Chapter 1

Cluster Computing at a Glance

MARK BAKER† AND RAJKUMAR BUYYA‡

†Division of Computer Science
University of Portsmouth
Southsea, Hants, UK

‡ School of Computer Science and Software Engineering
Monash University
Melbourne, Australia

Email: *Mark.Baker@port.ac.uk, rajkumar@dgs.monash.edu.au*

1.1 Introduction

Very often applications need more computing power than a sequential computer can provide. One way of overcoming this limitation is to improve the operating speed of processors and other components so that they can offer the power required by computationally intensive applications. Even though this is currently possible to certain extent, future improvements are constrained by the speed of light, thermodynamic laws, and the high financial costs for processor fabrication. A viable and cost-effective alternative solution is to connect multiple processors together and coordinate their computational efforts. The resulting systems are popularly known as parallel computers, and they allow the sharing of a computational task among multiple processors.

As Pfister [1] points out, there are three ways to improve performance:

- Work harder,
- Work smarter, and
- Get help.

In terms of computing technologies, the analogy to this mantra is that working harder is like using faster hardware (high performance processors or peripheral devices). Working smarter concerns doing things more efficiently and this revolves around the algorithms and techniques used to solve computational tasks. Finally, getting help refers to using multiple computers to solve a particular task.

1.1.1 Eras of Computing

The computing industry is one of the fastest growing industries and it is fueled by the rapid technological developments in the areas of computer hardware and software. The technological advances in hardware include chip development and fabrication technologies, fast and cheap microprocessors, as well as high bandwidth and low latency interconnection networks. Among them, the recent advances in VLSI (Very Large Scale Integration) technology has played a major role in the development of powerful sequential and parallel computers. Software technology is also developing fast. Mature software, such as OSs (Operating Systems), programming languages, development methodologies, and tools, are now available. This has enabled the development and deployment of applications catering to scientific, engineering, and commercial needs. It should also be noted that grand challenging applications, such as weather forecasting and earthquake analysis, have become the main driving force behind the development of powerful parallel computers.

One way to view computing is as two prominent developments/eras:

- Sequential Computing Era
- Parallel Computing Era

A review of the changes in computing eras is shown in Figure 1.1. Each computing era started with a development in hardware architectures, followed by system software (particularly in the area of compilers and operating systems), applications, and reaching its zenith with its growth in PSEs (Problem Solving Environments). Each component of a computing system undergoes three phases: R&D (Research and Development), commercialization, and commodity. The technology behind the development of computing system components in the sequential era has matured, and similar developments are yet to happen in the parallel era. That is, parallel computing technology needs to advance, as it is not mature enough to be exploited as commodity technology.

The main reason for creating and using parallel computers is that parallelism is one of the best ways to overcome the speed bottleneck of a single processor. In addition, the price performance ratio of a small cluster-based parallel computer as opposed to a minicomputer is much smaller and consequently a better value. In short, developing and producing systems of moderate speed using parallel architectures is much cheaper than the equivalent performance of a sequential system.

The remaining parts of this chapter focus on architecture alternatives for constructing parallel computers, motivations for transition to low cost parallel computing, a generic model of a cluster computer, commodity components used in building clusters, cluster middleware, resource management and scheduling, programming environments and tools, and representative cluster systems. The chapter ends with a summary of hardware and software trends, and concludes with future cluster technologies.

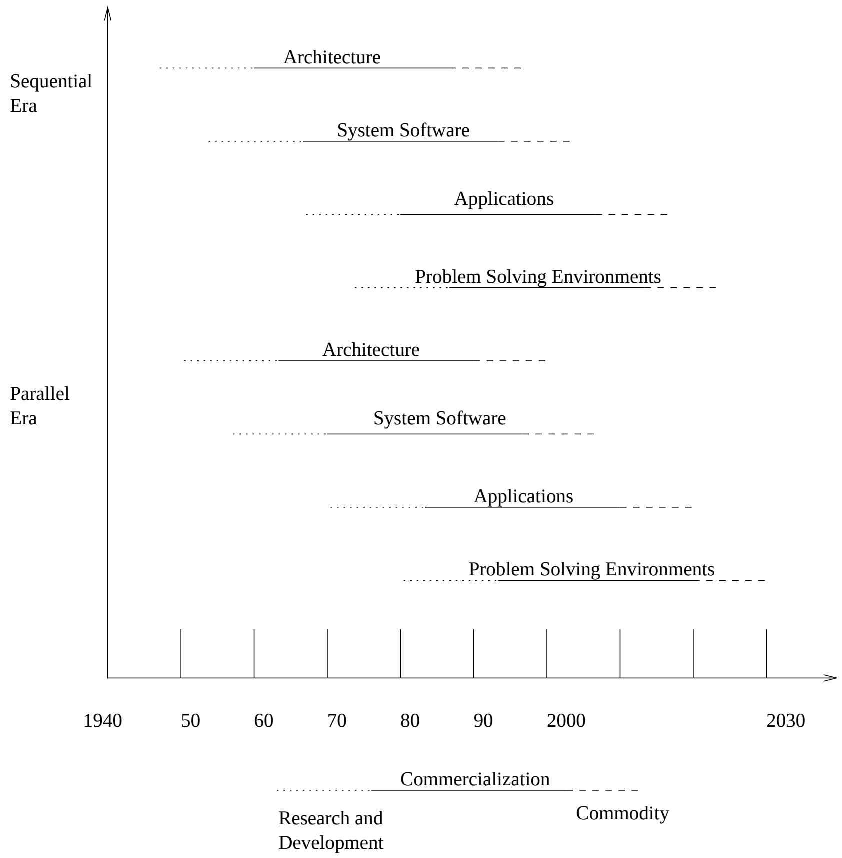

Figure 1.1 Two eras of computing.

1.2 Scalable Parallel Computer Architectures

During the past decade many different computer systems supporting high performance computing have emerged. Their taxonomy is based on how their processors, memory, and interconnect are laid out. The most common systems are:

- Massively Parallel Processors (MPP)
- Symmetric Multiprocessors (SMP)
- Cache-Coherent Nonuniform Memory Access (CC-NUMA)
- Distributed Systems
- Clusters

Table 1.1 shows a modified version comparing the architectural and functional characteristics of these machines originally given in [2] by Hwang and Xu.

An MPP is usually a large parallel processing system with a shared-nothing architecture. It typically consists of several hundred processing elements (nodes), which are interconnected through a high-speed interconnection network/switch. Each node can have a variety of hardware components, but generally consists of a main memory and one or more processors. Special nodes can, in addition, have peripherals such as disks or a backup system connected. Each node runs a separate copy of the operating system.

Table 1.1 Key Characteristics of Scalable Parallel Computers

Characteristic	**MPP**	**SMP CC-NUMA**	**Cluster**	**Distributed**
Number of Nodes	O(100)-O(1000)	O(10)-O(100)	O(100) or less	O(10)-O(1000)
Node Complexity	Fine grain or medium	Medium or coarse grained	Medium grain	Wide Range
Internode communication	Message passing/ shared variables for distributed shared memory	Centralized and Distributed Shared Memory (DSM)	Message Passing	Shared files, RPC, Message Passing and IPC
Job Scheduling	Single run queue on host	Single run queue mostly	Multiple queue but coordinated	Independent queues
SSI Support	Partially	Always in SMP and some NUMA	Desired	No
Node OS copies and type	N micro-kernels monolithic or layered OSs	One monolithic SMP and many for NUMA	N OS platforms -homogeneous or micro-kernel	N OS platforms homogeneous
Address Space	Multiple - single for DSM	Single	Multiple or single	Multiple
Internode Security	Unnecessary	Unnecessary	Required if exposed	Required
Ownership	One organization	One organization	One or more organizations	Many organizations

SMP systems today have from 2 to 64 processors and can be considered to have shared-everything architecture. In these systems, all processors share all the global resources available (bus, memory, I/O system); a single copy of the operating system runs on these systems.

CC-NUMA is a scalable multiprocessor system having a cache-coherent nonuniform memory access architecture. Like an SMP, every processor in a CC-NUMA system has a global view of all of the memory. This type of system gets its name (NUMA) from the nonuniform times to access the nearest and most remote parts of memory.

Distributed systems can be considered conventional networks of independent computers. They have multiple system images, as each node runs its own operating system, and the individual machines in a distributed system could be, for example, combinations of MPPs, SMPs, clusters, and individual computers.

At a basic level a cluster [1] is a collection of workstations or PCs that are interconnected via some network technology. For parallel computing purposes, a cluster will generally consist of high performance workstations or PCs interconnected by a high-speed network. A cluster works as an integrated collection of resources and can have a single system image spanning all its nodes. Refer to [1] and [2] for a detailed discussion on architectural and functional characteristics of the competing computer architectures.

1.3 Towards Low Cost Parallel Computing and Motivations

In the 1980s it was believed that computer performance was best improved by creating faster and more efficient processors. This idea was challenged by parallel processing, which in essence means linking together two or more computers to jointly solve some computational problem. Since the early 1990s there has been an increasing trend to move away from expensive and specialized proprietary parallel supercomputers towards networks of workstations. Among the driving forces that have enabled this transition has been the rapid improvement in the availability of commodity high performance components for workstations and networks. These technologies are making networks of computers (PCs or workstations) an appealing vehicle for parallel processing, and this is consequently leading to low-cost *commodity supercomputing*.

The use of parallel processing as a means of providing high performance computational facilities for large-scale and grand-challenge applications has been investigated widely. Until recently, however, the benefits of this research were confined to the individuals who had access to such systems. The trend in parallel computing is to move away from specialized traditional supercomputing platforms, such as the Cray/SGI T3E, to cheaper, 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 which is suitable for a large class of applications and workloads.

The use of clusters to prototype, debug, and run parallel applications is becoming an increasingly popular alternative to using specialized, typically expensive, parallel computing platforms. An important factor that has made the usage of clusters a practical proposition is the standardization of many of the tools and utilities used by parallel applications. Examples of these standards are the message passing library MPI [8] and data-parallel language HPF [3]. In this context, standardization enables

[1] Clusters, Network of Workstations (NOW), Cluster of Workstations (COW), and Workstation Clusters are synonymous.

applications to be developed, tested, and even run on NOW, and then at a later stage to be ported, with little modification, onto dedicated parallel platforms where CPU-time is accounted and charged.

The following list highlights some of the reasons NOW is preferred over specialized parallel computers [5], [4]:

- Individual workstations are becoming increasingly powerful. That is, workstation performance has increased dramatically in the last few years and is doubling every 18 to 24 months. This is likely to continue for several years, with faster processors and more efficient multiprocessor machines coming into the market.

- The communications bandwidth between workstations is increasing and latency is decreasing as new networking technologies and protocols are implemented in a LAN.

- Workstation clusters are easier to integrate into existing networks than special parallel computers.

- Typical low user utilization of personal workstations.

- The development tools for workstations are more mature compared to the contrasting proprietary solutions for parallel computers, mainly due to the nonstandard nature of many parallel systems.

- Workstation clusters are a cheap and readily available alternative to specialized high performance computing platforms.

- Clusters can be easily grown; node's capability can be easily increased by adding memory or additional processors.

Clearly, the workstation environment is better suited to applications that are not communication-intensive since a LAN typically has high message start-up latencies and low bandwidths. If an application requires higher communication performance, the existing commonly deployed LAN architectures, such as Ethernet, are not capable of providing it.

Traditionally, in science and industry, a workstation referred to a UNIX platform and the dominant function of PC-based machines was for administrative work and word processing. There has been, however, a rapid convergence in processor performance and kernel-level functionality of UNIX workstations and PC-based machines in the last three years (this can be attributed to the introduction of high performance Pentium-based machines and the Linux and Windows NT operating systems). This convergence has led to an increased level of interest in utilizing PC-based systems as a cost-effective computational resource for parallel computing. This factor coupled with the comparatively low cost of PCs and their widespread availability in both academia and industry has helped initiate a number of software projects whose primary aim is to harness these resources in some collaborative way.

1.4 Windows of Opportunity

The resources available in the average NOW, such as processors, network interfaces, memory and hard disk, offer a number of research opportunities, such as:

Parallel Processing - Use the multiple processors to build MPP/DSM-like systems for parallel computing.

Network RAM - Use the memory associated with each workstation as aggregate DRAM cache; this can dramatically improve virtual memory and file system performance.

Software RAID (Redundant Array of Inexpensive Disks) - Use the arrays of workstation disks to provide cheap, highly available, and scalable file storage by using redundant arrays of workstation disks with LAN as I/O backplane. In addition, it is possible to provide parallel I/O support to applications through middleware such as MPI-IO.

Multipath Communication - Use the multiple networks for parallel data transfer between nodes.

Scalable parallel applications require good floating-point performance, low latency and high bandwidth communications, scalable network bandwidth, and fast access to files. Cluster software can meet these requirements by using resources associated with clusters. A file system supporting parallel I/O can be built using disks associated with each workstation instead of using expensive hardware-RAID. Virtual memory performance can be drastically improved by using Network RAM as a backing store instead of hard disk. In a way, parallel file systems and Network RAM reduces the widening performance gap between processors and disks.

It is very common to connect cluster nodes using the standard Ethernet and specialized high performance networks such as Myrinet. These multiple networks can be utilized for transferring data simultaneously across cluster nodes. The multipath communication software performs demultiplexing of data at the transmitting end across multiple networks and multiplexing of data at the receiving end. Thus, all available networks can be utilized for faster communication of data between cluster nodes.

1.5 A Cluster Computer and its Architecture

A cluster is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers working together as a single, integrated computing resource.

A computer node can be a single or multiprocessor system (PCs, workstations, or SMPs) with memory, I/O facilities, and an operating system. A cluster generally refers to two or more computers (nodes) connected together. The nodes can exist in a single cabinet or be physically separated and connected via a LAN. An interconnected (LAN-based) cluster of computers can appear as a single system to users and applications. Such a system can provide a cost-effective way to gain features and benefits (fast and reliable services) that have historically been found only on more expensive proprietary shared memory systems. The typical architecture of a cluster is shown in Figure 1.2.

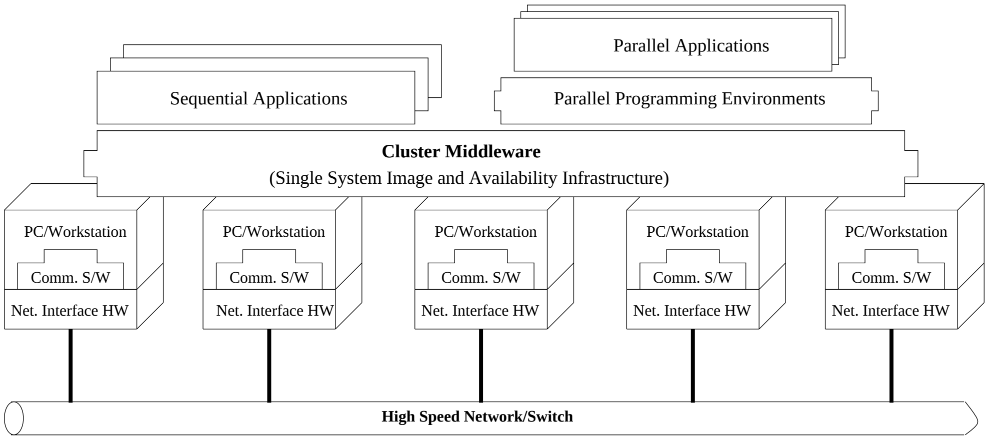

Figure 1.2 Cluster computer architecture.

The following are some prominent components of cluster computers:

- Multiple High Performance Computers (PCs, Workstations, or SMPs)
- State-of-the-art Operating Systems (Layered or Micro-kernel based)
- High Performance Networks/Switches (such as Gigabit Ethernet and Myrinet)
- Network Interface Cards (NICs)
- Fast Communication Protocols and Services (such as Active and Fast Messages)
- Cluster Middleware (Single System Image (SSI) and System Availability Infrastructure)
  - Hardware (such as Digital (DEC) Memory Channel, hardware DSM, and SMP techniques)
  - Operating System Kernel or Gluing Layer (such as Solaris MC and GLUnix)
  - Applications and Subsystems
    - Applications (such as system management tools and electronic forms)
    - Runtime Systems (such as software DSM and parallel file system)

    * Resource Management and Scheduling software (such as LSF (Load Sharing Facility) and CODINE (COmputing in DIstributed Networked Environments))

- Parallel Programming Environments and Tools (such as compilers, PVM (Parallel Virtual Machine), and MPI (Message Passing Interface))
- Applications
  - Sequential
  - Parallel or Distributed

The network interface hardware acts as a communication processor and is responsible for transmitting and receiving packets of data between cluster nodes via a network/switch. (Refer to Chapter 9 for further details on cluster interconnects and network interfaces.)

Communication software offers a means of fast and reliable data communication among cluster nodes and to the outside world. Often, clusters with a special network/switch like Myrinet use communication protocols such as active messages for fast communication among its nodes. They potentially bypass the operating system and thus remove the critical communication overheads providing direct user-level access to the network interface.

The cluster nodes can work collectively, as an integrated computing resource, or they can operate as individual computers. The cluster middleware is responsible for offering an illusion of a unified system image (single system image) and availability out of a collection on independent but interconnected computers.

Programming environments can offer portable, efficient, and easy-to-use tools for development of applications. They include message passing libraries, debuggers, and profilers. It should not be forgotten that clusters could be used for the execution of sequential or parallel applications.

1.6 Clusters Classifications

Clusters offer the following features at a relatively low cost:

- High Performance
- Expandability and Scalability
- High Throughput
- High Availability

Cluster technology permits organizations to boost their processing power using standard technology (commodity hardware and software components) that can be acquired/purchased at a relatively low cost. This provides expandability–an affordable upgrade path that lets organizations increase their computing power–while preserving their existing investment and without incurring a lot of extra expenses.

The performance of applications also improves with the support of scalable software environment. Another benefit of clustering is a failover capability that allows a backup computer to take over the tasks of a failed computer located in its cluster.

Clusters are classified into many categories based on various factors as indicated below.

1. **Application Target -** Computational science or mission-critical applications.

    - High Performance (HP) Clusters
    - High Availability (HA) Clusters

    The main concentration of this book is on HP clusters and the technologies and environments required for using them in parallel computing. However, we also discuss issues involved in building HA clusters with an aim for integrating performance and availability into a single system (see Chapter 4).

2. **Node Ownership -** Owned by an individual or dedicated as a cluster node.

    - Dedicated Clusters
    - Nondedicated Clusters

    The distinction between these two cases is based on the ownership of the nodes in a cluster. In the case of dedicated clusters, a particular individual does not own a workstation; the resources are shared so that parallel computing can be performed across the entire cluster [6]. The alternative nondedicated case is where individuals own workstations and applications are executed by stealing idle CPU cycles [7]. The motivation for this scenario is based on the fact that most workstation CPU cycles are unused, even during peak hours. Parallel computing on a dynamically changing set of nondedicated workstations is called adaptive parallel computing.

    In nondedicated clusters, a tension exists between the workstation owners and remote users who need the workstations to run their application. The former expects fast interactive response from their workstation, while the latter is only concerned with fast application turnaround by utilizing any spare CPU cycles. This emphasis on sharing the processing resources erodes the concept of node ownership and introduces the need for complexities such as process migration and load balancing strategies. Such strategies allow clusters to deliver adequate interactive performance as well as to provide shared resources to demanding sequential and parallel applications.

3. **Node Hardware -** PC, Workstation, or SMP.

    - Clusters of PCs (CoPs) or Piles of PCs (PoPs)
    - Clusters of Workstations (COWs)

- Clusters of SMPs (CLUMPs)

4. **Node Operating System -** Linux, NT, Solaris, AIX, etc.
    - Linux Clusters (e.g., Beowulf)
    - Solaris Clusters (e.g., Berkeley NOW)
    - NT Clusters (e.g., HPVM)
    - AIX Clusters (e.g., IBM SP2)
    - Digital VMS Clusters
    - HP-UX clusters.
    - Microsoft Wolfpack clusters.

5. **Node Configuration -** Node architecture and type of OS it is loaded with.
    - Homogeneous Clusters: All nodes will have similar architectures and run the same OSs.
    - Heterogeneous Clusters: All nodes will have different architectures and run different OSs.

6. **Levels of Clustering -** Based on location of nodes and their count.
    - Group Clusters (#nodes: 2-99): Nodes are connected by SANs (System Area Networks) like Myrinet and they are either stacked into a frame or exist within a center.
    - Departmental Clusters (#nodes: 10s to 100s)
    - Organizational Clusters (#nodes: many 100s)
    - National Metacomputers (WAN/Internet-based): (#nodes: many departmental/organizational systems or clusters)
    - International Metacomputers (Internet-based): (#nodes: 1000s to many millions)

Individual clusters may be interconnected to form a larger system (clusters of clusters) and, in fact, the Internet itself can be used as a computing cluster. The use of wide-area networks of computer resources for high performance computing has led to the emergence of a new field called Metacomputing. (Refer to Chapter 7 for further details on Metacomputing.)

1.7 Commodity Components for Clusters

The improvements in workstation and network performance, as well as the availability of standardized programming APIs, are paving the way for the widespread usage of cluster-based parallel systems. In this section, we discuss some of the hardware and software components commonly used to build clusters and nodes. The trends in hardware and software technologies are discussed in later parts of this chapter.

1.7.1 Processors

Over the past two decades, phenomenal progress has taken place in microprocessor architecture (for example RISC, CISC, VLIW, and Vector) and this is making the single-chip CPUs almost as powerful as processors used in supercomputers. Most recently researchers have been trying to integrate processor and memory or network interface into a single chip. The Berkeley Intelligent RAM (IRAM) project [9] is exploring the entire spectrum of issues involved in designing general purpose computer systems that integrate a processor and DRAM onto a single chip – from circuits, VLSI design, and architectures to compilers and operating systems. Digital, with its Alpha 21364 processor, is trying to integrate processing, memory controller, and network interface into a single chip.

Intel processors are most commonly used in PC-based computers. The current generation Intel x86 processor family includes the Pentium Pro and II. These processors, while not in the high range of performance, match the performance of medium level workstation processors [10]. In the high performance range, the Pentium Pro shows a very strong integer performance, beating Sun's UltraSPARC at the same clock speed; however, the floating-point performance is much lower. The Pentium II Xeon, like the newer Pentium IIs, uses a 100 MHz memory bus. It is available with a choice of 512KB to 2MB of L2 cache, and the cache is clocked at the same speed as the CPU, overcoming the L2 cache size and performance issues of the plain Pentium II. The accompanying 450NX chipset for the Xeon supports 64-bit PCI busses that can support Gigabit interconnects.

Other popular processors include x86 variants (AMD x86, Cyrix x86), Digital Alpha, IBM PowerPC, Sun SPARC, SGI MIPS, and HP PA. Computer systems based on these processors have also been used as clusters; for example, Berkeley NOW uses Sun's SPARC family of processors in their cluster nodes. (For further information on industrial high performance microprocessors refer to web-based VLSI Microprocessors Guide [11].)

1.7.2 Memory and Cache

Originally, the memory present within a PC was 640 KBytes, usually 'hardwired' onto the motherboard. Typically, a PC today is delivered with between 32 and 64 MBytes installed in slots with each slot holding a Standard Industry Memory Module (SIMM); the potential capacity of a PC is now many hundreds of MBytes.

Computer systems can use various types of memory and they include Extended Data Out (EDO) and fast page. EDO allows the next access to begin while the previous data is still being read, and fast page allows multiple adjacent accesses to be made more efficiently.

The amount of memory needed for the cluster is likely to be determined by the cluster target applications. Programs that are parallelized should be distributed such that the memory, as well as the processing, is distributed between processors for scalability. Thus, it is not necessary to have a RAM that can hold the entire problem in memory on each system, but it should be enough to avoid the occurrence of too much swapping of memory blocks (page-misses) to disk, since disk access has a large impact on performance.

Access to DRAM is extremely slow compared to the speed of the processor, taking up to orders of magnitude more time than a CPU clock cycle. Caches are used to keep recently used blocks of memory for very fast access if the CPU references a word from that block again. However, the very fast memory used for cache is expensive and cache control circuitry becomes more complex as the size of the cache grows. Because of these limitations, the total size of a cache is usually in the range of 8KB to 2MB.

Within Pentium-based machines it is not uncommon to have a 64-bit wide memory bus as well as a chip set that supports 2 MBytes of external cache. These improvements were necessary to exploit the full power of the Pentium and to make the memory architecture very similar to that of UNIX workstations.

1.7.3 Disk and I/O

Improvements in disk access time have not kept pace with microprocessor performance, which has been improving by 50 percent or more per year. Although magnetic media densities have increased, reducing disk transfer times by approximately 60 to 80 percent per year, overall improvement in disk access times, which rely upon advances in mechanical systems, has been less than 10 percent per year.

Grand challenge applications often need to process large amounts of data and data sets. Amdahl's law implies that the speed-up obtained from faster processors is limited by the slowest system component; therefore, it is necessary to improve I/O performance such that it balances with CPU performance. One way of improving I/O performance is to carry out I/O operations in parallel, which is supported by parallel file systems based on hardware or software RAID. Since hardware RAIDs can be expensive, software RAIDs can be constructed by using disks associated with each workstation in the cluster.

1.7.4 System Bus

The initial PC bus (AT, or now known as ISA bus) used was clocked at 5 MHz and was 8 bits wide. When first introduced, its abilities were well matched to the rest of the system. PCs are modular systems and until fairly recently only the processor and memory were located on the motherboard, other components were typically found on daughter cards connected via a system bus. The performance of PCs has increased by orders of magnitude since the ISA bus was first used, and it has consequently become a bottleneck, which has limited the machine throughput. The ISA bus was extended to be 16 bits wide and was clocked in excess of 13 MHz. This, however, is still not sufficient to meet the demands of the latest CPUs, disk interfaces, and other peripherals.

A group of PC manufacturers introduced the VESA local bus, a 32-bit bus that matched the system's clock speed. The VESA bus has largely been superseded by the Intel-created PCI bus, which allows 133 Mbytes/s transfers and is used inside

Pentium-based PCs. PCI has also been adopted for use in non-Intel based platforms such as the Digital AlphaServer range. This has further blurred the distinction between PCs and workstations, as the I/O subsystem of a workstation may be built from commodity interface and interconnect cards.

1.7.5 Cluster Interconnects

The nodes in a cluster communicate over high-speed networks using a standard networking protocol such as TCP/IP or a low-level protocol such as Active Messages. In most facilities it is likely that the interconnection will be via standard Ethernet. In terms of performance (latency and bandwidth), this technology is showing its age. However, Ethernet is a cheap and easy way to provide file and printer sharing. A single Ethernet connection cannot be used seriously as the basis for cluster-based computing; its bandwidth and latency are not balanced compared to the computational power of the workstations now available. Typically, one would expect the cluster interconnect bandwidth to exceed 10 MBytes/s and have message latencies of less than 100 μs. A number of high performance network technologies are available in the marketplace; in this section we discuss a few of them.

Ethernet, Fast Ethernet, and Gigabit Ethernet

Standard Ethernet has become almost synonymous with workstation networking. This technology is in widespread usage, both in the academic and commercial sectors. However, its 10 Mbps bandwidth is no longer sufficient for use in environments where users are transferring large data quantities or there are high traffic densities. An improved version, commonly known as Fast Ethernet, provides 100 Mbps bandwidth and has been designed to provide an upgrade path for existing Ethernet installations. Standard and Fast Ethernet cannot coexist on a particular cable, but each uses the same cable type. When an installation is hub-based and uses twisted-pair it is possible to upgrade the hub to one, which supports both standards, and replace the Ethernet cards in only those machines where it is believed to be necessary.

Now, the state-of-the-art Ethernet is the Gigabit Ethernet[2] and its attraction is largely due to two key characteristics. First, it preserves Ethernet's simplicity while enabling a smooth migration to Gigabit-per-second (Gbps) speeds. Second, it delivers a very high bandwidth to aggregate multiple Fast Ethernet segments and to support high-speed server connections, switched intrabuilding backbones, interswitch links, and high-speed workgroup networks.

Asynchronous Transfer Mode (ATM)

ATM is a switched virtual-circuit technology and was originally developed for the telecommunications industry [12]. It is embodied within a set of protocols and standards defined by the International Telecommunications Union. The international

[2] Gigabit Ethernet is Ethernet, only faster!

ATM Forum, a non-profit organization, continues this work. Unlike some other networking technologies, ATM is intended to be used for both LAN and WAN, presenting a unified approach to both. ATM is based around small fixed-size data packets termed cells. It is designed to allow cells to be transferred using a number of different media such as both copper wire and fiber optic cables. This hardware variety also results in a number of different interconnect performance levels.

When first introduced, ATM used optical fiber as the link technology. However, this is undesirable in desktop environments; for example, twisted pair cables may have been used to interconnect a networked environment and moving to fiber-based ATM would mean an expensive upgrade. The two most common cabling technologies found in a desktop environment are telephone style cables (CAT-3) and a better quality cable (CAT-5). CAT-5 can be used with ATM allowing upgrades of existing networks without replacing cabling.

Scalable Coherent Interface (SCI)

SCI is an IEEE 1596-1992 standard aimed at providing a low-latency distributed shared memory across a cluster [13]. SCI is the modern equivalent of a Processor-Memory-I/O bus and LAN combined. It is designed to support distributed multiprocessing with high bandwidth and low latency. It provides a scalable architecture that allows large systems to be built out of many inexpensive mass-produced components.

SCI is a point-to-point architecture with directory-based cache coherence. It can reduce the delay of interprocessor communications even when compared to the newest and best technologies currently available, such as Fiber Channel and ATM. SCI achieves this by eliminating the need for runtime layers of software protocol-paradigm translation. A remote communication in SCI takes place as just part of a simple load or store process in a processor. Typically, a remote address results in a cache miss. This in turn causes the cache controller to address remote memory via SCI to get the data. The data is fetched to the cache with a delay in the order of a few μss and then the processor continues execution.

Dolphin currently produces SCI cards for SPARC's SBus; however, they have also announced availability of PCI-based SCI cards. They have produced an SCI MPI which offers less than 12 μs zero message-length latency on the Sun SPARC platform and they intend to provide MPI for Windows NT. A SCI version of High Performance Fortran (HPF) is available from Portland Group Inc.

Although SCI is favored in terms of fast distributed shared memory support, it has not been taken up widely because its scalability is constrained by the current generation of switches and its components are relatively expensive.

Myrinet

Myrinet is a 1.28 Gbps full duplex interconnection network supplied by Myricom [15]. It is a proprietary, high performance interconnect. Myrinet uses low latency cut-through routing switches, which is able to offer fault tolerance by automatic mapping of the network configuration. This also simplifies setting up the network. Myrinet supports both Linux and NT. In addition to TCP/IP support, the MPICH implementation of MPI is also available on a number of custom-developed packages such as Berkeley active messages, which provide sub-10 μs latencies.

Myrinet is relatively expensive when compared to Fast Ethernet, but has real advantages over it: very low-latency (5 μs, one-way point-to-point), very high throughput, and a programmable on-board processor allowing for greater flexibility. It can saturate the effective bandwidth of a PCI bus at almost 120 Mbytes/s with 4Kbytes packets.

One of the main disadvantages of Myrinet is, as mentioned, its price compared to Fast Ethernet. The cost of Myrinet-LAN components, including the cables and switches, is in the range of $1,500 per host. Also, switches with more than 16 ports are unavailable, so scaling can be complicated, although switch chaining is used to construct larger Myrinet clusters.

1.7.6 Operating Systems

A modern operating system provides two fundamental services for users. First, it makes the computer hardware easier to use. It creates a virtual machine that differs markedly from the real machine. Indeed, the computer revolution of the last two decades is due, in part, to the success that operating systems have achieved in shielding users from the obscurities of computer hardware. Second, an operating system shares hardware resources among users. One of the most important resources is the processor. A multitasking operating system, such as UNIX or Windows NT, divides the work that needs to be executed among processes, giving each process memory, system resources, at least one thread of execution, and an executable unit within a process. The operating system runs one thread for a short time and then switches to another, running each thread in turn. Even on a single-user system, multitasking is extremely helpful because it enables the computer to perform multiple tasks at once. For example, a user can edit a document while another document is printing in the background or while a compiler compiles a large program. Each process gets its work done, and to the user all the programs appear to run simultaneously.

Apart from the benefits mentioned above, the new concept in operating system services is supporting multiple threads of control in a process itself. This concept has added a new dimension to parallel processing, the parallelism within a process, instead of across the programs. In the next-generation operating system kernels, address space and threads are decoupled so that a single address space can have multiple execution threads. Programming a process having multiple threads of control is known as multithreading. POSIX threads interface is a standard programming environment for creating concurrency/parallelism within a process.

A number of trends affecting operating system design have been witnessed over the past few years, foremost of these is the move towards modularity. Operating systems such as Microsoft's Windows, IBM's OS/2, and others, are splintered into discrete components, each having a small, well defined interface, and each communicating with others via an intertask messaging interface. The lowest level is the micro-kernel, which provides only essential OS services, such as context switching. Windows NT, for example, also includes a hardware abstraction layer (HAL) beneath its micro-kernel, which enables the rest of the OS to perform irrespective of the underlying processor. This high level abstraction of OS portability is a driving force behind the modular, micro-kernel-based push. Other services are offered by subsystems built on top of the micro-kernel. For example, file services can be offered by the file-server, which is built as a subsystem on top of the microkernel. (Refer to Chapter 29 for details on a micro-kernel based cluster operating system offering single system image.)

This section focuses on the various operating systems available for workstations and PCs. Operating system technology is maturing and can easily be extended and new subsystems can be added without modifying the underlying OS structure. Modern operating systems support multithreading at the kernel level and high performance user level multithreading systems can be built without their kernel intervention. Most PC operating systems have become stable and support multitasking, multithreading, and networking.

UNIX and its variants (such as Sun Solaris and IBM's AIX, HP UX) are popularly used on workstations. In this section, we discuss three popular operating systems that are used on nodes of clusters of PCs or Workstations.

LINUX

Linux [16] is a UNIX-like OS which was initially developed by Linus Torvalds, a Finnish undergraduate student in 1991-92. The original releases of Linux relied heavily on the Minix OS; however, the efforts of a number of collaborating programmers have resulted in the development and implementation of a robust and reliable, POSIX compliant, OS.

Although Linux was developed by a single author initially, a large number of authors are now involved in its development. One major advantage of this distributed development has been that there is a wide range of software tools, libraries, and utilities available. This is due to the fact that any capable programmer has access to the OS source and can implement the feature that they wish. Linux quality control is maintained by only allowing kernel releases from a single point, and its availability via the Internet helps in getting fast feedback about bugs and other problems. The following are some advantages of using Linux:

- Linux runs on cheap x86 platforms, yet offers the power and flexibility of UNIX.
- Linux is readily available on the Internet and can be downloaded without cost.
- It is easy to fix bugs and improve system performance.

- Users can develop or fine-tune hardware drivers which can easily be made available to other users.

Linux provides the features typically found in UNIX implementations such as: preemptive multitasking, demand-paged virtual memory, multiuser, and multiprocessor support [17]. Most applications written for UNIX will require little more than a recompilation. In addition to the Linux kernel, a large amount of application/systems software is also freely available, including GNU software and XFree86, a public domain X-server.

Solaris

The Solaris operating system from SunSoft is a UNIX-based multithreaded and multiuser operating system. It supports Intel x86 and SPARC-based platforms. Its networking support includes a TCP/IP protocol stack and layered features such as Remote Procedure Calls (RPC), and the Network File System (NFS). The Solaris programming environment includes ANSI-compliant C and C++ compilers, as well as tools to profile and debug multithreaded programs.

The Solaris kernel supports multithreading, multiprocessing, and has real-time scheduling features that are critical for multimedia applications. Solaris supports two kinds of threads: Light Weight Processes (LWPs) and user level threads. The threads are intended to be sufficiently lightweight so that there can be thousands present and that synchronization and context switching can be accomplished rapidly without entering the kernel.

Solaris, in addition to the BSD file system, also supports several types of non-BSD file systems to increase performance and ease of use. For performance there are three new file system types: CacheFS, AutoClient, and TmpFS. The CacheFS caching file system allows a local disk to be used as an operating system managed cache of either remote NFS disk or CD-ROM file systems. With AutoClient and CacheFS, an entire local disk can be used as cache. The TmpFS temporary file system uses main memory to contain a file system. In addition, there are other file systems like the *Proc* file system and *Volume* file system to improve system usability.

Solaris supports distributed computing and is able to store and retrieve distributed information to describe the system and users through the Network Information Service (NIS) and database. The Solaris GUI, OpenWindows, is a combination of X11R5 and the Adobe Postscript system, which allows applications to be run on remote systems with the display shown along with local applications.

Microsoft Windows NT

Microsoft Windows NT (New Technology) is a dominant operating system in the personal computing marketplace [18]. It is a preemptive, multitasking, multiuser, 32-bit operating system. NT supports multiple CPUs and provides multi-tasking, using symmetrical multiprocessing. Each 32-bit NT-application operates in its own virtual memory address space. Unlike earlier versions (such as Windows for Workgroups and Windows 95/98), NT is a complete operating system, and not an addition to DOS. NT supports different CPUs and multiprocessor machines with threads. NT has an object-based security model and its own special file system (NTFS) that allows permissions to be set on a file and directory basis.

A schematic diagram of the NT architecture is shown in Figure 1.3. NT has the network protocols and services integrated with the base operating system.

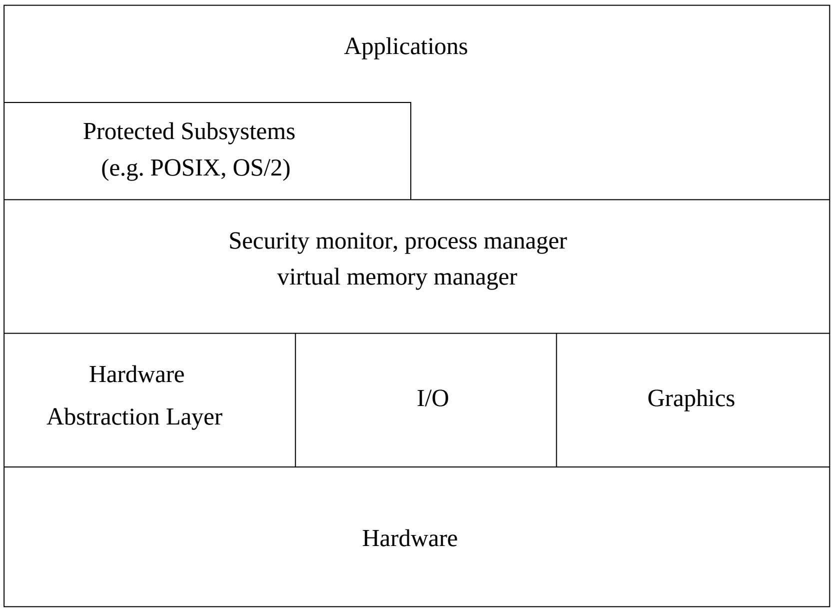

Figure 1.3 Windows NT 4.0 architecture.

Packaged with Windows NT are several built-in networking protocols, such as IPX/SPX, TCP/IP, and NetBEUI and APIs, such as NetBIOS, DCE RPC, and Windows Sockets (WinSock). TCP/IP applications use WinSock to communicate over a TCP/IP network.

1.8 Network Services/Communication SW

The communication needs of distributed applications are diverse and varied and range from reliable point-to-point to unreliable multicast communications. The communications infrastructure needs to support protocols that are used for bulk-data transport, streaming data, group communications, and those used by distributed objects.

The communication services employed provide the basic mechanisms needed by a cluster to transport administrative and user data. These services will also provide the cluster with important quality of service parameters, such as latency, bandwidth, reliability, fault-tolerance, and jitter control. Typically, the network services are designed as a hierarchical stack of protocols. In such a layered system each protocol layer in the stack exploits the services provided by the protocols below it in the stack. The classic example of such a network architecture is the ISO OSI 7-layer system.

Traditionally, the operating system services (pipes/sockets) have been used for communication between processes in message passing systems. As a result, communication between source and destination involves expensive operations, such as the passing of messages between many layers, data copying, protection checking, and reliable communication measures. Often, clusters with a special network/switch like Myrinet use lightweight communication protocols such as active messages for fast communication among its nodes. They potentially bypass the operating system and thus remove the critical communication overheads and provide direct, user-level access to the network interface.

Often in clusters, the network services will be built from a relatively low-level communication API (Application Programming Interface) that can be used to support a wide range of high-level communication libraries and protocols. These mechanisms provide the means to implement a wide range of communications methodologies, including RPC, DSM, and stream-based and message passing interfaces such as MPI and PVM. (A further discussion of communications and network protocols can be found in Chapter 10.)

1.9 Cluster Middleware and Single System Image

If a collection of interconnected computers is designed to appear as a unified resource, we say it possesses a Single System Image (SSI). The SSI is supported by a middleware layer that resides between the operating system and user-level environment. This middleware consists of essentially two sublayers of software infrastructure [19]:

- Single System Image infrastructure.
- System Availability infrastructure.

The SSI infrastructure glues together operating systems on all nodes to offer unified access to system resources. The system availability infrastructure enables the cluster services of checkpointing, automatic failover, recovery from failure, and fault-tolerant support among all nodes of the cluster.

The following are the advantages/benefits of a cluster middleware and SSI, in particular:

- It frees the end user from having to know where an application will run.
- It frees the operator from having to know where a resource (an instance of resource) is located.

- It does not restrict the operator or system programmer who needs to work on a particular region; the end user interface (hyperlink - makes it easy to inspect consolidated data in more detail) can navigate to the region where a problem has arisen.
- It reduces the risk of operator errors, with the result that end users see improved reliability and higher availability of the system.
- It allows to centralize/decentralize system management and control to avoid the need of skilled administrators for system administration.
- It greatly simplifies system management; actions affecting multiple resources can be achieved with a single command, even where the resources are spread among multiple systems on different machines.
- It provides location-independent message communication. Because SSI provides a dynamic map of the message routing as it occurs in reality, the operator can always be sure that actions will be performed on the current system.
- It helps track the locations of all resources so that there is no longer any need for system operators to be concerned with their physical location while carrying out system management tasks.

The benefits of a SSI also apply to system programmers. It reduces the time, effort and knowledge required to perform tasks, and allows current staff to handle larger or more complex systems.

1.9.1 Single System Image Levels/Layers

The SSI concept can be applied to applications, specific subsystems, or the entire server cluster. Single system image and system availability services can be offered by one or more of the following levels/layers:

- Hardware (such as Digital (DEC) Memory Channel, hardware DSM, and SMP techniques)
- Operating System Kernel—Underware[3] or Gluing Layer (such as Solaris MC and GLUnix)
- Applications and Subsystems—Middleware
  - Applications (such as system management tools and electronic forms)
  - Runtime Systems (such as software DSM and parallel file system)
  - Resource Management and Scheduling software (such as LSF and CODINE)

[3] It refers to the infrastructure hidden below the user/kernel interface.

It should also be noted that programming and runtime systems like PVM can also serve as cluster middleware.

The SSI layers support both cluster-aware (such as parallel applications developed using MPI) and non-aware applications (typically sequential programs). These applications (cluster-aware, in particular) demand operational transparency and scalable performance (i.e., when cluster capability is enhanced, they need to run faster). Clusters, at one operational extreme, act like an SMP or MPP system with a high degree of SSI, and at another they can function as a distributed system with multiple system images.

The SSI and system availability services play a major role in the success of clusters. In the following section, we briefly discuss the layers supporting this infrastructure. A detailed discussion on cluster infrastructure can be found in the rest of the chapter with suitable pointers for further information.

Hardware Layer

Systems such as Digital (DEC's) Memory Channel and hardware DSM offer SSI at hardware level and allow the user to view cluster as a shared memory system. Digital's memory channel, a dedicated cluster interconnect, provides virtual shared memory among nodes by means of internodal address space mapping. (Refer to Chapter 9 for further discussion on DEC memory channel.)

Operating System Kernel (Underware) or Gluing Layer

Cluster operating systems support an efficient execution of parallel applications in an environment shared with sequential applications. A goal is to pool resources in a cluster to provide better performance for both sequential and parallel applications. To realize this goal, the operating system must support gang-scheduling of parallel programs, identify idle resources in the system (such as processors, memory, and networks), and offer globalized access to them. It has to support process migration for dynamic load balancing and fast interprocess communication for both the system and user-level applications. The OS must make sure these features are available to the user without the need of new system calls or commands and having the same syntax. OS kernels supporting SSI include SCO UnixWare and Sun Solaris-MC.

A full cluster-wide SSI allows all physical resources and kernel resources to be visible and accessible from all nodes within the system. Full SSI can be achieved as underware (SSI at OS level). In other words, each node's OS kernel cooperating to present the same view from all kernel interfaces on all nodes.

The full SSI at kernel level, can save time and money because existing programs and applications do not have to be rewritten to work in this new environment. In addition, these applications will run on any node without administrative setup, and processes can be migrated to load balance between the nodes and also to support fault-tolerance if necessary.

Most of the operating systems that support a SSI are built as a layer on top of the existing operating systems and perform global resource allocation. This strategy makes the system easily portable, tracks vendor software upgrades, and reduces development time. Berkeley GLUnix follows this philosophy and proves that new systems can be built quickly by mapping new services onto the functionality provided by the layer underneath.

Applications and Subsystems Layer (Middleware)

SSI can also be supported by applications and subsystems, which presents multiple, cooperating components of an application to the user/administrator as a single application. The application level SSI is the highest and in a sense most important, because this is what the end user sees. For instance, a cluster administration tool offers a single point of management and control SSI services. These can be built as GUI-based tools offering a single window for the monitoring and control of cluster as a whole, individual nodes, or specific system components.

The subsystems offer a software means for creating an easy-to-use and efficient cluster system. Run time systems, such as cluster file systems, make disks attached to cluster nodes appear as a single large storage system. SSI offered by file systems ensures that every node in the cluster has the same view of the data. Global job scheduling systems manage resources, and enables the scheduling of system activities and execution of applications while offering high availability services transparently.

1.9.2 SSI Boundaries

A key that provides structure to the SSI lies in noting the following points [1]:

- Every single system image has a boundary; and
- Single system image support can exist at different levels within a system–one able to be built on another.

For instance, a subsystem (resource management systems like LSF and CODINE) can make a collection of interconnected machines appear as one big machine. When any operation is performed within the SSI boundary of the subsystem, it provides an illusion of a classical supercomputer. But if anything is performed outside its SSI boundary, the cluster appears to be just a bunch of connected computers. Another subsystem/application can make the same set of machines appear as a large database/storage system. For instance, a cluster file system built using local disks associated with nodes can appear as a large storage system (software RAID)/parallel file system and offer faster access to the data.

1.9.3 Middleware Design Goals

The design goals of cluster-based systems are mainly focused on complete transparency in resource management, scalable performance, and system availability in supporting user applications.

Complete Transparency

The SSI layer must allow the user to use a cluster easily and effectively without the knowledge of the underlying system architecture. The operating environment appears familiar (by providing the same look and feel of the existing system) and is convenient to use. The user is provided with the view of a globalized file system, processes, and network. For example, in a cluster with a single entry point, the user can login at any node and the system administrator can install/load software at anyone's node and have be visible across the entire cluster. Note that on distributed systems, one needs to install the same software for each node. The details of resource management and control activities such as resource allocation, de-allocation, and replication are invisible to user processes. This allows the user to access system resources such as memory, processors, and the network transparently, irrespective of whether they are available locally or remotely.

Scalable Performance

As clusters can easily be expanded, their performance should scale as well. This scalability should happen without the need for new protocols and APIs. To extract the maximum performance, the SSI service must support load balancing and parallelism by distributing workload evenly among nodes. For instance, single point entry should distribute ftp/remote exec/login requests to lightly loaded nodes. The cluster must offer these services with small overhead and also ensure that the time required to execute the same operation on a cluster should not be larger than on a single workstation (assuming cluster nodes and workstations have similar configuration).

Enhanced Availability

The middleware services must be highly available at all times. At any time, a point of failure should be recoverable without affecting a user's application. This can be achieved by employing checkpointing and fault tolerant technologies (hot standby, mirroring, failover, and failback services) to enable rollback recovery.

When SSI services are offered using the resources available on multiple nodes, failure of any node should not affect the system's operation and a particular service should support one or more of the design goals. For instance, when a file system is distributed among many nodes with a certain degree of redundancy, when a node fails, that portion of file system could be migrated to another node transparently.

1.9.4 Key Services of SSI and Availability Infrastructure

Ideally, a cluster should offer a wide range of SSI and availability services. These services offered by one or more layers, stretch along different dimensions of an application domain. The following sections discuss SSI and availability services offered by middleware infrastructures.

SSI Support Services

Single Point of Entry: A user can connect to the cluster as a single system (like telnet beowulf.myinstitute.edu), instead of connecting to individual nodes as in the case of distributed systems (like telnet node1.beowulf.myinstitute.edu).

Single File Hierarchy (SFH): On entering into the system, the user sees a file system as a single hierarchy of files and directories under the same root directory. Examples: xFS and Solaris MC Proxy.

Single Point of Management and Control: The entire cluster can be monitored or controlled from a single window using a single GUI tool, much like an NT workstation managed by the Task Manager tool or PARMON monitoring the cluster resources (discussed later).

Single Virtual Networking: This means that any node can access any network connection throughout the cluster domain even if the network is not physically connected to all nodes in the cluster.

Single Memory Space: This illusion of shared memory over memories associated with nodes of the cluster (discussed later).

Single Job Management System: A user can submit a job from any node using a transparent job submission mechanism. Jobs can be scheduled to run in either batch, interactive, or parallel modes (discussed later). Example systems include LSF and CODINE.

Single User Interface: The user should be able to use the cluster through a single GUI. The interface must have the same look and feel of an interface that is available for workstations (e.g., Solaris OpenWin or Windows NT GUI).

Availability Support Functions

Single I/O Space (SIOS): This allows any node to perform I/O operation on local or remotely located peripheral or disk devices. In this SIOS design, disks associated with cluster nodes, RAIDs, and peripheral devices form a single address space.

Single Process Space: Processes have a unique cluster-wide process id. A process on any node can create child processes on the same or different node (through a UNIX fork) or communicate with any other process (through signals and pipes) on a remote node. This cluster should support globalized process management and allow the management and control of processes as if they are running on local machines.

Checkpointing and Process Migration: Checkpointing mechanisms allow a process state and intermediate computing results to be saved periodically. When a node fails, processes on the failed node can be restarted on another working node without the loss of computation. Process migration allows for dynamic load balancing among the cluster nodes.

1.10 Resource Management and Scheduling (RMS)

Resource Management and Scheduling (RMS) is the act of distributing applications among computers to maximize their throughput. It also enables the effective and efficient utilization of the resources available. The software that performs the RMS consists of two components: a resource manager and a resource scheduler. The resource manager component is concerned with problems, such as locating and allocating computational resources, authentication, as well as tasks such as process creation and migration. The resource scheduler component is concerned with tasks such as queuing applications, as well as resource location and assignment.

RMS has come about for a number of reasons, including: load balancing, utilizing spare CPU cycles, providing fault tolerant systems, managed access to powerful systems, and so on. But the main reason for their existence is their ability to provide an increased, and reliable, throughput of user applications on the systems they manage.

The basic RMS architecture is a client-server system. In its simplest form, each computer sharing computational resources runs a server daemon. These daemons maintain up-to-date tables, which store information about the RMS environment in which it resides. A user interacts with the RMS environment via a client program, which could be a Web browser or a customized X-windows interface. Application can be run either in interactive or batch mode, the latter being the more commonly used. In batch mode, an application run becomes a job that is submitted to the RMS system to be processed. To submit a batch job, a user will need to provide job details to the system via the RMS client. These details may include information such as location of the executable and input data sets, where standard output is to be placed, system type, maximum length of run, whether the job needs sequential or parallel resources, and so on. Once a job has been submitted to the RMS environment, it uses the job details to place, schedule, and run the job in the appropriate way.

RMS environments provide middleware services to users that should enable heterogeneous environments of workstations, SMPs, and dedicated parallel platforms to be easily and efficient utilized. The services provided by a RMS environment can include:

Process Migration - This is where a process can be suspended, moved, and restarted on another computer within the RMS environment. Generally, process migration occurs due to one of two reasons: a computational resource has become too heavily loaded and there are other free resources, which can be utilized, or in conjunction with the process of minimizing the impact of users, mentioned below.

Checkpointing - This is where a snapshot of an executing program's state is saved and can be used to restart the program from the same point at a later time if necessary. Checkpointing is generally regarded as a means of providing reliability. When some part of an RMS environment fails, the programs executing on it can be restarted from some intermediate point in their run, rather than restarting them from scratch.

Scavenging Idle Cycles - It is generally recognized that between 70 percent and 90 percent of the time most workstations are idle. RMS systems can be set up to utilize idle CPU cycles. For example, jobs can be submitted to workstations during the night or at weekends. This way, interactive users are not impacted by external jobs and idle CPU cycles can be taken advantage of.

Fault Tolerance - By monitoring its jobs and resources, an RMS system can provide various levels of fault tolerance. In its simplest form, fault tolerant support can mean that a failed job can be restarted or rerun, thus guaranteeing that the job will be completed.

Minimization of Impact on Users - Running a job on public workstations can have a great impact on the usability of the workstations by interactive users. Some RMS systems attempt to minimize the impact of a running job on interactive users by either reducing a job's local scheduling priority or suspending the job. Suspended jobs can be restarted later or migrated to other resources in the systems.

Load Balancing - Jobs can be distributed among all the computational platforms available in a particular organization. This will allow for the efficient and effective usage of all the resources, rather than a few which may be the only ones that the users are aware of. Process migration can also be part of the load balancing strategy, where it may be beneficial to move processes from overloaded system to lightly loaded ones.

Multiple Application Queues - Job queues can be set up to help manage the resources at a particular organization. Each queue can be configured with certain attributes. For example, certain users have priority of short jobs run before long jobs. Job queues can also be set up to manage the usage of specialized resources, such as a parallel computing platform or a high performance graphics workstation. The queues in an RMS system can be transparent to users; jobs are allocated to them via keywords specified when the job is submitted.

There are many commercial and research packages available for RMS; a few popular ones are listed in Table 1.2. There are several in-depth reviews of the available RMS systems [5], [20].

Table 1.2 Some Popular Resource Management Systems

Project	Commercial Systems - URL
LSF	http://www.platform.com/
CODINE	http://www.genias.de/products/codine/tech_desc.html
Easy-LL	http://www.tc.cornell.edu/UserDoc/SP/LL12/Easy/
NQE	http://www.cray.com/products/software/nqe/
	Public Domain Systems - URL
CONDOR	http://www.cs.wisc.edu/condor/
GNQS	http://www.gnqs.org/
DQS	http://www.scri.fsu.edu/~pasko/dqs.html
PRM	http://gost.isi.edu/gost-group/products/prm/
PBS	http://pbs.mrj.com/

1.11 Programming Environments and Tools

The availability of standard programming tools and utilities have made clusters a practical alternative as a parallel-processing platform. In this section we discuss a few of the most popular tools.

1.11.1 Threads

Threads are a popular paradigm for concurrent programming on uniprocessor as well as multiprocessors machines. On multiprocessor systems, threads are primarily used to simultaneously utilize all the available processors. In uniprocessor systems, threads are used to utilize the system resources effectively. This is achieved by exploiting the asynchronous behavior of an application for overlapping computation and communication. Multithreaded applications offer quicker response to user input and run faster. Unlike forked process, thread creation is cheaper and easier to manage. Threads communicate using shared variables as they are created within their parent process address space.

Threads are potentially portable, as there exists an IEEE standard for POSIX threads interface, popularly called pthreads. The POSIX standard multithreading interface is available on PCs, workstations, SMPs, and clusters [21]. A programming language such as Java has built-in multithreading support enabling easy development of multithreaded applications. Threads have been extensively used in developing both application and system software (including an environment used to create this chapter and the book as a whole!).

1.11.2 Message Passing Systems (MPI and PVM)

Message passing libraries allow efficient parallel programs to be written for distributed memory systems. These libraries provide routines to initiate and configure the messaging environment as well as sending and receiving packets of data. Currently, the two most popular high-level message-passing systems for scientific and engineering application are the PVM (Parallel Virtual Machine) [22] from Oak Ridge National Laboratory, and MPI (Message Passing Interface) defined by MPI Forum [8].

PVM is both an environment and a message passing library, which can be used to run parallel applications on systems ranging from high-end supercomputers through to clusters of workstations. Whereas MPI is a message passing specification, designed to be standard for distributed memory parallel computing using explicit message passing. This interface attempts to establish a practical, portable, efficient, and flexible standard for message passing. MPI is available on most of the HPC systems, including SMP machines.

The MPI standard is the amalgamation of what were considered the best aspects of the most popular message passing systems at the time of its conception. It is the result of the work undertaken by the MPI Forum, a committee composed of vendors and users formed at the SC'92 with the aim of defining a message passing standard. The goals of the MPI design were portability, efficiency and functionality. The standard only defines a message passing library and leaves, among other things, the initialization and control of processes to individual developers to define. Like PVM, MPI is available on a wide range of platforms from tightly coupled systems to metacomputers. The choice of whether to use PVM or MPI to develop a parallel application is beyond the scope of this chapter, but, generally, application developers choose MPI, as it is fast becoming the de facto standard for message passing. MPI and PVM libraries are available for Fortran 77, Fortran 90, ANSI C and C++. There also exist interfaces to other languages – one such example is `mpiJava` [23].

1.11.3 Distributed Shared Memory (DSM) Systems

The most efficient, and widely used, programming paradigm on distributed memory systems is message passing. A problem with this paradigm is that it is complex and difficult to program compared to shared memory programming systems. Shared memory systems offer a simple and general programming model, but they suffer from scalability. An alternate cost-effective solution is to build a DSM system on distributed memory system, which exhibits simple and general programming model and scalability of a distributed memory systems.

DSM enables shared-variable programming and it can be implemented by using software or hardware solutions. The characteristics of software DSM systems are: they are usually built as a separate layer on top of the communications interface; they take full advantage of the application characteristics; virtual pages, objects, and language types are units of sharing. Software DSM can be implemented either solely by run-time, compile time, or combined approaches. Two representative software DSM systems are TreadMarks [24] and Linda [25]. The characteristics of hardware DSM systems are: better performance (much faster than software DSM), no burden on user and software layers, fine granularity of sharing, extensions of the cache coherence schemes, and increased hardware complexity. Two examples of hardware DSM systems are DASH [26] and Merlin [27].

1.11.4 Parallel Debuggers and Profilers

To develop correct and efficient high performance applications it is highly desirable to have some form of easy-to-use parallel debugger and performance profiling tools. Most vendors of HPC systems provide some form of debugger and performance analyzer for their platforms. Ideally, these tools should be able to work in a heterogeneous environment, thus making it possible to develop and implement a parallel application on, say a NOW, and then actually do production runs on a dedicated HPC platform, such as the Cray T3E.

Debuggers

The number of parallel debuggers that are capable of being used in a cross-platform, heterogeneous, development environment is very limited. Therefore, in 1996 an effort was begun to define a cross-platform parallel debugging standard that defined the features and interface users wanted. The High Performance Debugging Forum (HPDF) was formed as a Parallel Tools Consortium project [28]. The forum has developed a HPD Version specification which defines the functionality, semantics, and syntax for a command-line parallel debugger. Ideally, a parallel debugger should be capable of:

- Managing multiple processes and multiple threads within a process.
- Displaying each process in its own window.
- Displaying source code, stack trace, and stack frame for one or more processes.
- Diving into objects, subroutines, and functions.
- Setting both source-level and machine-level breakpoints.
- Sharing breakpoints between groups of processes.
- Defining watch and evaluation points.
- Displaying arrays and its slices.
- Manipulating code variables and constants.

TotalView

TotalView is a commercial product from Dolphin Interconnect Solutions [29]. It is currently the only widely available GUI-based parallel debugger that supports multiple HPC platforms. TotalView supports most commonly used scientific languages (C, C++, F77/F90 and HPF), message passing libraries (MPI and PVM) and operating systems (SunOS/Solaris, IBM AIX, Digital UNIX and SGI IRIX). Even though TotalView can run on multiple platforms, it can only be used in homogeneous environments, namely, where each process of the parallel application being debugged must be running under the same version of the OS.

1.11.5 Performance Analysis Tools

The basic purpose of performance analysis tools is to help a programmer to understand the performance characteristics of an application. In particular, it should analyze and locate parts of an application that exhibit poor performance and create program bottlenecks. Such tools are useful for understanding the behavior of normal sequential applications and can be enormously helpful when trying to analyze the performance characteristics of parallel applications.

Most performance monitoring tools consist of some or all of the following components:

- A means of inserting instrumentation calls to the performance monitoring routines into the user's application.
- A run-time performance library that consists of a set of monitoring routines that measure and record various aspects of a program performance.
- A set of tools for processing and displaying the performance data.

A particular issue with performance monitoring tools is the intrusiveness of the tracing calls and their impact on the applications performance. It is very important to note that instrumentation affects the performance characteristics of the parallel application and thus provides a false view of its performance behavior. Table 1.3 shows the most commonly used tools for performance analysis of message passing programs.

1.11.6 Cluster Administration Tools

Monitoring clusters is a challenging task that can be eased by tools that allow entire clusters to be observed at different levels using a GUI. Good management software is crucial for exploiting a cluster as a high performance computing platform.

There are many projects investigating system administration of clusters that support parallel computing, including Berkeley NOW [4], SMILE [30] (Scalable Multicomputer Implementation using Low-cost Equipment), and PARMON [31]. The Berkeley NOW system administration tool gathers and stores data in a relational database. It uses a Java applet to allow users to monitor a system from their browser. The SMILE administration tool is called K-CAP. Its environment consists of compute nodes (these execute the compute-intensive tasks), a management node (a file server and cluster manager as well as a management console), and a client that can control and monitor the cluster. K-CAP uses a Java applet to connect to the management node through a predefined URL address in the cluster. The Node Status Reporter (NSR) provides a standard mechanism for measurement and access to status information of clusters [32]. Parallel applications/tools can access NSR through the NSR Interface. PARMON is a comprehensive environment for monitoring large clusters. It uses client-server techniques to provide transparent access to all nodes to be monitored. The two major components of PARMON are the

Table 1.3 Performance Analysis and Visualization Tools

Tool	Supports	URL
AIMS	instrumentation, monitoring library, analysis	http://science.nas.nasa.gov/Software/AIMS
MPE	logging library and snapshot performance visualization	http://www.mcs.anl.gov/mpi/mpich
Pablo	monitoring library and analysis	http://www-pablo.cs.uiuc.edu/Projects/Pablo/
Paradyn	dynamic instrumentation runtime analysis	http://www.cs.wisc.edu/paradyn
SvPablo	integrated instrumentor, monitoring library and analysis	http://www-pablo.cs.uiuc.edu/Projects/Pablo/
Vampir	monitoring library performance visualization	http://www.pallas.de/pages/vampir.htm
Dimemas	performance prediction for message passing programs	http://www.pallas.com/pages/dimemas.htm
Paraver	program visualization and analysis	http://www.cepba.upc.es/paraver

parmon-server (system resource activities and utilization information provider) and the parmon-client (a Java applet or application capable of gathering and visualizing realtime cluster information).

1.12 Cluster Applications

Earlier in this chapter we have discussed the reasons why we would want to put together a high performance cluster, that of providing a computational platform for all types of parallel and distributed applications. The class of applications that a cluster can typically cope with would be considered grand challenge or super-computing applications. GCAs (Grand Challenge Applications) are fundamental problems in science and engineering with broad economic and scientific impact [33]. They are generally considered intractable without the use of state-of-the-art parallel computers. The scale of their resource requirements, such as processing time, memory, and communication needs distinguishes GCAs.

A typical example of a grand challenge problem is the simulation of some phenomena that cannot be measured through experiments. GCAs include massive crystallographic and microtomographic structural problems, protein dynamics and biocatalysis, relativistic quantum chemistry of actinides, virtual materials design and processing, global climate modeling, and discrete event simulation.

The design and implementation of various GCAs on clusters has been discussed in Volume 2 of this book [34].

1.13 Representative Cluster Systems

There are many projects [35] investigating the development of supercomputing class machines using commodity off-the-shelf components. We briefly describe the following popular efforts:

- Network of Workstations (NOW) project at University of California, Berkeley.
- High Performance Virtual Machine (HPVM) project at University of Illinois at Urbana-Champaign.
- Beowulf Project at the Goddard Space Flight Center, NASA.
- Solaris-MC project at Sun Labs, Sun Microsystems, Inc., Palo Alto, CA.

1.13.1 The Berkeley Network Of Workstations (NOW) Project

The Berkeley NOW project [4] demonstrates building of a large-scale parallel computing system using mass produced commercial workstations and the latest commodity switch-based network components. To attain the goal of combining distributed workstations into a single system, the NOW project included research and development into network interface hardware, fast communication protocols, distributed file systems, distributed scheduling, and job control. The architecture of NOW system is shown in Figure 1.4.

Interprocess Communication

Active Messages (AM) is the basic communications primitives in Berkeley NOW. It generalizes previous AM interfaces to support a broader spectrum of applications such as client/server programs, file systems, operating systems, and provide continuous support for parallel programs. The AM communication is essentially a simplified remote procedure call that can be implemented efficiently on a wide range of hardware. NOW includes a collection of low-latency, parallel communication primitives: Berkeley Sockets, Fast Sockets, shared address space parallel C (Split-C), and MPI.

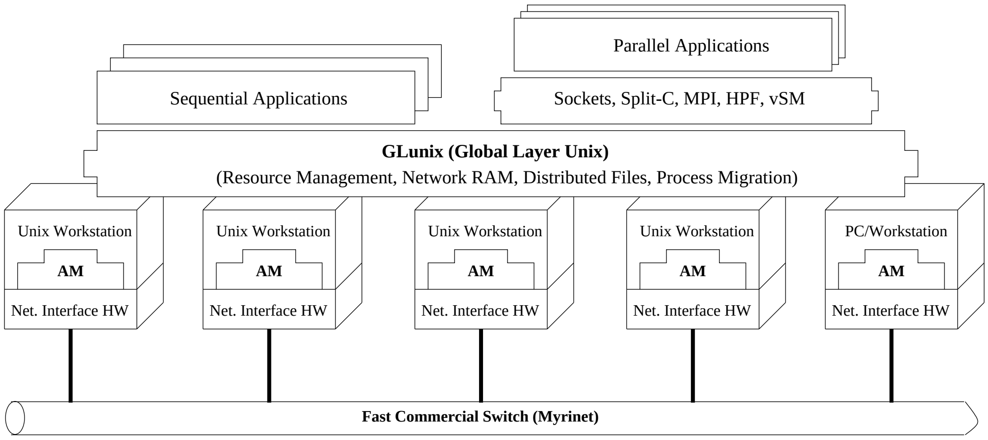

Figure 1.4 Architecture of NOW system.

Global Layer Unix

(GLUnix) GLUnix is an OS layer designed to provide transparent remote execution, support for interactive parallel and sequential jobs, load balancing, and backward compatibility for existing application binaries. GLUnix is a multiuser system implemented at the userlevel so that it can be easily ported to a number of different platforms. GLUnix aims to provide a cluster-wide namespace and uses Network PIDs (NPIDs) and Virtual Node Numbers (VNNs). NPIDs are globally unique process identifiers for both sequential and parallel programs throughout the system. VNNs are used to facilitate communications among processes of a parallel program. A suite of user tools for interacting and manipulating NPIDs and VNNs, equivalent to UNIX run, kill, etc. are supported. A GLUnix API allows interaction with NPIDs and VNNs.

Network RAM

Network RAM allows us to utilize free resources on idle machines as a paging device for busy machines. The designed system is serverless, and any machine can be a server when it is idle, or a client when it needs more memory than physically available. Two prototype systems have been developed. One of these uses custom Solaris segment drivers to implement an external user-level pager, which exchanges pages with remote page daemons. The other provides similar operations on similarly mapped regions using signals.

xFS: Serverless Network File System

xFS is a serverless, distributed file system, which attempts to have low latency, high bandwidth access to file system data by distributing the functionality of the server among the clients. The typical duties of a server include maintaining cache coherence, locating data, and servicing disk requests. The function of locating data in xFS is distributed by having each client responsible for servicing requests on a subset of the files. File data is striped across multiple clients to provide high bandwidth.

1.13.2 The High Performance Virtual Machine (HPVM) Project

The goal of the HPVM project [36] is to deliver supercomputer performance on a low cost COTS (commodity-off-the-shelf) system. HPVM also aims to hide the complexities of a distributed system behind a clean interface. The HPVM project provides software that enables high performance computing on clusters of PCs and workstations. The HPVM architecture (Figure 1.5) consists of a number of software components with high-level APIs, such as MPI, SHMEM, and Global Arrays, that allows HPVM clusters to be competitive with dedicated MPP systems.

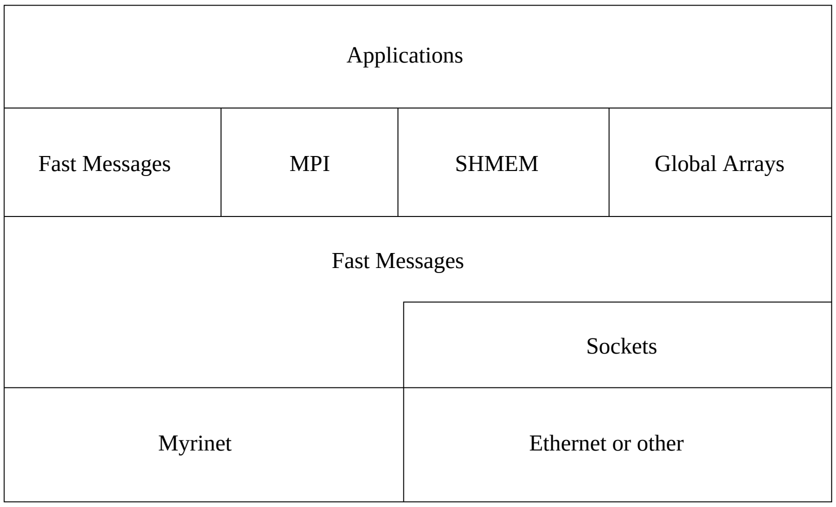

Figure 1.5 HPVM layered architecture.

The HPVM project aims to address the following challenges:

- Delivering high performance communication to standard, high-level APIs.
- Coordinating scheduling and resource management.
- Managing heterogeneity.

A critical part of HPVM is a high-bandwidth and low-latency communications protocol known as Fast Messages (FM), which is based on Berkeley AM. Unlike other messaging layers, FM is not the surface API, but the underlying semantics.

FM contains functions for sending long and short messages and for extracting messages from the network. The services provided by FM guarantees and controls the memory hierarchy that FM provides to software built with FM. FM also guarantees reliable and ordered packet delivery as well as control over the scheduling of communication work.

The FM interface was originally developed on a Cray T3D and a cluster of SPARCstations connected by Myrinet hardware. Myricom's Myrinet hardware is a programmable network interface card capable of providing 160 MBytes/s links with switch latencies of under a μs. FM has a low-level software interface that delivers hardware communication performance; however, higher-level layers interface offer greater functionality, application portability, and ease of use.

1.13.3 The Beowulf Project

The Beowulf project's [6] aim was to investigate the potential of PC clusters for performing computational tasks. Beowulf refers to a Pile-of-PCs (PoPC) to describe a loose ensemble or cluster of PCs, which is similar to COW/NOW. PoPC emphasizes the use of mass-market commodity components, dedicated processors (rather than stealing cycles from idle workstations), and the use of a private communications network. An overall goal of Beowulf is to achieve the 'best' overall system cost/performance ratio for the cluster.

System Software

The collection of software tools being developed and evolving within the Beowulf project is known as *Grendel.* These tools are for resource management and to support distributed applications. The Beowulf distribution includes several programming environments and development libraries as separate packages. These include PVM, MPI, and BSP, as well as, SYS V-style IPC, and pthreads.

The communication between processors in Beowulf is through TCP/IP over the Ethernet internal to cluster. The performance of interprocessor communications is, therefore, limited by the performance characteristics of the Ethernet and the system software managing message passing. Beowulf has been used to explore the feasibility of employing multiple Ethernet networks in parallel to satisfy the internal data transfer bandwidths required. Each Beowulf workstation has user-transparent access to multiple parallel Ethernet networks. This architecture was achieved by 'channel bonding' techniques implemented as a number of enhancements to the Linux kernel. The Beowulf project has shown that up to three networks can be ganged together to obtain significant throughput, thus validating their use of the channel bonding technique. New network technologies, such as Fast Ethernet, will ensure even better interprocessor communications performance.

In the interests of presenting a uniform system image to both users and applications, Beowulf has extended the Linux kernel to allow a loose ensemble of nodes to participate in a number of global namespaces. In a distributed scheme it is often convenient for processes to have a PID that is unique across an entire cluster, spanning several kernels. Beowulf implements two Global Process ID (GPID) schemes. The first is independent of external libraries. The second, GPID-PVM, is designed to be compatible with PVM Task ID format and uses PVM as its signal transport. While the GPID extension is sufficient for cluster-wide control and signaling of processes, it is of little use without a global view of the processes. To this end, the Beowulf project is developing a mechanism that allows unmodified versions of standard UNIX utilities (e.g., ps) to work across a cluster.

1.13.4 Solaris MC: A High Performance Operating System for Clusters

Solaris MC (Multicomputer) [37] is a distributed operating system for a multicomputer, a cluster of computing nodes connected by a high-speed interconnect. It provides a single system image, making the cluster appear like a single machine to the user, to applications, and to the network. The Solaris MC is built as a globalization layer on top of the existing Solaris kernel, as shown in Figure 1.6. It extends operating system abstractions across the cluster and preserves the existing Solaris ABI/API, and hence runs existing Solaris 2.x applications and device drivers without modifications. The Solaris MC consists of several modules: C++ and object framework; and globalized process, file system, and networking.

The interesting features of Solaris MC include the following:

- Extends existing Solaris operating system
- Preserves the existing Solaris ABI/API compliance
- Provides support for high availability
- Uses C++, IDL, CORBA in the kernel
- Leverages Spring technology

The Solaris MC uses an object-oriented framework for communication between nodes. The object-oriented framework is based on CORBA and provides remote object method invocations. It looks like a standard C++ method invocation to the programmers. The framework also provides object reference counting: notification to object server when there are no more references (local/remote) to the object. Another feature of the Solaris MC object framework is that it supports multiple object handlers.

A key component in proving a single system image in Solaris MC is the global file system. It provides consistent access from multiple nodes to files and file attributes and uses caching for high performance. It uses a new distributed file system called ProXy File System (PXFS), which provides a globalized file system without the need for modifying the existing file system.

The second important component of Solaris MC supporting a single system image is its globalized process management. It globalizes process operations such as signals. It also globalizes the /proc file system providing access to process state

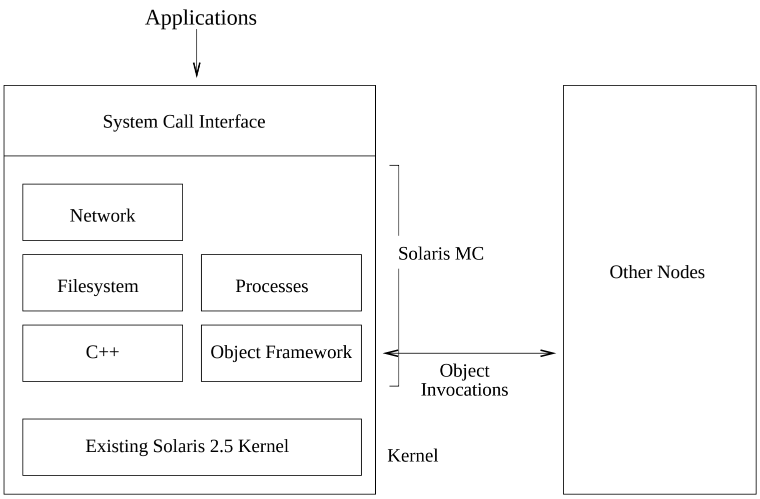

Figure 1.6 Solaris MC architecture.

for commands such as 'ps' and for the debuggers. It supports remote execution, which allows to start up new processes on any node in the system.

Solaris MC also globalizes its support for networking and I/O. It allows more than one network connection and provides support to multiplex between arbitary the network links.

1.13.5 A Comparison of the Four Cluster Environments

The cluster projects described in this chapter share a common goal of attempting to provide a unified resource out of interconnected PCs or workstations. Each system claims that it is capable of providing supercomputing resources from COTS components. Each project provides these resources in different ways, both in terms of how the hardware is connected together and the way the system software and tools provide the services for parallel applications.

Table 1.4 shows the key hardware and software components that each system uses. Beowulf and HPVM are capable of using any PC, whereas Berkeley NOW and Solaris MC function on platforms where Solaris is available – currently PCs, Sun workstations, and various clone systems. Berkeley NOW and HPVM use Myrinet with a fast, low-level communications protocol (Active and Fast Messages). Beowulf uses multiple standard Ethernet, and Solaris MC uses NICs, which are supported by Solaris and ranges from Ethernet to ATM and SCI.

Each system consists of some middleware interfaced into the OS kernel, which is used to provide a globalization layer, or unified view, of the distributed cluster resources. Berkeley NOW uses the Solaris OS, whereas Beowulf uses Linux with a modified kernel and HPVM is available for both Linux and Windows NT. All four

Table 1.4 Cluster Systems Comparison Matrix

Project	Platform	Communications	OS	Other
`Beowulf`	PCs	Multiple Ethernet with TCP/IP	Linux and Grendel	MPI/PVM, Sockets and HPF
`Bereley NOW`	Solaris-based PCs and workstations	Myrinet and Active Messages	Solaris + GLUunix + xFS	AM, PVM, MPI, HPF, Split-C
`HPVM`	PCs	Myrinet with Fast Messages	NT or Linux connection and global resource manager + LSF	Java-frontend, FM, Sockets, Global Arrays, SHMEM and MPI
`Solaris MC`	Solaris-based PCs and workstations	Solaris-supported	Solaris + Globalization layer	C++ and CORBA

systems provide a wide variety of tools and utilities commonly used to develop, test, and run parallel applications. These include various high-level APIs for message passing and shared-memory programming.

1.14 Cluster of SMPs (CLUMPS)

The advances in hardware technologies in the area of processors, memory, and network interfaces, is enabling the availability a low cost and small configuration (2-8 multiprocessors) shared memory SMP machines. It is also observed that clusters of multiprocessors (CLUMPS) promise to be the supercomputers of the future. In CLUMPS, multiple SMPs with several network interfaces can be connected using high performance networks.

This has two advantages: It is possible to benefit from the high performance, easy-to-use-and-program SMP systems with a small number of CPUs. In addition, clusters can be set up with moderate effort (for example, a 32-CPU cluster can be constructed by using either commonly available eight 4-CPU SMPs or four 8-CPU SMPs instead of 32 single CPU machines) resulting in easier administration and better support for data locality inside a node.

This trend puts a new demand on cluster interconnects. For example, a single NIC will not be sufficient for an 8-CPU system and will necessitate the need for multiple network devices. In addition, software layers need to implement multiple mechanisms for data transfer (via shared memory inside an SMP node and the network to other nodes).

1.15 Summary and Conclusions

In this chapter we have discussed the different hardware and software components that are commonly used in the current generation of cluster-based systems. We have also described four state-of-the-art projects that are using subtly different approaches ranging from an all-COTS approach to a mixture of technologies. In this section we summarize our findings, and make a few comments about possible future trends.

1.15.1 Hardware and Software Trends

In the last five years several important advances have taken place. Prominent among them are:

- A network performance increase of tenfold using 100BaseT Ethernet with full duplex support.
- The availability of switched network circuits, including full crossbar switches for proprietary network technologies such as Myrinet.
- Workstation performance has improved significantly.
- Improvement of microprocessor performance has led to the availability of desktop PCs with performance of low-end workstations, but at significantly lower cost.
- The availability of fast, functional, and stable OSs (Linux) for PCs, with source code access.
- The performance gap between supercomputer and commodity-based clusters is closing rapidly.
- Parallel supercomputers are now equipped with COTS components, especially microprocessors (SGI-Cray T3E - DEC Alpha), whereas earlier systems had custom components.
- Increasing usage of SMP nodes with two to four processors.

A number of hardware trends have been quantified in [38]. Foremost of these is the design and manufacture of microprocessors. A basic advance is the decrease in feature size which enables circuits to work faster or consume low power. In conjunction with this is the growing die size that can be manufactured. These factors mean that:

- The average number of transistors on a chip is growing by about 40 percent per annum.
- The clock frequency growth rate is about 30 percent per annum.

It is anticipated that by the year 2000 there will be 700 MHz processors with about 100 million transistors.

There is a similar story for storage, but the divergence between memory capacity and speed is more pronounced. Memory capacity increased by three orders of magnitude between 1980 and 1995, yet its speed has only doubled. It is anticipated that Gigabit DRAM will be available in early 2000, but the gap to processor speed is getting greater all the time.

The problem is that memories are getting larger while processors are getting faster. So getting access to data in memory is becoming a bottleneck. One method of overcoming this bottleneck is to configure the DRAM in banks and then transfer data from these banks in parallel. In addition, multilevel memory hierarchies organized as caches make memory access more effective, but their design is complicated. The access bottleneck also applies to disk access, which can also take advance to parallel disks and caches.

The ratio between the cost and performance of network interconnects is falling rapidly. The use of network technologies such as ATM, SCI, and Myrinet in clustering for parallel processing appears to be promising. This has been demonstrated by many commercial and academic projects such as Berkeley NOW and Beowulf. But no single network interconnect has emerged as a clear winner. Myrinet is not a commodity product and costs a lot more than Ethernet, but it has real advantages over it: very low-latency, high bandwidth, and a programmable on-board processor allowing for greater flexibility. SCI network has been used to build distributed shared memory system, but lacks scalability. ATM is used in clusters that are mainly used for multimedia processing.

Two of the most popular operating systems of the 1990s are Linux and NT. Linux has become a popular alternative to a commercial operating system due to its free availability and superior performance compared to other desktop operating systems such as NT. Linux currently has more than 7 million users worldwide and it has become the researcher's choice of operating system.

NT has a large installed base and it has almost become a ubiquitous operating system. NT 5 will have a thinner and faster TCP/IP stack, which supports faster communication of messages, yet it will use standard communication technology. NT systems for parallel computing is in a situation similar to the UNIX workstation five to seven years ago and it is only a matter of time before NT catches up–NT developers need not invest time or money on research as they are borrowing most of the technology developed by the UNIX community!

1.15.2 Cluster Technology Trends

We have discussed a number of cluster projects within this chapter. These range from those which are commodity but proprietary components based (Berkeley NOW) to a totally commodity system (Beowulf). HPVM can be considered as a hybrid-system using commodity computers and specialized network interfaces. It should be noted that the projects detailed in this chapter are a few of the most popular and well known, rather than an exhaustive list of all those available.

All the projects discussed claim to consist of commodity components. Although this is true; one could argue, however, that true commodity technologies would be those that are pervasive at most academic or industrial sites. If this were the case, then true commodity would mean PCs running Windows 95 with standard 10 Mbps Ethernet. However, when considering parallel applications with demanding computational and network needs, this type of low-end cluster would be incapable of providing the resources needed.

Each of the projects discussed tries to overcome the bottlenecks that arise while using cluster-based systems for running demanding parallel applications in a slightly different way. Without fail, however, the main bottleneck is not the computational resource (be it a PC or UNIX workstation), rather it is the provision of a low-latency, high-bandwidth interconnect and an efficient low-level communications protocol to provide high-level APIs.

The Beowulf project explores the use of multiple standard Ethernet cards to overcome the communications bottleneck, whereas Berkeley NOW and HPVM use programmable Myrinet cards and AM/FM communications protocols. Solaris MC uses Myrinet NICs and TCP/IP. The choice of what is the best solution cannot just be based on performance; the cost per node to provide the NIC should also be considered. For example, a standard Ethernet card costs less than \$100, whereas Myrinet cards cost in excess of \$1000 each. Another factor that must also be considered in this equation is the availability of Fast Ethernet and the advent of GigaBit Ethernet. It seems that Ethernet technologies are likely to be more mainstream, mass produced, and consequently cheaper than specialized network interfaces. As an aside, all the projects that have been discussed are in the vanguard of the cluster computing revolution and their research is helping the following army determine which are the best techniques and technologies to adopt.

1.15.3 Future Cluster Technologies

Emerging hardware technologies along with maturing software resources mean that cluster-based systems are rapidly closing the performance gap with dedicated parallel computing platforms. Cluster systems that scavenge idle cycles from PCs and workstations will continue to use whatever hardware and software components are available on public workstations. Clusters dedicated to high performance applications will continue to evolve as new and more powerful computers and network interfaces become available in the market place.

It is likely that individual cluster nodes will be SMPs. Currently two and four processor PCs and UNIX workstations are becoming common. Software that allows SMP nodes to be efficiently and effectively used by parallel applications will be developed and added to the OS kernel in the near future. It is likely that there will be widespread usage of Gigabit Ethernet and, as such, it will become the de facto standard for clusters. To reduce message passing latencies cluster software systems will bypass the OS kernel, thus avoiding the need for expensive system calls, and exploit the usage of intelligent network cards. This can obviously be achieved using intelligent NICs, or alternatively using on-chip network interfaces such as those used by the new DEC Alpha 21364.

The ability to provide a rich set of development tools and utilities as well as the provision of robust and reliable services will determine the choice of the OS used on future clusters. UNIX-based OSs are likely to be most popular, but the steady improvement and acceptance of Windows NT will mean that it will be not far behind.

1.15.4 Final Thoughts

Our need for computational resources in all fields of science, engineering and commerce far weigh our ability to fulfill these needs. The usage of clusters of computers is, perhaps, one of most promising means by which we can bridge the gap between our needs and the available resources. The usage of COTS-based cluster systems has a number of advantages including:

- Price/performance when compared to a dedicated parallel supercomputer.
- Incremental growth that often matches yearly funding patterns.
- The provision of a multipurpose system: one that could, for example, be used for secretarial purposes during the day and as a commodity parallel supercomputing at night.

These and other advantages will fuel the evolution of cluster computing and its acceptance as a means of providing commodity supercomputing facilities.

Acknowledgments

We thank Dan Hyde, Toni Cortes, Lars Rzymianowicz, Marian Bubak, Krzysztof Sowa, Lori Pollock, Jay Fenwick, Eduardo Pinheiro, and Miguel Barreiro Paz for their comments and suggestions on this chapter.

1.16 Bibliography

[1] G. Pfister. *In Search of Clusters.* Prentice Hall PTR, NJ, 2nd Edition, NJ, 1998.

[2] K. Hwang and Z. Xu. *Scalable Parallel Computing: Technology, Architecture, Programming.* WCB/McGraw-Hill, NY, 1998.

[3] C. Koelbel et al. *The High Performance Fortran Handbook.* The MIT Press, Massachusetts, 1994.

[4] T. Anderson, D. Culler, and D. Patterson. A Case for Networks of Workstations. *IEEE Micro*, Feb. 95. http://now.cs.berkeley.edu/

[5] M.A. Baker, G.C. Fox, and H.W. Yau. *Review of Cluster Management Software.* NHSE Review, May 1996. http://www.nhse.org/NHSEreview/CMS/

[6] *The Beowulf Project.* http://www.beowulf.org

[7] *QUT Gardens Project.* http://www.fit.qut.edu.au/CompSci/PLAS/

[8] *MPI Forum.* http://www.mpi-forum.org/docs/docs.html

[9] *The Berkeley Intelligent RAM Project.* http://iram.cs.berkeley.edu/

[10] *The Standard Performance Evaluation Corporation (SPEC).* http://open.specbench.org

[11] Russian Academy of Sciences. *VLSI Microprocessors: A Guide to High Performance Microprocessors.* http://www.microprocessor.sscc.ru/

[12] ATM Forum. *ATM User Level Network Interface Specification.* Prentice Hall, NJ, June 1995.

[13] *SCI Association.* http://www.SCIzzL.com/

[14] *MPI-FM: MPI for Fast Messages.* http://www-csag.cs.uiuc.edu/projects/comm/mpi-fm.html

[15] N. Boden et. al. Myrinet - A Gigabit-per-Second Local-Area Network. *IEEE Micro*, February 1995. http://www.myri.com/

[16] *The Linux Documentation Project.* http://sunsite.unc.edu/mdw/linux.html

[17] *Parallel Processing using Linux.* http://yara.ecn.purdue.edu/~pplinux/

[18] H. Custer. *Inside Windows NT.* Microsoft Press, NY, 1993.

[19] Kai Hwang et. al. Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space. *IEEE Concurrency*, vol.7(1), Jan.- March, 1999.

[20] J. Jones and C. Bricknell. *Second Evaluation of Job Scheduling Software.* http://science.nas.nasa.gov/Pubs/TechReports/NASreports/NAS-97-013/

[21] F. Mueller. On the Design and Implementation of DSM-Threads. *In Proceedings of the PDPTA'97 Conference*, Las Vegas, USA, 1997.

[22] *The PVM project.* http://www.epm.ornl.gov/pvm/

[23] `mpiJava` *Wrapper.* http://www.npac.syr.edu/projects/prpc/mpiJava/, Aug. 1998.

[24] *TreadMarks.* http://www.cs.rice.edu/~willy/TreadMarks/overview.html

[25] N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, April 1989.

[26] D. Lenoski et al. The Stanford DASH Multiprocessor. *IEEE Computer*, March 1992.

[27] C. Mapples and Li Wittie. Merlin: A Superglue for Multiprocessor Systems. *In Proceedings of CAMPCON'90*, March 1990.

[28] *Parallel Tools Consortium project.* http://www.ptools.org/

[29] *Dolphin Interconnect Solutions.* http://www.dolphinics.no/

[30] P. Uthayopas et. al. Building a Resources Monitoring System for SMILE Beowulf Cluster. *In Proceedings of HPC Asia98 Conference*, Singapore, 1998.

[31] R. Buyya et. al. PARMON: A Comprehensive Cluster Monitoring System. *In Proceedings of the AUUG'98 Conference*, Sydney, Australia, 1998.

[32] C. Roder et. al. Flexible Status Measurement in Heterogeneous Environment. *In Proceedings of the PDPTA'98 Conference*, Las Vegas, 1998.

[33] *Grand Challenging Applications.* http://www.mcs.anl.gov/Projects/grand-challenges/

[34] R. Buyya. *High Performance Cluster Computing: Programming and Applications.* vol. 2, Prentice Hall PTR, NJ, 1999.

[35] *Computer Architecture Links.* http://www.cs.wisc.edu/~arch/www/

[36] *HPVM.* http://www-csag.cs.uiuc.edu/projects/clusters.html

[37] *Solaris MC.* http://www.sunlabs.com/research/solaris-mc/

[38] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach.* M. K. Publishers, San Francisco, CA, 1998.