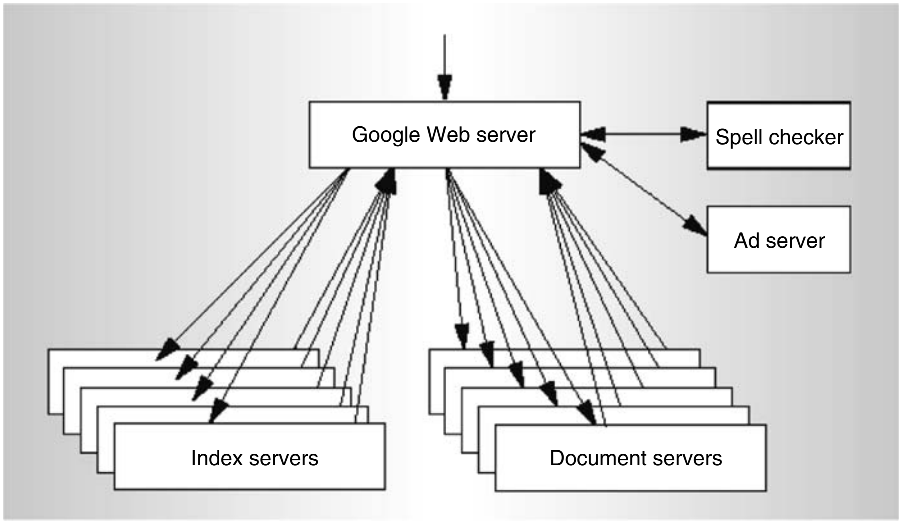

Figure 16.9. Google query-serving architecture (L. A. Barroso et al. [68]).

and the ad server. The spell checker verifies that the spelling of the query keywords is correct, while the ad server generate advertisements that relate to the query and may therefore interest the user.

6.1.2 Petroleum Reservoir Simulation

Petroleum reservoir simulation facilitates a better understanding of petroleum reservoirs, which is crucial to better reservoir management and more efficient oil and gas production. Petroleum reservoir simulation is an example of GCA, since it demands intensive computations in order to simulate geological and physical models. For example, the Center for Petroleum and Geosystems Engineering of the University of Texas at Austin is constructing a new parallel petroleum reservoir simulator called General Purpose Adaptive Simulator (GPAS) [69] using a cluster of 64 dual-processor servers with a total of 128 processors.

A typical petroleum reservoir simulator consists of a coupled set of nonlinear partial differential equations and constitutive relations that describe the physical processes occurring in a petroleum reservoir. There are two most widely used simulators. The first is the black oil simulator, which uses water, oil, and gas phases for modeling fluid flow in a reservoir. The second is the compositional simulator, which uses phases with different chemical species for modeling physical processes occurring in a reservoir. Previously, compositional simulators were used less often, since they are more complicated and thus require more intensive memory and processing requirements. With the advent of cluster computing, more researchers are using compositional simulators that use more data to characterize reservoirs.

The GPAS [69] is a compositional petroleum reservoir simulator that can perform more accurate, efficient and high-resolution simulation of fluid flow in permeable media. It uses a finite-difference method that divides a continuous domain into smaller cells to solve the governing partial differential equations. The higher number of cells produces more accurate results but requires more computation time. A fully implicit solution results in a structure of nonlinear equations that are then resolved using Newton's method. However, large sparse linear systems of equations are needed to obtain a numerical solution of these nonlinear equations. Therefore, the Portable Extensible Toolkit for Scientific Computation (PETSc) [70], a set of tools for solving partial differential equations, is used to solve these linear systems.

To handle the parallel processing requirements, an Integrated Parallel Accurate Reservoir Simulator (IPARS) framework has been developed to separate the physical model development from parallel processing. IPARS provides input and output, memory management, domain decomposition, and message passing among processors to update overlapping regions. Communications between the simulator framework and a physical model are carried out through FORTRAN subroutine calls provided within the IPARS, thus hiding the complexities from the physical model developers, who only need to call the FORTRAN subroutines to perform corresponding tasks.

6.1.3 Protein Explorer

The Bioinformatics Group at RIKEN Genomic Sciences Center in Japan is currently building the world's first petaflops supercomputer. The *Protein Explorer (PE)* system [71] will be a specialized system for molecular dynamics simulations—specifically, protein simulations—and is expected to be ready in early 2006. The PE system will be a PC cluster equipped with special-purpose engines to calculate nonbonded interactions between molecular atoms. These calculations constitute the most time-consuming portion of the simulations. The PE project is motivated by the national Protein 3000 project in Japan that was initiated in 2002 with the goal of solving the structures of 3,000 proteins by the year 2007.

Figure 16.10 shows the components of the PE system. It will be a cluster of 256 dual-processor nodes giving a total of 512 processors, connected via Gigabit Ethernet. Each cluster node has two special-purpose engine boards (with 12 MDGRAPE-3 chips on each board) connected to it, giving it a total of 6,144 chips.

The cluster nodes will transmit the coordinates and the other data of particles for the molecular dynamics simulation to the special-purpose engines, which then calculate the nonbonded forces such as Coulomb force and van der Walls force between particles before returning the results to the hosts. In other words, the special-purpose engines only focus on computing the most complex portion of the simulation, that is, calculating the nonbonded forces. All the coordination and other calculations are handled by the cluster nodes themselves.

6.1.4 Earthquake Simulation

Earthquake simulation is classified as a GCA, given its high modeling and computational complexities [72]. First, multiple spatial scales characterize the earthquake source and basin response, ranging from tens of kilometers for the basin dimensions to hundreds of kilometers for earthquake sources. Second,

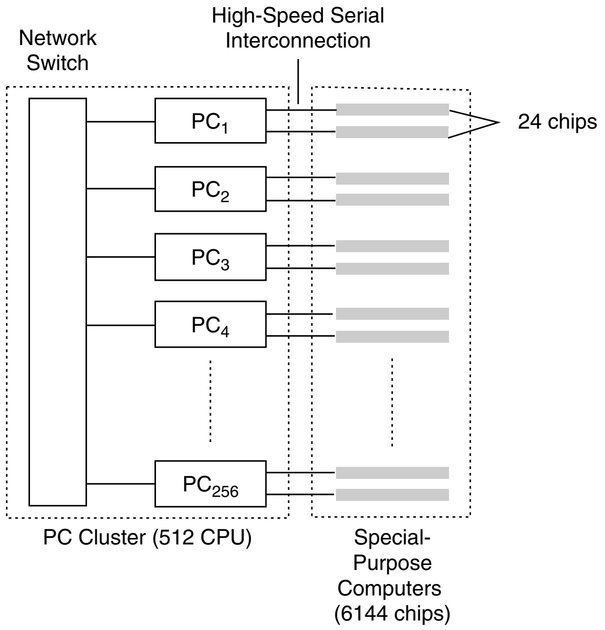
Network Switch
High-Speed Serial Interconnection
PC1
PC2
PC3
PC4
PC256
24 chips
PC Cluster (512 CPU)
Special-Purpose Computers (6144 chips)

Figure 16.10. Block diagram of Protein Explorer system (M. Taiji et al. [71]).

temporal scales differ—from hundredths of a second for depicting the highest frequencies of the earthquake source to several minutes of shaking within the basin. Third, many basins have highly irregular geometry. Fourth, the soils in the basins comprise heterogeneous material properties. And fifth, there remains great uncertainty into the modeling process due to the indirect observation of geology and source parameters.

An ongoing research project in the United States [72] focuses on developing the capability for generating realistic inversion-based models of complex basin geology and earthquake sources. This capability could then be used to model and forecast strong ground motion during earthquakes in large basins such as Los Angeles (LA). Ground motion modeling and forecasting is essential to studying which structures will become vulnerable during the occurrence of an earthquake. This modeling can be used to design future earthquake-resistant structures and to retrofit existing structures so as to mitigate the effects of an earthquake. The Los Angeles region was chosen for the case study because it is the most highly populated seismic region in the USA, has well-characterized geological structures (including a varied fault system), and has extensive records of past earthquakes.

The earthquake simulation is conducted using a terra-scale HP AlphaServer cluster that has 750 quadruple-processor nodes at the Pittsburgh Supercomputing Center (PSC). It simulates the 1994 Northridge earthquake in the Greater LA Basin at 1 Hz maximum frequency resolution and 100 m/s minimum shear wave velocity. The resulting unstructured mesh contains over 100 million grid points and 80 million hexahedral finite elements, ranking it as one of the largest unstructured mesh simulations ever conducted. It is also the most highly resolved simulation of the Northridge earthquake ever done. It sustains nearly a teraflops over 12 hours in solving the 300 million wave propagations.

The simulations are based on multiresolution mesh algorithms that can model the wide range of length and time scales depicting earthquake response. Figure 16.11 shows the process of generating a mesh using the etree method. That method is used for earthquake simulations in heterogeneous basins, where the shear wave velocity and maximum resolved frequency determine the local element size. At the initial "construct" step, an octree is constructed and stored on disk. The decompositions of the octants are dependent on the geometry or physics being modeled, thus resulting in an unbalanced octree. Then the balance step recursively decomposes all the large octants that violate the 2-to-1 constraint until there are no more illegal conditions, thus creating a balanced octree. Finally, in the transform step, mesh-specific information such as the element–node relationship and the node coordinates are derived from the balanced octree and separately stored in two databases: one for the mesh elements, another for the mesh nodes.

For the balancing step, the whole domain is first partitioned into equal-size blocks. Then, internal balancing enforces the 2-to-1 constraint within each block. Finally, boundary balancing is used to resolve interactions between adjacent blocks. This local balancing step is very effective, since it can achieve a speedup ranging from 8 to 28, depending on the size of the meshes being balanced.

Figure 16.12 shows snapshots at different times of the simulation of the wave propagation throughout the basin based on the 1994 Northridge earthquake. These snapshots reflect the directivity of the ground motion along the strike from the epicenter and the concentration of motion near the fault corners.

6.1.5 Image Rendering

The Scientific Computing and Imaging (SCI) Institute at University of Utah has explored cluster-based scientific visualization [73] using a 32-node visualization cluster composed of commodity hardware components connected with a high-speed network. The OpenGL [74] scientific visualization tool, Simian, has been modified to create a cluster-aware version of Simian that supports parallelization by making explicit use of remote cluster nodes through a message-passing interface (MPI). Simian is able to generate 3D images for fire-spread simulations that model scenarios such as when a missile located within a pool of jet fuel catches fire and explodes. The use of image rendering for fire-spread simulations enables researchers to a better visualize the destructive effects.

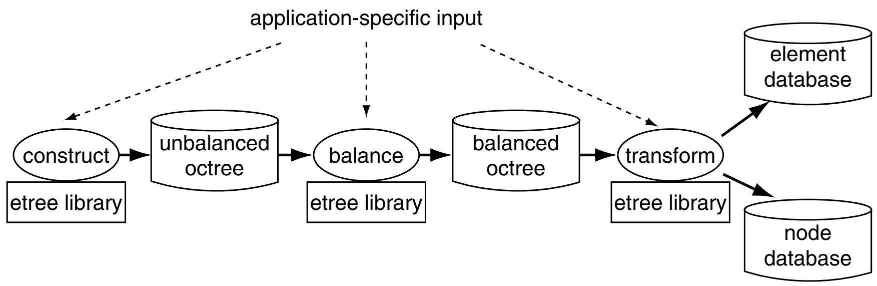

Figure 16.11. The etree method of generating octree meshes (V. Akcelik et al. [72]).

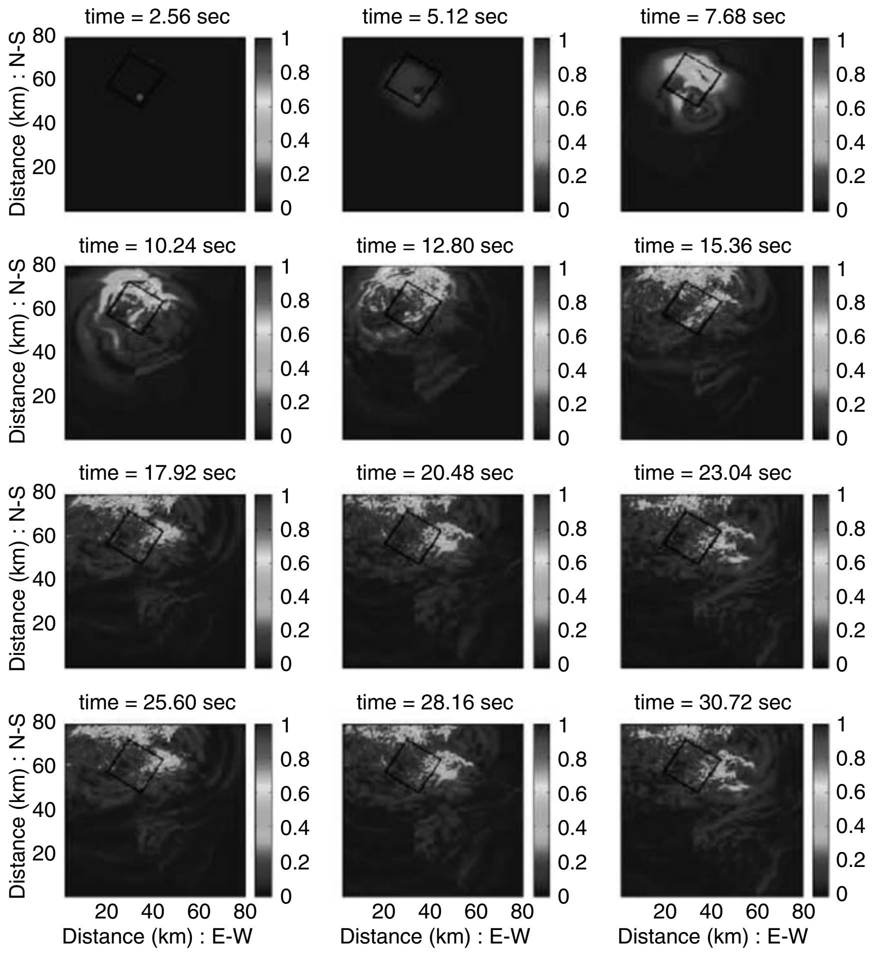

Figure 16.12. Snapshots of propagating waves from simulation of 1994 Northridge earthquake (V. Akcelik et al. [72]).

Normally, Simian uses a swapping mechanism to manage datasets that are too large to load into the available texture memory, resulting in low performance and interactivity. For the cluster-aware Simian, large datasets are divided into subvolumes that can be distributed across multiple cluster nodes, thus achieving the interactive performance. This "divide-and-conquer" technique first decomposes the dataset into subvolumes before distributing the subvolumes to multiple remote cluster nodes. Each node is then responsible for rendering its subvolume by using the locally available graphics hardware. The individual results are finally combined using a binary-swap compositing algorithm to generate the final image. This enables the cluster-aware Simian to visualize large-scale datasets and maintain interactive rates without the need for texture swapping.

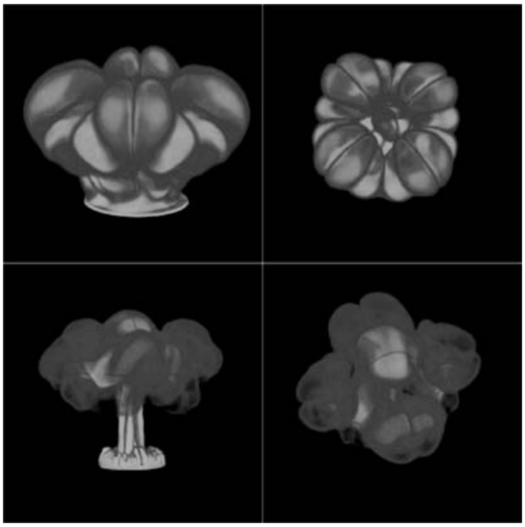

Figure 16.13. Visualization of fire-spread datasets (C. Gribble et al. [73]).

Figure 16.13 shows the visualization of two fire-spread datasets simulating a heptane pool fire, generated by the cluster-aware version of Simian using eight cluster nodes. The top row of Figure 16.13 shows two views (side and top views) of the h300_0075 dataset, while the bottom row shows the h300_0130 dataset.

SUMMARY

We have discussed the motivation for cluster computing as well as the technologies available for building cluster systems. The emphasis placed on the use of commodity-based hardware and software components to achieve high performance, availability, and scalability means that cluster computing is a more cost-effective platform compared with traditional high-performance platforms.

We have examined the various cluster-specific components such as interconnection technology, operating system, middleware, and programming model. We have discussed the performance of a number of common cluster interconnects. We have also presented various parallel programming models and concepts of single system image and its realization at the cluster resource management level. The rapid research and development of cluster hardware and software components has enhanced the usage of cluster computing for a wide variety of applications, both in scientific and commercial domains. We have studied some of these applications and how clusters are used to implement them.

For recent developments and innovations in cluster computing technologies and their applications, we recommend readers to refer to the Proceedings of the IEEE Task Force on Cluster Computing (TFCC) [75] events such as the ClusterXY [76] and CCGridXY [77] conference series.

REFERENCES

[1] R. Buyya (ed) (1999): *High Performance Cluster Computing: Architectures and Systems*, *1*, Prentice Hall.

[2] The Beowulf Cluster site, http://www.beowulf.org

[3] T. E. Anderson, D. Culler, and D. A. Patterson (1995): A Case for NOW (Network of Workstations), *IEEE Micro*, *15*(1), 54–64.

[4] A. Chien, S. Pakin, M. Lauria, M. Buchanan, K. Hane, L. Giannini, and J. Prusakova (1997): High Performance Virtual Machines (HPVM): Clusters with Supercomputing APIs and Performance, *Proc. 8th SIAM Conference on Parallel Processing for Scientific Computing (PP97)*, Minneapolis, USA.

[5] G. F. Pfister (1998): *In Search of Clusters*, *2nd Edition*, Prentice Hall.

[6] T. Shanley (2002): *Infiniband Network Architecture*, Addison-Wesley.

[7] N. J. Boden, D. Cohen, R. E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and Wen-King Su (1995): Myrinet: A Gigabit-per-second Local Area Network, *IEEE Micro*, *15*, (1), 29–36.

[8] K. Alnaes, E. H. Kristiansen, D. B. Gustavson, and D. V. James (1990): Scalable Coherent Interface, *Proc. 1990 IEEE International Conference on Computer Systems and Software Engineering (CompEuro '90)*, Tel-Aviv, Israel, pp. 446–453.

[9] D. Cameron and G. Regnier (2002): *Virtual Interface Architecture*, Intel Press.

[10] Message Passing Interface (MPI) Forum, http://www.mpi-forum.org

[11] M. Baker, A. Apon, R. Buyya, and H. Jin (2002): Cluster Computing and Applications, *Encyclopedia of Computer Science and Technology*, 45 (Supplement 30), A. Kent and J. Williams (eds), Marcel Dekker, pp. 87–125.

[12] MVICH: MPI for VIA, http://old-www.nersc.gov/research/FTG/mvich

[13] M-VIA: VIA for Linux, http://old-www.nersc.gov/research/FTG/via

[14] MPI/PRO, http://www.mpi-softtech.com

[15] M. Banikazemi, J. Liu, D. K. Panda, and P. Sadayappan (2001): Implementing TreadMarks over Virtual Interface Architecture on Myrinet and Gigabit Ethernet: Challenges, Design Experience, and Performance Evaluation, *Proc. 2001 International Conference on Parallel Processing (ICPP '01)*, Valencia, Spain, pp. 167–174.

[16] Z. Lan and P. Deshikachar (2003): Performance Analysis of a Large-Scale Cosmology Application on Three Cluster Systems, *Proc. 2003 IEEE International Conference on Cluster Computing (Cluster 2003)*, Hong Kong, China, pp. 56–63.

[17] A. J. van der Steen (2003): An Evaluation of Some Beowulf Clusters, *Cluster Computing*, *6*(4), 287–297.

[18] H. Chen, P. Wyckoff, and K. Moor (2000): Cost/Performance Evaluation of Gigabit Ethernet and Myrinet as Cluster Interconnects, *Proc. 2000 Conference on Network and Application Performance (OPNETWORK 2000)*, Washington, USA.

[19] J. Hsieh, T. Leng, V. Mashayekhi, and R. Rooholamini (2000): Architectural and Performance Evaluation of GigaNet and Myrinet Interconnects on Clusters of Small-Scale SMP Servers, *Proc. 2000 ACM/IEEE Conference on Supercomputing (SC2000)*, Dallas, USA.

[20] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and W. K. Weeratunga (1991): The NAS Parallel Benchmarks, *International Journal of Supercomputing Applications*, *5*(3), 63–73.

[21] Pallas MPI Benchmarks, http://www.pallas.com/e/products/pmb/index.htm

[22] E. Lindahl, B. Hess, and D. van der Spoel (2001): GROMACS 3.0: a package for molecular simulation and trajectory analysis, *Journal of Molecular Modeling*, *7*(8), 306–317.

[23] H. Pourreza, R. Eskicioglu, and P. C. J. Graham (2004): Preliminary Performance Assessment of Four Cluster Interconnects on Identical Hardware, *Proc. 18th International Symposium on High Performance Computing Systems and Applications (HPCS2004)*, Winnipeg, Canada.

[24] R. Buyya, T. Cortes, and H. Jin (2001): Single System Image (SSI), *International Journal of High Performance Computing Applications*, *15*(2), 124–135.

[25] Hewlett-Packard, Memory Channel, http://www.hp.com/techservers/systems/symc.html

[26] A Comprehensive Bibliography of Distributed Shared Memory, http://dsm-biblio.cs.umanitoba.ca/WEB

[27] Distributed Shared Memory (DSM), http://www.cs.umd.edu/~keleher/dsm.html

[28] A. Barak and O. La'adan (1998): The MOSIX multicomputer operating system for high performance cluster computing, *Future Generation Computer Systems*, *13*(4–5), 361–372.

[29] Y. A. Khalidi, J. M. Bernabeu, V. Matena, K. Shirriff, and M. Thadani (1995): Solaris MC: A Multi-Computer OS, *Sun Microsystems Technical Report TR-95-48*.

[30] SCO Unixware, http://www.thescogroup.com/products/unixware713

[31] B. Walker and D. Steel (1999): Implementing a Full Single System Image UnixWare Cluster: Middleware vs. Underware, *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA99)*, Las Vegas, USA, pp. 2767–2773.

[32] D. P. Ghormley, D. Petrou, S. H. Rodrigues, A. M. Vahdat, and T. E. Anderson (1998): GLUnix: A Global Layer Unix for a Network of Workstations, *Software: Practice and Experience*, *28*(9), 929–961.

[33] V. S. Sunderam (1990): PVM: A framework for parallel distributed computing, *Concurrency: Practice and Experience*, *2*(4), 315–339.

[34] University of Wisconsin-Madison, *Condor Version 6.6.2 Manual*, 2004.

[35] IBM, *LoadLeveler for AIX 5L V3.2 Using and Administering*, *SA22-7881-01*, 2003.

[36] Platform Computing, *LSF V4.1 Administrator's Guide*, 2001.

[37] Altair Grid Technologies: *OpenPBS Release 2.3 Administrator Guide*, 2000.

[38] Sun Microsystems, *Sun ONE Grid Engine*, *Administration and User's Guide,* Oct. 2002.

[39] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya (2004): Libra: A Computational Economy based Job Scheduling System for Clusters, *Software: Practice and Experience*, *34*(6), 573–590.

[40] R. Buyya (2000): PARMON: a portable and scalable monitoring system for clusters, *Software: Practice and Experience*, *30*(7), 723–739.

[41] W. Zhang (2000): Linux Virtual Server for Scalable Network Services, *Linux Symposium*, Ottawa, Canada.

[42] E. Gallopoulos, E. Houstis, and J. R. Rice (1994): Computer as thinker/doer: problem-solving environments for computational science, *IEEE Computational Science and Engineering*, *1*(2), 11–23.

[43] Linux Online, http://www.linux.org

[44] IBM AIX: UNIX Operating System, http://www.ibm.com/servers/aix

[45] SGI IRIX, http://www.sgi.com

[46] HP/Compaq Tru64, http://www.tru64unix.compaq.com

[47] Microsoft Windows 2000: http://www.microsoft.com/windows2000

[48] I. Foster and C. Kesselman (eds), (1999): *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kauffman Publishers.

[49] I. Foster and C. Kesselman (1997): Globus: A Metacomputing Infrastructure Toolkit, *International Journal Supercomputer Applications*, *11*(2), 115–128.

[50] R. Evard, N. Desai, J. Navarro, and D. Nurmi (2002): Clusters as large-scale development facilities, *Proc. 2002 IEEE International Conference on Cluster Computing (Cluster 2002)*, Chicago, USA.

[51] N. Pundit (2002): CPlant: The Largest Linux Cluster, *Newsletter of IEEE Task Force on Cluster Computing*, *4*(1), Fall.

[52] M. Flynn (1972): Some computer organizations and their effectiveness, *IEEE Transactions on Computers*, *21*(9), 948–960.

[53] S. Ghandeharizadeh and F. Sommers (2001): Parallel Databases and Decision Support Systems, *Handbook of Data Mining and Knowledge Discovery*, W. Klosgen and J. Zytkow (eds), Oxford University Press.

[54] T. Nash (1992): Cluster Computing at FermiLab, presentation to IEEE SSS.

[55] G. Papadopoulos and F. Arbab (1998): Coordination models and languages, *Centrum voor Wiskunde en Informatica Technical Report SEN-R9834*.

[56] N. Carriero and D. Gelernter (1990): *How to Write Parallel Programs: A First Course*, MIT Press.

[57] K. Li and P. Hudak (1986): Memory Coherence in Shared Virtual Memory Systems, *Proc. 5th Annual ACM Symposium on Principles of Distributed Computing*, Calgary, Canada, pp. 229–239.

[58] J. Waldo et al. (2001): *The Jini Specifications*, *2nd Edition*, Addison-Wesley.

[59] B. Carpenter, V. Getov, G. Judd, T. Skjellum, and G. Fox (1998): MPI For Java: Position Document and Draft API Specification, *Java Grande Forum Technical Report JGF-TR-03*.

[60] A. Skjellum, N. E. Doss, and P. V. Bangalore (1993): Writing Libraries in MPI, *Proc. Scalable Parallel Libraries Conference,* Mississippi State, USA, pp. 166–173.

[61] W. Gropp, E. Lusk, N. Doss, and A. Skjellum (1996): A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard, *Parallel Computing*, *22*(6), 789–828.

[62] I. Foster, and N. Karonis (1998): A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems, *Proc. 1998 IEEE/ACM Supercomputing Conference (SC98)*, Orlando, USA.

[63] JavaGroups, http://www.jgroups.org/javagroupsnew/docs

[64] JBoss, http://www.jboss.org

[65] G. Fagg and J. Dongarra (1996): PVMPI: An Integration of the PVM and MPI Systems, *Calculateurs Parallèles*, *8*(2), 151–166.

[66] D. Abramson, R. Sosic, J. Giddy, and B. Hall (1995): Nimrod: A Tool for Performing Parametised Simulations Using Distributed Workstations, *Proc. 4th IEEE Symposium on High Performance Distributed Computing (HPDC95)*, Pentagon City, USA, pp. 112–121.

[67] National Coordination Office for Informational Technology Research and Development, Grand Challenge Applications, *High Performance Computing and Communications: Foundation for America's Information Future*, http://www.nitrd.gov/pubs/blue96/section.2.6.0.html

[68] L. A. Barroso, J. Dean, and U. Holzle (2003): Web search for a planet: The Google cluster architecture, *IEEE Micro*, *23*(2), 22–28.

[69] T. Uetani, B. Guler, and K. Sepehrnoori (2002): Parallel Reservoir Simulation on High Performance Clusters, *Proc. 6th World Multi-Conf. on Systemics, Cybernetics and Informatics (SCI2002)*, *V.*

[70] Portable Extensible Toolkit for Scientific Computation (PETSc), http://www-unix.mcs.anl.gov/petsc

[71] M. Taiji, T. Narumi, Y. Ohno, N. Futatsugi, A. Suenaga, N. Takada, and A. Konagaya (2003): Protein Explorer: A Petaflops Special-Purpose Computer System for Molecular Dynamics Simulations, *Proc. 2003 ACM/IEEE Supercomputing Conference (SC2003)*, Phoenix, USA.

[72] V. Akcelik, J. Bielak, G. Biros, I. Epanomeritakis, A. Fernandez, O. Ghattas, E. J. Kim, J. Lopez, D. O'Hallaron, T. Tu, and J. Urbanic (2003): High Resolution Forward and Inverse Earthquake Modeling on Terascale Computers, *Proc. 2003 ACM/IEEE Supercomputing Conference (SC2003)*, Phoenix, USA.

[73] C. Gribble, X. Cavin, M. Hartner, and C. Hansen (2003): Cluster-based Interactive Volume Rendering with Simian, *University of Utah School of Computing Technical Report UUCS-03-017.*

[74] OpenGL, http://www.opengl.org

[75] IEEE Task Force on Cluster Computing, http://www.ieeetfcc.org

[76] ClusterXY - IEEE Intl. Conference on Cluster Computing, http://www.cluster comp.org

[77] CCGridXY - IEEE Intl. Symposium on Cluster Computing and the Grid, http://www.ccgrid.org