Cluster Computing White Paper

Status – First Release

Version 1.01

Date – 5th July 2000

Editor - Mark Baker, University of Portsmouth, UK

Contributing authors:

Amy Apon, University of Arkansas, USA,
David Bader, New Mexico, USA,
Mark Baker, University of Portsmouth, UK,
Shirley Browne, University of Tennessee, USA,
Rajkumar Buyya, Monash University, Australia,
Steve Chapin, Syracuse University, USA,
Toni Cortes, UPC, Barcelona, Spain,
Jack Dongarra, University of Tennessee, and ORNL, USA,
Graham Fagg, University of Tennessee, USA,
Salim Hariri, University of Arizona, USA,
Daniel C. Hyde, Bucknell University, USA,
Daniel S. Katz, JPL, California Institute of Technology, Pasadena, CA, USA,
Jeremy Kepner, MIT Lincoln Laboratory, Lexington, MA, USA,
Robert Pennington, NCSA, USA,
Ira Pramanick, Sun Microsystems, USA,
Erich Schikuta, University of Vienna, Austria,
Anne Trefethen, Numerical Algorithms Group Ltd, UK,
Helmut Wanek, University of Vienna, Austria,
Barry Wilkinson, University of North Carolina at Charlotte, USA
Joachim Worringen, RWTH Aachen, Germany.

1. Introduction

Cluster computing is not a new area of computing. It is, however, evident that there is a growing interest in its usage in all areas where applications have traditionally used parallel or distributed computing platforms. The growing interest has been fuelled in part by the availability of powerful microprocessors and high-speed networks as off-the-shelf commodity components as well as in part by the rapidly maturing software components available to support high performance and high availability applications.

This mounting interest in clusters led to the formation of an IEEE Computer Society Task Force on Cluster Computing (TFCC[1]) in early 1999. An objective of the TFCC was to act both as a magnet and a focal point for all cluster computing related activities. As such, an early activity that was deemed necessary was to produce a White Paper on cluster computing and its related technologies.

Generally a White Paper is looked upon as a statement of policy on a particular subject. The aim of this White Paper is to provide a relatively unbiased report on the existing, new and emerging technologies as well as the surrounding infrastructure deemed important to the cluster computing community.

This White Paper provides an authoritative review all the hardware and software technologies that can be used to make up a cluster now or in the near future. These technologies range from the network level, through the operating system and middleware levels up to the application and tools level. The White Paper also tackles the increasingly important areas of High Availability as well as Education, which is considered a crucial area for the future success of cluster computing.

The White Paper has been broken down into eleven sections, each of which has been put together by academics and industrial researchers who are both experts in their fields and where willing to volunteer their time and effort to put together this White Paper.

The status of this paper is draft and we are at the stage of publicizing its presence and making a Request For Comments (RFC).

Mark Baker
University of Portsmouth, UK
April 2000

[1] TFCC – `http://www.dcs.port.ac.uk/~mab/tfcc/`

2. Network Technologies

Amy Apon, University of Arkansas, USA, Mark Baker, University of Portsmouth, UK, Graham Fagg, University of Tennessee, USA, Salim Hariri, University of Arizona, USA, and Joachim Worringen, RWTH Aachen, Germany.

2.1 Introduction

The performance of cluster computing systems depends largely on the bandwidth and latency of the inter-cluster network. Achieving high bandwidth and low latency requires not only fast hardware, but also efficient communication protocols and host-network interfaces that minimize the communication software overhead.

The problem of providing the needed communication bandwidth for parallel and distributed algorithms is now relatively easy to solve, given the mature state of fiber-optics and opto-electronic device technologies. Achieving the low latencies necessary, however, remains a challenge. Reducing latency requires progress on a number of fronts:

- Current communication protocols do not scale well to a high-speed environment. To keep latencies low it is desirable to execute the entire protocol stack, up to the transport layer, in hardware.
- The communication interface of the operating system must be streamlined to allow direct transfer of data from the network interface to the memory space of the application program.

In general, achieving low latency requires a two-pronged approach:

- Latency Reduction – Minimize protocol-processing overhead by using streamlined protocols executed in hardware and by improving the network interface of the operating system.
- Latency Hiding – Modify the computational algorithm to hide latency by pipelining communication and computation.

These problems are now perhaps most fundamental to the success of high-performance distributed computing, a fact that is increasingly being recognized by the research community.

2.2 Background and Overview

A cluster communication system can be viewed as a collection of physical and logical components that together perform the communication tasks. The physical components (network devices) transfer data between the host memory and the communication medium. The logical components provide services for message assembly and/or de-assembly, buffering, formatting, routing and error checking. Consequently, the design of a communication system involves defining the resources required to implement the functions associated with each component. The physical components determine the type of network to be used (LANs, MANs, WANs), the type of network topology (fully connected, bus, tree, ring, mixture, and random), and the type of communication medium (twisted pair, coaxial cables, fiber optics, wireless, and satellite). The physical components also determine how the host is to access the network resources. The logical components determine the type of communication services (packet switching, message switching, circuit switching), type of information (data, voice, facsimile, image and video), management techniques (centralized and/or distributed), and type of communication protocols. The network issues that need to be addressed to implement the communication infrastructure includes:

1) Network Type – the implementation of the physical computer network.
2) Communications Protocols – the design of communication protocols on distributed system performance.
3) Host Network Interface – the techniques used to implement high performance network interfaces.

2.3 Network Types

Several types of networks are often used in cluster computing. These networks include Ethernet, Fast Ethernet, Gigabit Ethernet, SCI, Myrinet, and cLAN. Other networks used less often but which deserve mention include HIPPI, ATM, Fibre Channel, and AmpNet.

2.3.1 Ethernet, Fast Ethernet, and Gigabit Ethernet

Ethernet has been, and continues to be, the most widely used technology for local area networking. Standard Ethernet transmits at 10Mbps and has a large installation base. However, this bandwidth is no longer sufficient for use in environments where users are transferring large data quantities. Standard Ethernet cannot be used seriously as the basis for cluster computing since its bandwidth is not balanced compared to the computational power of the computers now available. An improved version, commonly known as Fast Ethernet, provides 100Mbps bandwidth and is designed to provide an upgrade path for existing Ethernet installations.

Both Standard and Fast Ethernet are based on the concept of a collision domain. Within a collision domain, the communication channel is shared so that only one node can send at a time. If another node tries to send at the same time then a collision occurs, both messages are garbled, and both nodes retry their communication after a random waiting period. With a shared Ethernet or Fast Ethernet hub, all nodes share the same collision domain and only one node on the network can send at a time. With a switched hub, each node may be on a different collision domain, depending on the manufacturer's specifications. In this case, then two nodes can send at the same time as long as they are not both sending to the same receiver. In general, all switched networks, not just Ethernet networks, have this capability. For cluster computing, switches are preferred since the ability for more than one node to send at the same time can greatly improve overall performance.

Although the name of Gigabit Ethernet sounds like the traditional Ethernet that has been used for decades in local area networks, all Gigabit Ethernet products except for the lowest end hubs are based on high-speed point-to-point switches in which each node is in its own separate collision domain. Gigabit Ethernet transmits signals at 1Gbps, but because of the encoding required at high speeds, has an effective user data rate of approximately 800Mbps. In addition, because of the physical nature of sending high-speed signals, Gigabit Ethernet products may have additional distance and wiring requirements over Fast Ethernet products. Currently, due to the speed of the processor, a single computer can transmit data onto a network faster than Ethernet bandwidth, but somewhat slower than Fast Ethernet bandwidth. The speed of processors is increasing rapidly. It is likely that the next generation of processors will be able to fill the capacity of a Fast Ethernet network. However, it will be some time before a single processor will be able to fill the capacity of a Gigabit Ethernet network.

Fast Ethernet products are affordable and provide adequate capacity and latency for many cluster-computing applications. Most new computers come with a built-in Fast Ethernet adapter, and small switches can be purchased for less than $50. Gigabit Ethernet products are expensive, but generally less than other gigabit networking products.

One disadvantage of Gigabit Ethernet and Fast Ethernet products is that they are not capable of native support of the Virtual Interface Architecture (VIA) standard (described later). A native VIA implementation off-loads a portion of the processing required to send and receive messages to a processor on the network interface card. By doing this, networks that support VIA operate at very low latencies. However, Gigabit Ethernet and Fast Ethernet do have emulated VIA implementations. While these implementations still require the host CPU to perform much of the processing for sending and receiving messages, they can operate with lower latencies than the more traditional TCP/IP protocols.

2.3.2 Scalable Coherent Interface (SCI)

SCI was the first interconnection technology to be developed as a standard specifically for the purposes of cluster computing. To surpass the performance of bus-based systems, SCI is a point-to-point architecture with small packet sizes and split transactions while maintaining the impression of a bus-functionality from the upper layers. The IEEE 1596 standard [1] for SCI was published in 1992 and specifies the physical layers of the network as well as higher layers for cache-coherent distributed shared memory across a network, which can optionally be cache-coherent. The standard allows serial fiber-optic links and parallel copper links on the physical layer. The bandwidth of current implementation of SCI links is located between 400 and 1000 MBytes/s, and typical remote access latencies are little more than 2µs.

Since the lower SCI protocol layers are already meaningful for themselves, it has been possible for many commercial products to be developed using only these lower layers. Dolphin Interconnect Solutions, Inc. currently produces SCI adapter cards for IO-buses like the Sun SPARC SBus and PCI. This solution provides non-cache-coherent shared memory and is thus well suited for message passing [2, 3] or efficient software-supported distributed shared memory [4, 5].

At higher layers, SCI defines a distributed, pointer-based cache coherence scheme. This allows for caching of remote SCI memory: whenever data located in remote memory is modified, all cache lines on all nodes that, store this data are invalidated. Caching of remote SCI memory increases performance and allows for true, transparent shared memory programming. Examples for the implementation of cache-coherent SCI variants are the HP Convex, Sequent NUMA-Q (now IBM) and Data General AViiON NUMA systems. The IO-bus based solutions cannot implement the cache-coherence protocol since they have no access to the CPU bus to monitor memory accesses.

Next to multi-processor coupling, SCI is also used to implement I/O networks or transparently extend I/O buses like PCI: I/O address space from one bus is mapped into another one providing an arbitrary number of devices. Examples for this usage are the SGI/Cray GigaRing and Siemens' external I/O expansion for the RM600 enterprise servers [2].

Although SCI features a point-to-point architecture that makes the ring topology most natural, other (logical) topologies are possible as well using switches. Switches can be separate devices providing star – or switched-ringlet-topologies, or they can be integrated into the nodes allowing 2D- or even 3D-Torus topologies [2]. The cost of SCI components is in the range of other Gigabit-interconnects (less than US $1000 per node), making it an affordable solution even for "self-made" clusters. Switches are not needed for small clusters, but will improve performance and reliability if more than 4 nodes are to be used.

The Dolphin PCI-SCI adapters have support for every major operating system (Windows, Solaris, Linux, Lynx and more), CPU architecture (x86, PPC, Alpha) and offer a unified user-level programming interface [SISCI]. The usual parallel programming models are available, partly commercially [3], mostly as free software [8, 9, 5, 4]. A native VIA interface is not yet available, but is currently under development at [10].

.

SCI is an appealing cluster interconnect. Its performance is comparable or better than other Gigabit interconnects, adding to these the easy and efficient user-level communication via shared memory. Valuable sources of information about SCI are the IEEE standard [1] and a recently published book [11] to which many people from the SCI arena have contributed.

2.3.3 Giganet cLAN

Giganet [12] was the first industry provider of a native hardware implementation of the VIA standard. Giganet began shipping the cLAN product suite for Windows NT in fall 1998 and for Linux in fall 1999. In 1999, Giganet products include PCI adapters that support 1.25Gbps high-speed cluster interconnection and 1024 virtual interfaces, switches with up to 30 ports, and a software tool for managing and testing of the cluster network. Giganet products are expensive, but are likely to be a top competitor in the cluster computing market.

2.3.4 Myrinet

Myrinet [13] is a 1.28Gbps full duplex network supplied by Myricom. It was used in the Berkeley NOW cluster and many other academic clusters. Myrinet uses low-latency cut-through routing switches, which simplify network setup and offer fault tolerance by automatically mapping the network configuration. Myrinet has a programmable on-board processor that allows for experimentation with communication protocols. Myrinet offers a native implementation of VIA, as well as supporting the MPICH implementation of MPI directly. There are Myrinet drivers for many operating systems, including Linux, Solaris, and Windows NT.

Myrinet is expensive when compared with Fast Ethernet. However, by mid-2000, Myrinet switches with up to 32 ports are expected to be available, which will alleviate some previous problems with scaling. In addition, its low latency, high bandwidth, and programmability make it competitive for cluster computing.

2.4 Others

Several other high-speed networks have been used for cluster computing. A few of these bear mentioning, including HIPPI, ATM, Fibre Channel, and AmpNet.

2.4.1 HIPPI

HIPPI [14], or HIgh Performance Parallel Interface, is a gigabit network that was first designed for interconnecting high-performance parallel computers with network attached storage devices. When the first 800Mbps HIPPI standard was developed in the late 1980's, it was considered revolutionary. The standard has gone through several versions and now bears the name Gigabit System Network (GSN). GSN is the highest bandwidth and lowest latency interconnect standard, providing full duplex transmission rates of 6400Mbps (800 Mbytes/s) in each direction. GSN and HIPPI products are competitively priced when considering their high throughput. However, their cost prohibits their widespread usage, except in applications that require the highest performance.

2.4.2 ATM

ATM [15], or Asynchronous Transmission Mode, is an international network standard that grew out of the needs of the telecommunications industry. ATM is designed around the concept of cells, or small, fixed-sized packets. ATM is a switch-based technology that uses the idea of virtual circuits to deliver a connection-oriented service. ATM offers several classes of data delivery. Some classes are designed for voice and video communication, which require real-time guarantees on data delivery rates. Due to the recent consolidation of many telecommunications industry and Internet services, ATM is being increasingly used as a backbone technology in many organizations. While ATM products offer high bandwidth, in comparison to other similar technologies, it is relatively expensive.

2.4.3 Fibre Channel

Fibre Channel [16] is a set of standards that defines a high performance data transport connection technology that can transport many kinds of data at speeds up to 1Gbps, through copper wire or fiber optic cables. The Fibre Channel standard is defined in layers, beginning at the physical layer. Like the SCI standard, the standards layers are independent, enabling many companies to produce networking and cluster products based on the lower layers only. Fibre Channel is flexible, and mapping protocols have been defined which enable higher-level protocols, such as SCSI, IP, ATM, and HIPPI, to be used over Fibre Channel connections. Currently, the most frequent usage of Fibre Channel in clusters is for data storage devices. The high performance network capacity of Fibre Channel can be combined with SCSI and other I/O bridges to create a network of storage devices, separate from the primary cluster interconnect. A network of this type is sometimes called a Storage Area Network (also, SAN) and is an important subsystem of many commercial clusters. As Fibre Channel matures it may become more widely used as a general-purpose cluster interconnection technology.

2.4.4 AmpNet

AmpNet is a gigabit network product supplied by Belobox Networks [17]. After many years of use in military products, a PCI-based network card is available in mid-2000 that will make AmpNet an option for PC-based clusters and commercial products. In addition to its low latency and gigabit speed, AmpNet provides a level of fault-tolerance and self-healing not normally found in commercial networks. AmpNet may become a strong competitor for high availability cluster applications.

2.5 Communication Protocols

A protocol is a set of precisely defined rules and conventions for communication between two parties. A communication protocol defines the rules and conventions that will be used by two or more computers on the network to exchange information. Communication protocols can be classified as

- Connection-oriented or connectionless.
- Offering various levels of reliability, including fully guaranteed to arrive in order (reliable), or not guaranteed (unreliable)
- Unbuffered (synchronous), or buffered (asynchronous). Protocols are sometimes classified by the number of intermediate copies between buffers, which may be zero, one, or more.

Several protocols are used in clusters, including traditional protocols that were originally designed for the Internet, such as TCP/IP, and protocols that have been designed specifically for cluster communication, such as Active Messages, Fast Messages, VMMC, Unet, and BIP messages. The Virtual Interface Architecture (VIA) is a protocol standard that has been specially designed for use in cluster computing.

2.5.1 Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)

The standard TCP and UDP protocols and the *de facto* standard BSD sockets Application Programmer's Interface (API) to TCP and UDP were the first messaging libraries used for cluster computing. TCP and UDP are both transport layer protocols built over the Internet Protocol (IP). TCP offers a reliable, connection-oriented service between hosts on a network. UDP is an unreliable, connectionless transport layer service.

The TCP and UDP protocols are typically implemented using one or more buffers in system memory and with operating system services. To send a message, a user application constructs the message in user memory, and then makes an operating system request to copy the message into a system buffer. An interrupt is required for this operating system intervention before the message can be sent out by network hardware. When a message is received by network hardware, the network copies the message to system memory, and an interrupt is used to copy the message to user memory and to notify the receiving process that the message has arrived. This operating system overhead and the copies to and from system memory are a significant portion of the total time to send a message. As networks became faster during the 1990's, the overhead of operating system services became significantly larger than the actual hardware transmission time for messages. The goal of the low-latency messaging systems that were developed during the 1990's was to avoid operating system intervention while at the same time provide user-level messaging services across high-speed networks.

Low-overhead messaging systems developed during the 1990's include Active Messages, Fast Messages, the VMMC (Virtual Memory-Mapped Communication) system, and U-net, among others. By 1997, research on low-latency protocols had progressed sufficiently for a standards process to be initiated, and the Virtual Interface Architecture design document was formulated and published as a standard low-latency networking architecture in less than a year.

2.5.2 Active Messages

Active Messages [18], which was later extended to Generic Active Messages (GAM), is the enabling low-latency communications library for the Berkeley Network of Workstations (NOW) system. Short messages in Active Messages are synchronous, and are based on the concept of a request-reply protocol. The sending user-level application constructs a message in user memory. To transfer the data, the receiving process allocates a receive buffer, also in user memory on the receiving side, and sends a request to the sender. The sender replies by copying the message from the user buffer on the sending side directly to the network. No buffering in system memory is performed. Network hardware transfers the message to the receiver, and then the message is transferred from the network to the receive buffer in user memory. This process requires that memory on both the sending and receiving side be pinned to an address in physical memory. However, once the user memory buffers are established, no operating system intervention is required for a message to be sent. This protocol is also called a "zero-copy" protocol, since no copies from user memory to system memory are used. In GAM, a copy sometimes occurs to a buffer in system memory on the receiving side so that user buffers can be reused more efficiently. In this case, the protocol is referred to as a “one-copy” protocol. Fast Messages is a protocol similar to Active Messages, but extends Active Messages by imposing stronger guarantees on the underlying communication. In particular, Fast Messages guarantees that all messages arrive reliably and in-order, even if the underlying networks hardware does not.

2.5.3 VMMC

The Virtual Memory-Mapped Communication [19] system, later extended to VMMC-2, is the enabling low-latency protocol for the Princeton SHRIMP system. One goal of VMMC is to view messaging as reads and writes into the user-level virtual memory system. VMMC works by mapping a page of user virtual memory to physical memory, and in addition makes a correspondence between pages on the sending and receiving sides. VMMC also uses specially designed hardware that allows the network interface to snoop writes to memory on the local host and have these writes automatically updated on the remote hosts memory. Various optimizations of these writes have been developed that help to minimize the total number of writes, network traffic, and overall application performance. VMMC is an example of a class of systems known as distributed shared memory systems. In distributed shared memory systems memory is physically distributed among the nodes in a system, but processes in an application may view “shared” locations as identical and perform reads and writes to the “shared” locations.

2.5.4 U-net

The U-net network interface architecture [20] strongly influenced the design of the Virtual Interface Architecture. In addition to providing zero-copy messaging where possible, U-net, and VIA add the concept of a virtual network interface for each connection in a user application. Just as an application has a virtual memory address space that is mapped to real physical memory on demand, each communication endpoint of the application is viewed as a virtual network interface mapped to a real set of network buffers and queues on demand. The advantage of this architecture is that once the mapping is defined, each active interface has direct access to the network without operating system intervention. The result is that communication can occur with very low latency.

2.5.5 BIP

BIP [21] (Basic Interface for Parallelism) is a low level message layer for Myrinet based clusters developed at the University of Lyon. The system is designed to be used by higher-level layers much like NEXUS, so that BIP users would really use interfaces such as MPI. The initial BIP interface consisted of both blocking and non-blocking calls, with later versions (BIP-SMP) providing multiplexing between Myrinet and shared memory under a single API for use on clusters of SMPs.

The important features of BIP are its low overhead (5μs latency) and high bandwidth (126 MBytes/s) achieved by using different protocols for various message sizes (like NX/2 on the Intel Paragon) and a zero or single memory copy of user data. To simply the design and keep the overheads low, BIP guarantees in-order delivery of messages, although some flow control issues for small messages are passed to higher software levels. Flow control appearing to be a complication in all NICs that require the system to "pin-down" main memory in the host computers, for example the credit system used to manage pinned down memory in the FM system.

The highest level interface currently supported under BIP is a modified version of MPICH called MPI-BIP [22], which offers 7.6μs latency and 107Mbytes/s bandwidth across the

Myrinet network, with 3.3μs latency and 150Mbytes/s bandwidth intra-node across shared memory. Currently BIP-SMP is only supported under the Linux operating system.

2.5.6 VIA

The Virtual Interface Architecture [23] (VIA) is a standard that combines many of the best features of various academic projects, and will strongly influence the evolution of cluster computing. Although VIA can be used directly for application programming, the Virtual Interface Architecture is considered by many systems designers to be at too low a level for application programming. With VIA, the application must be responsible for allocating some portion of physical memory and using it effectively. It is expected that most operating system and middleware vendors will provide an interface to VIA that is suitable for application programming. Generally, the interface to VIA that is suitable for application programming comes in the form of a message-passing interface for scientific or parallel programming.

2.6 Conclusions

2.6.1 Summary

It is clear that all aspects of networks and their associated technologies are rapidly changing to meet the needs of current, new and emerging applications. At present it appears that network hardware is technologically, further advanced than the supporting software. The lag in software support for network hardware is typical in the computer industry. It is compounded by the fact that developers of network software technologies are not only concerned with low latency and high-bandwidth communication. Today there is an increasing need for features like, guaranteed Quality of Service (QoS) and the need for specific protocols to deal with streaming, groups and object technologies, to fulfil the needs of specific applications.

Clusters are based around COTS technologies. Therefore, the scope of exactly what can be done to a cluster is limited to what technology is available and how much it costs. Much of the research in this area is focusing on how to efficiently and effectively utilse those hardware components currently available. In particular, developing relatively low-level protocols that can optimally use network hardware and provide applications with services that are appropriate for their needs. It is interesting to note here, that the majority of applications are still based around message passing as opposed to other paradigms such as Distributed Shared Memory (DSM).

2.6.2 The Future

It is likely that there will be widespread usage of Gigabit Ethernet, as this technology will provide a low-risk solution for moving from Standard or Fast Ethernet. The price for using this technology will be higher communication latencies [24]. Where it is necessary for an application to attain very low latencies, network technologies that bypass the operating system kernel, thus avoiding the need for expensive system calls, will be used. This can be achieved using intelligent network interface cards, or alternatively using on-chip network interfaces such as those on the DEC Alpha 21364. Obviously, the appropriate software protocols will be necessary to support this technology and it is evident that there has been a general acceptance of VIA, both hardware and software vendors are designing products to this specification.

2.7 References

1 IEEE Standard 1596-1992, IEEE Standard for Scalable Coherent Interface (SCI), *Inst. of Electrical and Electronics Eng.*, Inc., New York, NY, August 1993

2 hpcLine – http://www.siemens.de/computer/hpc/en/hpcline/

3 Scali, ScaMPI – http//www.scali.com

4 M. Schulz, True Shared Memory Programming on SCI-based Clusters, *LNCS 1734*, Chapter 17, Springer, Heidelberg, 1999

5 K. Scholtyssik, M. Dormanns, Simplifying the use of SCI shared memory by using software SVM techniques, *Proc. 2nd Workshop Cluster Computing*, Karlsruhe, March 1999.

6 Interconnect Solutions AS, Oslo, Version 2.1.1. March 1999

7 F. Giacomini, T. Amundsen, A. Bogaerts, R. Hauser, B.D. Johnsen, H. Kohmann, R. Nordström, and P. Werner, Low-level SCI software functional specification, *Esprit Project 23174*, CERN, Geneva and Dolphin Interconnect Solutions AS, Oslo, Version 2.1.1. March 1999

8 M. Fischer, A.Reinefeld, PVM for SCI Clusters, *LNCS 1734*, Chapter 13, Springer, Heidelberg, 1999

9 J. Worringen, T. Bemmerl, MPICH for SCI-Connected Clusters, *Proc. SCI-Europe 1999*, pp. 3-11, Toulouse, 1999. – http://www.lfbs.rwth-aachen.de/users/joachim/SCI-MPICH

10 Chemnitz – http://www.tu-chemnitz.de/informatik/RA/

11 H. Hellwagner, A. Reinefeld (Eds.), SCI: Scalable Coherent Interface, Architecture & Software for High-Performance Compute Clusters, *LNCS 1734*, Springer, Heidelberg, 1999

12 Giganet – http://www.giganet.com

13 N. Boden, et. al. Myrinet – A Gigabit-per-Second Local-Area Network. *IEEE Micro*, Vol. 15, No.1, February 1995 – http://www.myri.com/

14 HIPPI Networking Forum – http://www.hnf.org/

15 ATM Forum. ATM User-Network Interface Specification, *Prentice Hall*, September 1993.

16 Fibre Channel Fundamentals, T. Weimer, Spectra Logic Corporation – http://www.spectralogic.com/fibre/fcfund.htm

17 Belobox Networks – http://www.belobox.com/

18 T. von Eicken, D. Culler, S. Goldstein and K. Shauser, Active Messges: a mechanism for integrated communications and computation, *Procs of International Symposium on Computer Architectures*, 1992.

19 M.A. Blumrich, C. Dubnicki, E.W. Felten, K. Li, and M.R. Mesarin, Virtual Memory Mapped Network Interfaces, *IEEE Micro*, February 1995

20 A. Basu, M. Welsh and T. von Eicken, Incorporating Memory Management into User-Level Network Interfaces, Presented at *Hot Interconnects V*, August 1997, Stanford University and also a Cornell University, Technical Report, TR97-1620 – http://www2.cs.cornell.edu/U-Net/papers.html

21 L. Prylli and B. Tourancheau. BIP: a new protocol designed for high performance networking on myrinet. In the PC-NOW workshop, IPPS/SPDP 1998, Orlando, USA, 1998.

22 L. Prylli, B. Tourancheau, and R. Westrelin, The design for a high performance mpi implementation on the myrinet network. In EuroPVM/MPI 99, *Lecture Notes in Computer Science*, Vol 1697, pp. 223-230, Springer, 1999.

23 Virtual Interface Architecture – http://www.viarch.org/

24 M.A. Baker, S.L. Scott, et al, Gigabit Ethernet Performance under NT and Linux, to appear in the International Parallel and Distributed Processing Symposium (IPDPS) 2000, Cancun, Mexico, May 2000.

3. Operating Systems

Steve Chapin, Syracuse University, USA and Joachim Worringen, RWTH Aachen, University of Technology, Germany

3.1. Introduction

Just as in a conventional desktop system, the operating system for a cluster lies at the heart of every node. Whether the user is opening files, sending messages, or starting additional processes, the operating system is omnipresent. While users may choose to use differing programming paradigms or middleware layers, the operating system is almost always the same for all users.

What then, is the role of the operating system in a cluster? The primary role is the same twofold task as in a desktop system: multiplex multiple user processes onto a single set of hardware components (resource management), and provide useful abstractions for high-level software (beautification). Some of these abstractions include protection boundaries, process/thread coordination and communication as well as device handling. Therefore, in the remainder of this section, we will examine the abstractions provided by current cluster operating systems, and explore current research issues for clusters.

3.2. Background and Overview

The ideal operating system would always help, and never hinder, the user. That is, it would help the user (which in this case is an application – or middleware-designer) to configure the system for optimal program execution by supplying a consistent and well-targeted set of functions offering as many system resources as possible. After setting up the environment, it is desired to stay out of the user's way avoiding any time-consuming context switches or excessive set up of data structures for performance-sensitive applications. The most common example of this is in high-speed message passing, in which the operating system pins message buffers in DMA-able memory, and then allows the network interface card, possibly guided by user-level message calls, to transfer messages directly to RAM without operating system intervention. On the other hand, it might be desired that the operating system offer a rich functionality for security, fault-tolerance and communication facilities – which of course contrasts with the need for performance that is omnipresent.

Exactly what functionality a cluster operating system should provide is still an open question. Here is a small list of desirable features:

- *Manageability*: An absolute necessity is remote and intuitive system administration; this is often associated with a Single System Image (SSI) which can be realized on different levels, ranging from a high-level set of special scripts, perhaps controlled via Java-enabled graphical front-end, down to real state-sharing on the OS level.
- *Stability*: The most important characteristics are robustness against crashing processes, failure recovery by dynamic reconfiguration, and usability under heavy load.
- *Performance*: The performance critical parts of the OS, such as memory management, process and thread scheduler, file I/O and communication protocols should work in as efficiently as possible. The user and programmer should be able to transparently modify the relevant parameters to fine-tune the OS for his specific demands.
- *Extendibility*: The OS should allow the easy integration of cluster-specific extensions; this implies, at a minimum, user-loadable device drivers and profound documentation of interfaces, and probably means source code access.

- *Scalability*: The scalability of a cluster is mainly influenced by the properties of the contained nodes, which is dominated by the performance characteristics of the interconnect. This includes the support of the OS to be able to use the potential performance of the interconnect by enabling low-overhead calls to access the interconnect (inter-node scalability). However, clusters are usually built with SMP nodes with an increasing number of CPUs contained in each node. The ability of the OS to benefit from these is determined by its intra-node scalability. This also includes the resource limits that an OS exhibits, foremost the maximum size of usable address space.
- *Support*: Many intelligent and technically superior approaches in computing failed due to the lack of support in its various aspects: which tools, hardware drivers and middleware environments are available. This support depends mainly on the number of users of a certain system, which in the context of clusters is mainly influenced by the hardware costs (because usually dozens of nodes are to be installed). Additionally, support for interconnect hardware; availability of open interfaces or even open source; support or at least demand by the industry to fund and motivate research and development are important. All this leads to a user community that employs required middleware, environments and tools to, at least, enable cluster applications.
- *Heterogeneity*: Clusters provide a dynamic and evolving environment in that they can be extended or updated with standard hardware just as the user needs to or can afford. Therefore, a cluster environment does not necessarily consists of homogenous hardware requiring the same operating system should run across multiple architectures or at least support a set of standardized APIs to simplify the development of middleware layers enabling heterogeneous use.

It should be noted that experience shows that these goals may be mutually exclusive. For example, supplying a SSI at the operating system level, while a definite boon in terms of manageability, drastically inhibits scalability. Another example is the availability of the source code in conjunction with the possibility to extend (and thus modify) the operating system on this base. This property has a negative influence on the stability and manageability of the system: over time, many variants of the operating system will develop, and the different extensions may conflict when there is no single supplier.

3.3. Technological Scope

In this sub-section, we will touch on some of the aspects of operating systems for clusters that frequently lead to discussions among the developers and users in this area. Some of these aspects are covered in more detail in other sections of this paper.

3.3.1 Functionality: OS versus Middleware

There has been little work on operating systems specifically for clusters. Much of the work one might consider as affecting the operating system, e.g. the GLUnix [1] work at U.C. Berkeley, is actually middleware. There are good reasons for this. First, operating systems are complex, and it is not always clear how a proposed change will affect the rest of the system. Isolating these changes in middleware can make good sense. Second, the applicability of the new functionality added by such a middleware layer is usually not limited to a single operating system, but can be ported to other operating systems as well.

For example, support for Distributed Shared Memory (DSM) arises from shared address spaces, which in a conventional kernel would be inside the operating system. However, in distributed computing, DSM software such as SVMlib [2] is most often implemented in middleware, running on diverse operating systems such as Windows NT and Solaris. The advantage here is that a single middleware layer can provide services to multiple operating systems, without requiring access or changes to OS source code.

On the other hand, one can view Intel's Paragon OS as a cluster operating system, and Sun's Solaris MC [11] is specifically intended for cluster use. These provide some shared state between kernels (although much of the Solaris MC functionality is implemented at the user level). However, a weakness of current distributed operating systems is that state sharing is binary: they use all-or-nothing sharing, which inhibits scalability. The Tornado operating system is designed for 2-level hierarchical clusters of workstations, although it is not yet clear how well this approach will work for more generalized clusters.

3.3.2 Single System Image (SSI)

Regarding the frequently desired feature of SSI, at least two variants of it should be distinguished: SSI for system administration or job scheduling purposes and SSI on a system-call level. The first is usually achieved by middleware, running daemons or services on each node delivering the required information to the administration tool or job scheduler. The latter would have to offer features like transparent use of devices located on remote nodes or using distributed storage facilities as one single standard file system. These features require extensions to current single-node operating systems.

Another aspect of SSI is the cluster-wide transparent sharing of resources. Next to the sharing of I/O space via traditional network file systems or other, more sophisticated means of real parallel I/O, the sharing of RAM (Distributed Shared Memory, DSM) is an area in which a lot of research has been done. However, the usual approach to DSM is meant as a parallel programming paradigm to use shared memory between the processes of a parallel application distributed across a number of nodes. Therefore, it is mostly realized via user-level libraries [2] and not as a service offered by the OS. Integration of DSM into the OS has rarely been done [3, 5]. The reason for this is that the performance for general-purpose use (requiring strict sequential consistency) is often to low. Using relaxed consistency models improves performance, but requires that special care is taken by the user of the DSM system that prohibits offering it as a standard service by the OS. Next to using memory situated on remote nodes for DSM, some experiments [6] have been done to use it as OS-managed remote memory: current interconnect technologies like SCI or Myrinet offer lower latencies and higher bandwidth of inter-node communication than what can be achieved between the primary storage level (RAM) and the secondary storage level (hard disk) in intra-node communication. This leads to the idea to use the memory of remote nodes instead of the local hard disk for purposes like swapping or paging. This is a promising approach, which is, however, limited by the fact that it requires permanent memory related load-imbalances inside the cluster that is not desired in environments of dedicated compute clusters.

An operating-system-level SSI implies detailed state sharing across all nodes of the cluster, and to this point, OS researchers and practitioners have been unable to scale this to clusters of significant size (more than a hundred nodes) using commodity interconnects. That does not mean that an OS-level SSI is a bad thing; for the vast majority of clusters, which have less than 32 nodes, an operating-system-level single-system image may be quite workable.

3.3.3 Heterogeneity

It is arguable whether one should attempt to accommodate heterogeneous hardware at the operating system level within a single cluster. There are definite efficiencies to be gained from homogeneous clusters, and it may well make economic sense to replace an existing cluster rather than doing incremental heterogeneous expansion. Even if one accepts heterogeneity as inevitable, the operating system may not be the best place to address it. What we really want is that, at some layer, we provide a homogeneous set of abstractions to higher layers.

The lowest level on which heterogeneity causes problems is the data representation – big-endian vs. little-endian. If such systems are to be connected, the adaptation of the different representations could also be done on the lowest level possible to gain suitable performance. However, approaches to do endian conversion in hardware have not yet been done (the closest thing we are aware of is the ability of the old MIPS processors to run in either endianness, although there was no dynamic conversion).

On the other hand, this can just as easily be at the middleware layer instead of at the operating system layer (the "end-to-end" argument in networking would argue for pushing this to the highest layer possible). Middleware systems have done an excellent job of providing the illusion of homogeneity in heterogeneous systems – consider the success of Java and the Java Virtual Machine. However, creating illusions does cost in terms of performance. Therefore, we consider an operating system that runs on heterogeneous hardware as more of a serendipitous benefit rather than a requirement.

3.3.4 User-level communication facilities:

One characteristic of high-performance clusters is the interconnect between the nodes being not an Ethernet-based network, but more sophisticated network like SCI [7], Myrinet [8], GigaNet [9], or other, mostly proprietary solutions. This kind of hardware offers communication bandwidth in the range of several Gigabits/s—however, to make best use of this performance, it is required that the access to the interconnect adapter involves as little overhead as possible. Therefore, the involvement of the operating system in this kind of communication is not desired; everything is to be done in the user space, preferably by techniques like protected user-level DMA. However, it is expected that the multiplexing of an arbitrary number of processes to a single interconnect adapter is still possible in a secure manner. This imposes a difficult task to the developers of the interconnect adapter and its driver—and also to the designers of the operating system into which the driver is to be embedded. The VIA industry standard [10], as discussed in the communications section of this document, appears to be the future for low-latency, high-bandwidth cluster communication.

3.3.5 Optimized parallel I/O:

Less work has been done on high-speed file input/output concepts than for high-speed inter-process communication. However, file I/O is crucial for the performance of many types of applications, scientific codes as well as databases. The usual way is to employ a number (one or more) of dedicated I/O nodes in the cluster. However, every shared resource represents a potential bottleneck in a system that has to be scalable. A good example for the problems involved in this is, once again, the Intel Paragon. Its I/O concept was used for traditional file input/output as well as for swapping, is organized in a complex, tree-oriented parallel manner and nevertheless did not deliver optimal performance. Clusters should follow other concepts by doing as much node-local I/O as possible to reduce inter-node communication and I/O contention, while maintaining a consistent global view of the I/O space.

3.4. Current State-of-the-art

State-of-the-art can be interpreted to mean the most common solution as well as the best solution using today's technology. By far the most common solution current clusters is running a conventional operating system, with little or no special modification. This operating system is usually a Unix derivative, although NT clusters are becoming more common. We do not intend "cluster" to be synonymous with "cluster of PCs," although this is, again, the most common case, for reasons of economics.

The single most popular cluster operating system is Linux [19]. This is primarily for three reasons:

1. It is free,
2. It is an open source operating system, meaning that one is free to customize the kernel to one's liking. To date, this has usually meant specialized drivers for underlying high-speed networks, and other I/O optimizations.
3. For historic reasons: Don Becker selected Linux for the original Beowulf cluster (this was for both technical and social reasons, as the Linux kernel was free of any possible licensing problems, unlike the BSD derivatives at the time), and thus Beowulf-derived systems have also used Linux.

Sun Microsystems has developed a multi-computer version of Solaris; aptly named Solaris MC [11] Solaris MC consists of a small set of kernel extensions and a middleware library. Solaris MC incorporates some of the research advances from Sun’s Spring operating system, including an object-oriented methodology and the use of CORBA IDL in the kernel. Solaris MC provides a SSI to the level of the device, i.e. processes running on one node can access remote devices as if they were local. The SSI also extends to a global file system and a global process space.

The Puma operating system [12], from Sandia National Labs and the University of New Mexico, represents the ideological opposite of Solaris MC. Puma takes a true minimalist approach: there is no sharing between nodes, and there is not even a file system or demand-paged virtual memory. This is because Puma runs on the “compute partition” of the Intel Paragon and Tflops/s machines, while a full-featured OS (e.g. Intel’s TflopsOS or Linux) runs on the Service and I/O partitions. The compute partition is focused on high-speed computation, and Puma supplies low-latency, high-bandwidth communication through its Portals mechanism.

MOSIX [13,14] is a set of kernel extensions for Linux that provides support for seamless process migration. Under MOSIX, a user can launch jobs on their home node, and the system will automatically load balance and migrate the jobs to lightly-loaded nodes. MOSIX maintains a single process space, so the user can still track the status of their migrated jobs. MOSIX is a mature system, growing out of the MOS project and having been implemented for seven different operating systems/architectures. MOSIX is free and is distributed under the GNU Public License.

Next to the cluster operating systems mainly used in research and for running computationally intensive applications (which do not require a high degree of OS-support), clustering is also in use in the commercial arena. The main commercial applications in this area are data-intensive and often involve database management systems (DBMS). AIX from IBM has a strong position here, running the SP family of cluster-based servers, featuring proven stability, good manageability and a number of clustered databases. From the other commercial Unix variants, Sun’s Solaris has a strong focus on clustering, high availability (HA) and is also widely used in research. IRIX from SGI relies a sophisticated NUMA-SMP technology that provides a very specific kind of clustering. Other operating systems with an emphasis on HA (Tandem Non-Stop Unix and Unixware 7) are covered in the related section of this paper.

In the area of high-performance I/O, only a very limited number of proven and usable solutions exist. One of these is GPFS (General Parallel File System) [15] by IBM, specifically designed for the SP server running AIX. It is based on storage arrays (disks) being connected to one or more servers that exchange data with the clients via multithreaded daemons. It fulfils the criterion “usable” in that it features a kernel extension that can be accessed like any standard UNIX file system. This avoids the necessity of recompiling or even modifying applications that are to use GPFS. However, it performs best for large sequential reads and writes due to its technique of striping the data across the disks. Many technically more sophisticated but less general (and thus mostly less usable in a general cluster environment) research projects exist [16,17] which often deal with the special, but frequently occurring scenario of collective I/O in large scientific applications.

We present a comparison of the most relevant operating systems used for clustering in Appendix A. We are aware that already the selection criteria “most relevant”, and also the comparison of the selected systems will lead to discussions. However, covering every aspect of each system is surely beyond the scope of this paper, and a stimulation of the discussion is a desired effect. Apart from our own experiences, we consider the studies that have been performed by D.H. Brown Associates [18,19].

3.5. Future work

As SMP become more common in clusters, we will see a natural hierarchy arise. SMP have tight coupling, and will be joined into clusters via low-latency high-bandwidth interconnection networks. Indeed, we fully expect that heterogeneous clusters of SMP will arise, having single, dual, quad, and octo-processor boxes in the same cluster. These clusters will, in turn, be joined by gigabit-speed wide-area networks, which will differ from SAN primarily in their latency characteristics. This environment will naturally have a three-level hierarchy, with each level having an order of magnitude difference in latency relative to the next layer.

This structure will have its greatest impact on scheduling, both in terms of task placement and in terms of selecting a process to run from a ready queue. Scheduling is traditionally considered an operating system activity, yet it is quite likely that at least some of this work will be carried out in middleware.

For example, as one of our research projects we are investigating thread management for mixed-mode (multi-threaded and message passing) computing using OpenMP and MPI, which we believe is a natural by product of clusters of SMP. Most cluster applications, particularly scientific computations traditionally solved via spatial decomposition, consist of multiple cooperating tasks. During the course of the computation, hot spots will arise, and a self-adapting program might wish to manage the number of active threads it has in each process. Depending on the relationship of new threads to existing threads (and their communication pattern) and the system state, a decision might be made to do one of the following:

- Add a new thread on an idle processor of the SMP where the process is already running.
- Expand the address space via distributed shared memory to include an additional node, and add a thread there.
- Add a thread to a non-idle processor already assigned to the process.
- Migrate the process to a larger SMP (e.g. from 2 nodes to 4 nodes) with an idle processor, and add a new thread there.

The described technique of thread placing and process migration is also related to the high-availability issue that is critical for commercial applications. Automatic and fast migration of running processes, taking benefit from advanced interconnect technology offering e.g. transparent remote memory access, will lift the definition of fail-over times into new dimensions.

We might also examine the issue of configurability. Users might want to alter the personality of the local operating system, e.g. "strip down" to a Puma-like minimalist kernel to maximize the available physical memory and remove undesired functionality. Possible mechanisms to achieve this range from a reload of a new kernel and a reboot to dynamically linking/unlinking code into/out of the kernel. This leads to the question: “How much (and which) functionality does a cluster operating-system need?” The more functionality a system has, the more complicated it gets to maintain it and the chance for malfunctions due to bad configurations or errors in the interaction of different parts rises. Again, Linux is the easiest way for the majority of researchers to study this area.

Another important aspect of cluster computing in which the operating system is strongly involved is distributed file I/O. Current solutions of I/O systems are mostly static, do not adapt very well to the actual workload and thus tend to have bottlenecks, mostly by the limited number of dedicated I/O nodes to which all data has to be routed. The transparent filesystem-level support of distributed and adaptive I/O is an open issue for cluster operating systems. As an example for an attempt, we are currently implementing an MPI-IO implementation that operates on the basis of an existing DSM library on top of an SCI interconnect. This technique may result in a dedicated filesystem for high-bandwidth, low-latency requirements that is totally distributed among the participating clients.

3.6. Conclusions

Cluster operating systems are similar in many ways to conventional workstation operating systems. How different one chooses to make the operating system depends on one's view of clustering. On the one hand, we have those who argue that each node of a cluster *must* contain a full-featured operating system such as Unix, with all the positives and negatives that implies. At the other extreme, we see researchers asking the question, "Just how much can I remove from the OS and have it still be useful?" These systems are typified by the work going on in the Computational Plant project at Sandia National Laboratories. Still others are examining the possibility of on-the-fly adaptation of the OS layer, reconfiguring the available services through dynamic loading of code into the cluster operating system.

It is worth noting that every notable attempt to provide SSI at the OS layer has been regarded as a failure on some level. Sun’s Solaris MC has never been offered as a product, and recently sun has approached academic computer scientists to evaluate Linux on Sparc stations as the basis of a cluster product. Intel’s Paragon OS is well-known for its tendency to lock up the entire machine because of minor problems on one node, as well as its wretched performance on large systems. We’re not saying it’s impossible to build a scalable SSI at the OS level, we’re just saying that no one has done it, and we think there’s good reason. The forthcoming SIO standard will blur the edges between remote and local devices, and perhaps this will lead to more highly scalable SSI.

A final note is that, through a combination of OS improvements and acquisition of relevant technology, Microsoft has become a viable option in the realm of clustering. The HPVM [20] effort has demonstrated that it is possible to build a reasonable cluster from Windows NT boxes, and with the recent installation of clusters such as the one at Cornell University, NT or Windows 2000 is going to be a factor in the cluster OS picture for the foreseeable future.

To sum it up, clusters for technical and scientific computing based on Linux and other standalone Unix platforms like AIX are here, and they work. In the area of commercial cluster computing, Linux still lacks essential functionalities which “conventional” Unix systems and in parts even Windows NT do offer. It is to be observed if the free and distributed development model of Linux will be able to offer proven solutions in this area, too, since these topics are rarely addressed in the Linux developer community. Nevertheless, more exotic OS technology is and will be the current focus of many research efforts, both academic and commercial. There will probably never be “THE” cluster OS, as Linux will adopt research results much more quickly than commercial vendors, particularly Microsoft, if history is a reliable guide.

3.7. References

1. Global Layer Unix – http://now.cs.berkeley.edu/Glunix/glunix.html
2. K. Scholtyssik and M. Dormanns, Simplifying the use of SCI shared memory by using software SVM techniques, *2nd Workshop Cluster-Computing*, Published in: W. Rehm, and T. Ungerer (Eds.), Cluster-Computing, Proc. 2. Workshop, 25./26. March 1999, Universität Karlsruhe (CSR-99-02) – http://www.tu-chemnitz.de/informatik/RA/CC99/
3. C. Amza, A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R Rajamony, W. Yu and W. Zwaenepoel, TreadMarks: Shared Memory Computing on Networks of Workstations, *IEEE Computer*, Vol. 29, No. 2, 1996 – http://standards.ieee.org/regauth/oui/tutorials/sci.html
4. S. Zeisset, S. Tritscher, M. Mairandres, A New Approach to Distributed Memory Management in the Mach Microkernel, *USENIX 1996 Technical Conference*, San Diego, California, 1996
5. G. Cabillic, T. Priol, I. Puaut, MYOAN: an implementation of the KOAN shared virtual memory on the Intel paragon, *Technical Report RR-2258*, Institute National de Recherche en Informatique et en Automatique, 1994
6. E. Anderson, A. Mainwaring, J. Neefe, C. Yoshikawa, T. Anderson, D. Culler, D. Patterson, Experience with Two Implementations of Network RAM, *Internal Report, Computer Science Division*, University of California at Berkeley – http://http.cs.berkeley.edu/~eanders/projects/netram/
7. SCI Association – http://www.SCIzzL.com/
8. Myrinet – http://www.myri.com
9. GigaNet – http://www.giganet.com
10. VIA – http://www.viarch.org
11. Solaris MC – http://www.sunlabs.com/research/solaris-mc
12. Puma – http://www.cs.sandia.gov/puma
13. MOSIX – http://www.mosix.cs.huji.ac.il.
14. A. Barak, O. La'adan and A. Shiloh, Scalable Cluster Computing with MOSIX for LINUX (ftp), *Proc. Linux Expo '99*, pp. 95-100, Raleigh, N.C., May 1999.
15. J. Barkes, M.R. Barrios, F. Cougard, P.G. Crumley, D. Martin, H. Reddy, and T. Thitayanum, GPFS: A Parallel File System, IBM *International Technical Support Organization*, SG24-5165-00, April 1998
16. D. Kotz, Parallel I/O Archive, Dartmouth College, http://www.cs.dartmouth.edu/pario/
17. Yong E. Cho, Efficient Resource Utilization for Parallel I/O in Cluster Environments, *Ph.D thesis, University of Illinois* at Urbana-Champaign, 1999
18. *1998-1999 Operating System Function Review*, D.H. Brown Associates Inc. - http://www.rs6000.ibm.com/resource/consult/dhbrown/osfrev.html, 1998
19. *Linux – How Good Is It?* Executive Summary, D.H. Brown Associates Inc. - http://www.dhbrown.com, 1999
20. HPVM – http://www-csag.ucsd.edu/projects/hpvm.html

4. Single System Image (SSI)

Rajkumar Buyy, Monash University, Australia, and Toni Cortes, UPC, Barcelona, Spain.

4.1 Introduction

A Single System Image (SSI) is the property of a system that hides the heterogeneous and distributed nature of the available resources and presents them to users as a single unified computing resource. SSI can be enabled in numerous ways, these range from those provided by extended hardware through to various software mechanisms. SSI means that users have a globalised view of the resources available to them irrespective of the node to which they are physically associated. Furthermore, SSI can ensure that a system continues to operate after some failure (high availability) as well as ensuring that the system is evenly loaded and providing communal multiprocessing (resource management and scheduling).

SSI design goals for cluster-based systems are mainly focused on complete transparency of resource management, scalable performance, and system availability in supporting user applications [1, 2, 3, 4, 5]. A SSI can be defined as the illusion [1, 2], created by hardware or software, that presents a collection of resources as one, more powerful unified resource.

4.3 Services and Benefits

The key services of a single-system image cluster include the following:

- Single point of entry,
- Single user interface,
- Single process space,
- Single memory space,
- Single I/O space,
- Single file hierarchy,
- Single virtual networking,
- Single job-management system,
- Single point of management and control.

The most important benefits of SSI include the following [1]:

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

4.4 SSI Layers/Levels

The two important characteristics of SSI [1, 2] are:

1. Every SSI has a boundary,
2. SSI support can exist at different levels within a system—one able to be built on another.

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

- Hardware,

- Operating System (so called underware [4]),
- Middleware (the runtime subsystems),
- Application.

It is important to remember that a good SSI is usually obtained by a co-operation between all these levels as a lower level can simplify the implementation of a higher one.

4.4.1 Hardware Level

Systems such as Digital/Compaq Memory Channel [6] and hardware Distributed Shared Memory (DSM) [7] offer SSI at hardware level and allow the user to view a cluster as a shared-memory system.

4.4.2 Operating System Level

This can be achiever by either modifying/extending OS kernel (as followed in SCO UnixWare [4] and Sun Solaris-MC [8]) or building a layer (such as GLUnix [9]) that glues the OSs running on each node and makes them collaboratively present a single system.

4.4.2 Middleware Level

Middleware, a layer that resides between OS and applications, is one of the common mechanisms used to realise SSI in clusters. They include runtime and programming environments such as PVM [10] and job-management and scheduling systems such as CODINE [11] and Condor [12].

4.4.3 Application Level

Finally, applications can also support SSI. The application-level SSI is the highest and, in a sense, most important because this is what the end-user sees. At this level, multiple co-operative components of an application are presented to the user as a single application. For instance, a GUI based tool such as PARMON [13] offers a single window representing all the resources or services available. The Linux Virtual Server [14] is a scalable and highl availability server built on a cluster of real servers. The architecture of the cluster is transparent to end-users as all they see a single virtual server. All other cluster-aware scientific and commercial applications (developed using APIs such as MPI) hide the existence of multiple interconnected computers and co-operative software components, and present themselves as if running on a single system.

4.4.4 Pros and Cons of Each Level

Each level of SSI has its own pros and cons. The hardware-level SSI can offer the highest level of transparency, but due to its rigid architecture, it does not offer the flexibility required during the extension and enhancement of the system. The kernel-level approach offers full SSI to all users (application developers and end-users). However, kernel-level cluster technology is expensive to develop and maintain, as its market-share is/will be probably be limited as it is difficult to keep pace with technological innovations emerging into mass-market operating systems.

An application-level approach helps realise SSI partially and requires that each application be developed as SSI-aware separately. A key advantage of application-level SSI compared to the kernel-level is that it can be realised in stages and the user can benefit from it immediately. Whereas, in the kernel-level approach, unless all components are specifically developed to support SSI, it cannot be put in use or released to the market. Due to this, kernel-level approach appears as a risky and economically non-viable approach. The middleware approach is a compromise between the other two mechanisms used to provide SSI. In some cases, like in PVM, each application needs to be implemented using special APIs on a case-by-case basis. This means, there is a higher cost of implementation and maintenance, otherwise the user cannot get any benefit from the cluster. The arguments on the, so-called, underware versus middleware level of SSI are presented in [4].

4.6 Conclusions

SSI can greatly enhance the acceptability and usability of clusters by hiding the physical existence of multiple independent computers by presenting them as a single, unified resource. SSI can be realised either using hardware or software techniques, each of them have their own advantages and disadvantages. The middleware approach appears to offer an economy of scale compared to other approaches although it cannot offer full SSI like the OS approach. In any case, the designers of software (system or application) for clusters must always consider SSI (transparency) as one of their important design goals in addition to scalable performance and enhanced availability.

4.7 References

1. R. Buyya (editor), *High Performance Cluster Computing: Architectures and Systems*, Vol. 1, Prentice Hall PTR, NJ, USA, 1999.
2. G. F. Pfister, *In Search of Clusters*, Second Edition, Prentice Hall PTR, NJ, USA, 1998.
3. K. Hwang et. al., *Designing SSI Clusters with Hierarchical Check pointing and Single I/O Space*, IEEE Concurrency, vol.7(1), Jan.- March, 1999.
4. B. Walker and D. Steel, *Implementing a Full Single System Image UnixWare Cluster: Middleware vs. Underware*, In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas, USA, 1999.
5. Gerald Popek and Bruce J. Walker (Ed.), The Locus Distributed System Architecture, *MIT Press*, 1996
6. Memory Channel, http://www.digital.com/info/hpc/systems/symc.html
7. Distributed Shared Memory Systems, http://www.cs.umd.edu/~keleher/dsm.html
8. Solaris MC, http://www.sunlabs.com/research/solaris-mc/
9. GLUnix: A Global Layer Unix for a Network of Workstations, http://now.cs.berkeley.edu/Glunix/glunix.html
10. Parallel Virtual Machine (PVM), http://www.epm.ornl.gov/pvm/
11. CODINE Resource-Management System, http://www.genias.de/products/codine/
12. Condor, http://www.cs.wisc.edu/condor/
13. PARMON: A Portable and Scalable Monitoring System for Clusters, http://www.csse.monash.edu.au/~rajkumar/parmon/
14. Linux Virtual Server Project, http://www.LinuxVirtualServer.org/

5. Middleware

Mark Baker, University of Portsmouth, UK and Amy Apon, University of Arkansas, USA.

5.1 Introduction

Middleware is generally considered the layer of software sandwiched between the operating system and applications. Middleware provides various services required by an application to function correctly. Middleware has been around since the 1960's. More recently, it has re-emerged as a means of integrating software applications running in a heterogeneous environment. There is large overlap between the infrastructure that it provides a cluster with high-level Single System Image (SSI) services and those provided by the traditional view of middleware. A definition of SSI can be found in Section 3.2.2, whereas in this section of the paper we are concerned with middleware, and it can be described as the software that resides above the kernel and provides services to applications.

5.2. Background and Overview

To develop heterogeneous applications developers have three potential problems to overcome:

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

Middleware has the ability help the application developer overcome these problems. However, the services that middleware provides are not restricted to application development. Middleware also provides services for the management and administration of a heterogeneous system.

5.3. Technological Scope

In this section, we briefly describe a range of technologies that are being used as middleware.

5.3.1 Message-based Middleware

Message-based middleware is a technology that uses a common communications protocol to exchange data between applications. The communications protocol hides many of the low-level message passing primitives from the application developer. Message-based middleware software can either pass messages directly between applications, send messages via software that queues waiting messages, or use some combination of the two. Examples of this type of middleware are the three upper layers of the OSI model [1] – the session, presentation and applications layers. Other examples are DECmessageQ [2] from Digital, MQSeries [3] from IBM, and TopEnd [4] from NCR.

5.3.2 RPC-based Middleware

Although the client-server model provides a convenient way to structure a distributed application, the communications paradigm is problematic. Inter-process communication mechanisms serve four important functions [5]:

- They allow communications between separate processes over a computer network,

- They provide mechanisms against failure, and provides the means to cross administrative boundaries,
- They enforce clean and simple interfaces, thus providing a natural aid for the modular structure of large distributed applications
- They hide the distinction between local and remote communication, thus allowing static or dynamic reconfiguration

The overwhelming numbers of interactions between processes in a distributed system are remote operations. An important characteristic of the implementation of the client-server model in terms of RPC is that the code of the application remains the same if its procedures are distributed or not. This property is because the whole communication system is external to the application.

Client and server do not execute in the same address space, so there are no global variables and pointers cannot be used across the interface. Marshalling is the term used for transferring data structures used in RPC [6] from one address space to another. Marshalling is required, as two goals need to be achieved. The data must be serialized for transport in messages, and the data structures must be converted from the data representation on the sending end and reconstructed on the receiving end.

Middleware tools built over RPC include Network Information Services [7] (NIS) and Network File Services [8] (NFS). Both NIS and NFS were originally developed by Sun Microsystems, but versions of each of them have been released into the public domain. NIS is a network naming and administrative tool for smaller networks that allows users to access files or applications on the network with a single login name and password. NFS allows files and directories to be exported from the server computer to client computers on the network. With NFS, users can have the same view of the file system from any computer in the network. With NFS and NIS, users can login; access an application that resides on the file server, and save files into a home directory on the file server from any computer on the network.

5.3.3 CORBA

CORBA describes an architectural framework that specifies the mechanisms for processing distributed objects. The middleware that supports the remote execution of objects is called the *Object Request Broker* (ORB). CORBA is an international standard supported by more than 700 groups and managed by the *Object Management Group* [9] (OMG). The OMG is a non profit-making organization whose objective is to define and promote standards for object orientation in order to integrate applications based on existing technologies.

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

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

5.3.4 OLE/COM

The term COM has two meanings. It stands for *Component Object Model* [10], which forms the object model underpinning Object Linking and Embedding (OLE) version 2.0. It also stands for the *Common Object Model* after an agreement between Microsoft and Digital. The first version of OLE was designed to allow composite documents (text and images) to be handled. The second version of OLE introduced a highly generic object model whose use can be extended well beyond the handling of composite documents. OLE2 offers a set of interfaces (Object Oriented) that allows applications to intercommunicate. The first version of OLE2 required applications to run on the same machine. OLE2 was later extended to run in a distributed fashion on Windows NT and DEC UNIX platforms.

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

5.3.5 Internet Middleware

The TCP/IP suite of network protocols is the enabling technology for the Internet. However, a number of middleware-like technologies are built over TCP/IP. HyperText Transport Protocol (HTTP) allows text, graphics, audio, and video files to be mixed together and accessed through a web browser. The Common Gateway Interface (CGI) standard enables retrieved files to be executed as a program, which allows web pages to be created dynamically. For example, a CGI program can be used to incorporate user information into a database query, and then display the query results on a web page. CGI programs can be written in any programming language.

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

5.3.6 Java Technologies

Java Remote Method Invocation [11] (RMI) allows communications between two or more Java entities located in distinct Java Virtual Machines (JVM). Java RMI allows a Java applet or application access another remote object and invokes one of its methods. In order to access the remote object the calling application must obtain its address. This is obtained by access to a Registry where the object’s name was registered. The Registry acts as a name server for all objects using RMI. The Registry contains a table where each object is associated with a reference – this is the object’s interface and unique address of the object.

Jini [11, 12] from Sun Microsystems, is an attempt to resolve the inter-operability of different types of computer-based devices, given the rising importance of the network. These devices, which come from many different vendors, may need to interact over a network. The network itself is dynamic - devices and services will need to be added and removed regularly. Jini provides mechanisms to enable the addition, removal, and discovery of devices and services attached to network. Jini also provides a programming model that is meant to make it easier for programmers to enable devices to interact.

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

Other Java Technologies

Sun Microsystems has produced a plethora of Java-based technologies that can be considered as middleware [14]. These technologies range from the Java Development Kit (JDK) product family that consists of the essential tools and APIs for all developers writing in the Java programming language through to APIs such as for telephony (JTAPI), database connectivity (JDBC), 2D and 3D graphics, security as well as electronic commerce. These technologies enable Java to interoperate with many other devices, technologies, and software standards.

5.3.7 Cluster Management Software

Cluster Management Software (CMS) is designed to administer and manage application jobs submitted to workstation clusters. The job can be a parallel or sequential application that needs to run interactively or in the background. This software encompasses the traditional batch and queuing systems. CMS can be used to help manage clusters in a variety of ways:

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

CMS software is widely available in both commercial and public domain offerings. There are several comprehensive reviews of their functionality and usefulness [15, 16, 17].

5.4 How can TFCC help?

The TFCC will act as a focal point for research and development in academia and industry for cluster-oriented middleware. In this area the TFCC will:

- Act as a focal point for R&D in cluster-oriented middleware software.
- Provide a repository of freely available educational materials.
- Organize and sponsor conferences, colloquiums, and workshops.
- Promote the creation of resources such as books, bibliographies and web pages.
- Provide pointers to commercial and freely available software.
- Collaborate with organizations and groups involved in the development of cluster-oriented middleware.
- Collaborate with groups involved in the assessing and reviewing middleware products.
- Facilitate the interaction of commercial, industrial and academic research community.

5.5. Conclusions

The purpose of middleware is to provide services to applications running in distributed heterogeneous environments. The choice of which middleware may best meet an organization's needs is difficult as the technology is still being developed and may be some time before it reaches maturity. The risk of choosing one solution over another can be reduced if [18]:

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

Traditional software for providing middleware services, such as DCE, is being replaced by a new generation of object-oriented software such as CORBA/COM and Java. Certainly, CORBA seems to have had a new lease on life since the appearance of Java in the mid-1990's. Even CORBA and COM are being developed to be inter-operable with Java. Jini, Sun Microsystems's latest distributed environment, has wide possibilities to subsume all the other systems. The key question about what system a particular organization should adopt is no longer as it was a few years ago. Today the key question is, "is the middleware that I chose to adopt interoperable with the other current and emerging standards in middleware?"

5.5 References

1. International Organization for Standardization, http://www.iso.ch/
2. DECmesssageQ Documentation – http://www.digital.com/info/decmessageg
3. MQseries – http://www.software.ibm.com/ts/
4. TopEnd – http://www.ncr.com/
5. Mullendar. S. (Ed), Distributed Systems, 2nd Edition, *Adison-Wesley*, 1993, ISDN 0-201-62427-3
6. The Open Group, http://www.opengroup.org/
7. Using Name Services, *Sun Microsystems*, http://www.sun.com/smcc/solaris-migration/docs/transition-guide_2.5/nis.html
8. The NFS Distributed File Service, *NFS White Paper, Sun Microsystems*, March, 1995, http://www.sun.com/software/white-papers/wp-nfs/
9. Object Management Group, http://www.omg.org/
10. Microsoft – http://www.microsoft.com/com/tech/com.asp
11. Java RMI – http://www.javasoft.com/products/jdk/rmi/
12. Jini – http://www.sun.com/jini/
13. W.K. Edwards, Core Jini, *The Sun Microsystems Press Java Series, Prentice Hall PTR*, 1999, ISBN 0-13-0114469-X
14. Javasoft – http://www.javasoft.com/products/
15. CMS Review – http://www.nhse.org/NHSEreview/96-1.html
16. J.S. Kaplan and M.L. Nelson, A Comparison of Queuing, Cluster and Distributed Computer System, *NASA Langley Research Center, Technical Report*, June 1994.
17. NASA – http://parallel.nas.nasa.gov/Parallel/JMS/
18. D. Serain, Middleware, *Practitioner Series, Springer-Verlag*, 1999, ISBN 1-852330012

6. Parallel I/O

Erich Schikuta, University of Vienna, Austria and Helmut Wanek, University of Vienna, Austria

6.1 Introduction

In the last few years applications in high performance computing shifted from being CPU-bound to be I/O bound. Thus, the performance cannot be simply scaled up by increasing the number of CPUs any more, but it is often necessary to increase the bandwidth of the I/O subsystem. This situation is commonly known as the I/O bottleneck in high performance computing. The result was a strong research stimulus on parallel I/O topics. However, the focus was on massive parallel processor (MPP) systems, mainly neglecting workstation clusters. Besides the cumulative processing power, a cluster system provides a large data storage capacity. Usually each workstation has at least one attached disk, which is accessible from the system. Using the network interconnect of the cluster, these disks can build a huge common storage medium.

6.2 Background and Overview

Galbreath et al. [6] grouped the I/O needs of typical supercomputing applications into six categories: input, debugging, scratch files, checkpoint/restart, output, and accessing out-of-core structures. Generally, all parallel I/O methods and techniques aim to accomplish the following:

- Maximize the use of available parallel I/O devices to increase the bandwidth.
- Minimize the number of disk read and write operations per device. It is advantageous to read the requested data in a few larger chunks instead of many small ones. This avoids disk specific latency time for moving the read/write arm and waiting for the positioning of the spinning disk.
- Minimize the number of I/O specific messages between processes to avoid unnecessary costly communication.
- Maximize the hit ratio (the ratio between accessed data to requested data), to avoid unnecessary data accesses. It is a common technique, known as data sieving, to read a large sequential block of the disk at once into main memory and then to extract smaller data blocks according to the application requests.

Three different groups of methods in the parallel I/O execution framework are distinguished: application level, I/O level, and access anticipation methods (see [20]).

Application level methods try to organize the main memory objects mapping the disk space (e.g., buffer) to make disk accesses efficient. Therefore, these methods are also known as buffering algorithms. Commonly these methods are realized by runtime libraries, which are linked to the application programs. Thus, the application program performs the data accesses itself without the need for dedicated I/O server programs. Examples for this group are the Two-Phase method [2], the Jovian framework [1], and the Extended Two-Phase method [16].

The I/O level methods try to reorganize the disk access requests of the application programs to achieve better performance. This is done by independent I/O node servers, which collect the requests and perform the accesses. Therefore, the disk requests (of the application) are separated from the disk accesses (of the I/O server). A typical representative of this group is the Disk-directed I/O method [8].

Extending the I/O framework into the time dimension delivers a third group of parallel I/O methods: access anticipation methods. This group can be seen as an extension to data prefetching. These methods anticipate data access patterns that are drawn by hints from the code advance to its execution. Hints can be placed on purpose by the programmer into the code or can be delivered automatically by appropriate tools (e.g., compiler). Examples for this group are informed prefetching [12], the PANDA project [3] or the Two-phase data administration [13].

6.3 Technological Scope

The most eminent drawback of cluster systems to MPPs is the latency of the interconnect network. This is a severe problem for computational-bound applications. The bus is a major system bottleneck because of the increased load placed on it from network communication. However, research is on the way to overcome this problem by applying new bus technologies (Giganet, Myrinet) or by using multiple I/O buses in parallel. Thus, we see the case to distinguish between parallel I/O in MPPs and parallel I/O in cluster systems, which we will call specifically cluster IO. However many MPPs are in fact cluster systems consisting of independent nodes communicating by a high-speed interconnect, which is mostly proprietary, custom-made and specifically tuned for the hardware architecture. In contrary most conventional clusters are connected by an off-the-shelf interconnect, which shows all known disadvantages reducing the connection bandwidth.

6.3.1 Cluster I/O versus Parallel I/O

The specific difference of cluster I/O to parallel I/O lies in the intensified interconnects latency and/or in the heterogeneity found in cluster systems. Interconnect Latency. For I/O intensive applications the problem of the interconnect latency is not as severe as for computational bound applications. It is shown [14] that with appropriate and affordable hardware support, the network bottleneck of cluster systems can be turned again into the known I/O bottleneck. Thus, the already developed methods and paradigms to reduce this I/O bottleneck on MPP systems can similarly be used on cluster systems. This leads to the conclusions that a cluster can be a suitable platform for I/O based applications.

Heterogeneity. A heterogeneous environment leads to the following issues for cluster IO.

- Hardware Architecture: Each node can show a unique hardware configuration, i.e. different types and numbers of disks, I/O bus and controller characteristics, etc. This necessitates complex analytical models to predict the performance behavior and therefore complicates the I/O optimization process, or makes it even impossible.
- Data Representation: Non-uniform operating system landscapes of clusters can result obviously in different physical representation of the I/O data. This is not a problem so far if data is stored without semantics, but modern applications use this property (e.g. database tasks). Thus, respective conversion routines and/or a standardized I/O formats have to be implemented.
- Workload: Clusters support a multi-user (multi-application) environment naturally. This leads to the situation that the overall distribution of the workload of a cluster is hard to predict and any static approach to I/O planning is useless. The I/O subsystem has to shape dynamically to the workload of the system, which needs respective analytical models.

6.3.2 Features

The parallel I/O problem. In spite of the developed techniques for parallel I/O the employment in practice shows difficulties. This is based on the situation that most of the available I/O systems are difficult to apply. Mostly it is very hard to foresee the problem specific I/O behavior, the I/O hardware characteristics and thus their impact on the application's performance. Another obstacle is the often clumsy and hard to use interface of available I/O systems. For example, MPI-IO is powerful but not easy to apply successfully.

A further problem is the static behavior of most of the available I/O systems. Especially in clusters the system behavior is highly dynamic and not as easily to predict as in MPPs. This leads to the situation that in practice most parallel applications omit parallel I/O operations.

To overcome this parallel I/O problem new approaches to parallel I/O are necessary. A way to the solution of this problem is to free the application from the parallel I/O burden. This can be reached by a data engineering approach known from database systems, by abstraction and transparency.

New parallel applications with novel requisitions on the I/O system generate novel requirements for parallel IO. An example is On-Line Analytical Processing (OLAP), which permits decision support systems operating on huge data sets. This necessitates the support of the semantic interpretation of the stored data. In our opinion to allow full flexibility for the programmer and the administration methods, we propose a tier model with three independent layers in the I/O architecture (similar to the three-level-architecture of database systems):

- Problem tier – Defines the problem specific data distribution among the cooperating parallel processes.
- File tier – Provides a composed (canonical) view of the persistently stored data in the system.
- Data tier – Defines the physical data distribution among the available disks.

These layers are separated conceptually from each other, providing mapping functions between these tiers. Logical data independence exists between the problem and the file tier, and physical data independence exists between the file and data tier analogous to the notation in database systems. Transparency can be reached by automatic optimization of I/O operations, independent from the programmer. The speed at which a specific I/O operation can be performed is determined by a number of parameters, which unfortunately are interdependent.

The most crucial parameters are

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

These parameters are essentially the same for clusters as for traditional supercomputers like MPPs. The (possible) heterogeneity in cluster systems however tends to increase the number of parameters dramatically. Latency and transfer rates of different disks may differ considerably. Further, the speed of communication may vary dependent on which nodes are communicating.

Many of the parameter values (like latencies and transfer rates) are strongly influenced by the underlying hardware. Nonetheless, these parameters are not constant but can change significantly if other parameters are slightly modified. Given the huge number of possible parameter combinations, finding a near optimal I/O strategy for a specific program on a given hardware constellation turns out to be a very hard task even for a human specialist. So in order to gain broad acceptance by the users (application programmers) a parallel I/O system has not only to offer a simple API but must also deliver very high I/O throughput with little or no user intervention. The I/O system should therefore automatically optimize I/O operations based on information about the underlying hardware and the client application. Appropriate hardware information can be gained by benchmarking the system. Information about the client application can be supplied by the compiler (static) and/or it can be gathered during runtime by essentially logging and analyzing the I/O calls issued by the application (dynamic).

The need for automatic optimizations implies the urge for a model that sufficiently describes the interdependencies of all the above parameters in order to be able to evaluate different choices of parameters with respect to I/O through-put. There exist already several models ([17], [18],[19]) but they all focus on special aspects (like for example checkpointing) and/or only use a subset of the parameters. In addition, some models developed in other fields (like operating system design) may proof valuable here. Because different cache levels in memory correspond somewhat to different distances (local, remote intranet, internet) in cluster environments, an appropriate reinterpretation of caching strategies may proof valuable here. Nevertheless, currently none of these models seems general enough to allow for automatic optimization of arbitrary I/O operations.

There is however strong reason to believe, that even if such a model can be found the optimization itself still will be demanding. Due to the strong (mostly not linear and sometimes non continuous) interdependencies of parameters it is very unlikely that optimal buffer and block sizes, distributions, caching policies and schedules can be simply deduced from hardware parameters like latencies and transfer rates of disks and network interconnects. A simple formula seems not to exist (nor has one be found for any of the more specialized models above). The optimization strategy therefore is to generate different parameter sets and to use the model to calculate the I/O performance that can be achieved by these different parameter settings. Clearly, the huge number of possible parameter sets inhibits an exhaustive search. So smart algorithms are needed that only generate parameter sets that seem promising in respect with I/O performance and select the best one based on the model evaluation. These algorithms should produce the optimal (or a near optimal) result and they should not take longer to run than the time that can be gained by the achieved.

6.4 I/O optimization.

AI algorithms that find near optimum constellations in large search spaces have already been successfully used for partial solutions of the problem (Simulated annealing, genetic algorithms). Artificial neural networks have been used to spot regular patterns in I/O accesses.

Finally the I/O system does not only have to find the static optimal solution for a given series of I/O requests, it also must be able to dynamically adapt to a changing environment (contention caused by additional tasks, failure of a network connection or a disk). This has to be done autonomously (the system detects the changes itself) and transparent to the client applications. Reading and writing raw data may be sufficient for scratch files and input/output of number-crunching applications. However, as soon as data is stored persistently a need for additional information arises. The date and time of file creation and the latest update or maybe even a list of updates are of importance in almost all cases. Furthermore, it often is of interest what kind of data (Field names, data types) is stored in a specific file.

Storing this semantic information along with the file allows different and independently developed applications to easily share data by operating on common files, provided the semantic information can be interpreted correctly by all of the participating programs. XML obviously is a promising standard for storing semantic information, which can be shared by various applications. OLAP and data warehousing applications could also use the semantic information to integrate the data stored distributed across the cluster into some sort of a cluster wide distributed and parallel database.

6.5 Conclusion

The need for the storage of huge data sets is apparent, which is shown by the doubling in sales of storage systems each year. The need for support of fast and efficient access of this data is even more urgent, due to the ascendance of very new application domains, as in the area of multimedia, knowledge engineering, and large-scale scientific computing.

Cluster systems can provide a viable platform for these I/O intensive tasks. However, the lessons learned from parallel I/O of MPPs are helpful but not comprehensive and specific enough to be mapped directly to clusters. Thus, cluster I/O will need new approaches and models to cope with the specific characteristics. Further new application domains arise heavily simulated from the database community, which necessitates the research focusing new directions. Thus, we believe that the cluster I/O topic will be a stimulating area for research in the coming years.

6.6 References

1. Robert Bennett, Kelvin Bryant, Alan Sussman, Raja Das, and Joel Saltz. Jovian: A Framework for Optimizing Parallel I/O. In Scalable Parallel Libraries Conference, IEEE Computer Society Press, MS, October 1994.
2. Rajesh Bordawekar, Juan Miguel del Rosario, and Alok Choudhary. Design and Evaluation of Primitives for Parallel I/O. In Supercomputing '93, IEEE Computer Society Press, pages 452{461, Portland, OR, 1993.
3. Y. Chen, M. Winslett, S. Kuo, Y. Cho, M. Subramaniam, and K. E. Seamons. Performance Modeling for the Panda Array I/O Library. In Supercomputing '96, ACM Press and IEEE Computer Society Press, November 1996.
4. Alok Choudhary and David Kotz. Large-Scale File Systems with the Flexibility of Databases. ACM Computing Surveys, vol. 28A(4), December 1996.
5. [Dror G. Feitelson, Peter F. Corbett, Yarson Hsu, and Jean-Pierre Prost. Parallel I/O Systems and Interfaces for Parallel Computers. In Topics in Modern Operating Systems, IEEE Computer Society Press, 1997.
6. N. Galbreath, W. Gropp, and D. Levine. Applications-Driven Parallel I/O. In Supercomputing '93, IEEE Computer Society Press, Portland, Oregon, 1993.
7. G. A. Gibson, J. S. Vitter, and J. Wilkes. Strategic Directions in Storage I/O Issues in Large-Scale Computing. ACM Computing Surveys, vol. 28(4), December 1996.
8. David Kotz. Disk-Directed I/O for MIMD Multiprocessors. ACM Transactions on Computer Systems, vol. 15(1), February 1997.

9. MPI-IO: A Parallel File I/O Interface for MPI. The MPI-IO Committee, April 1996.
10. Bill Nitzberg and Samuel A. Fineberg. Parallel I/O on Highly Parallel Systems, Supercomputing '95 Tutorial M6 Notes. Technical Report NAS-95-022, NASA Ames Research Center, December 1995.
11. D. A. Patterson and K. L. Hennessy. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers Inc., 1990.
12. R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed Prefetching and Caching. In Fifteenth ACM Symposium on Operating Systems Principles, ACM Press, pages 79{95, CO, December 1995.
13. Erich Schikuta, Thomas Fuerle, and Helmut Wanek. ViPIOS: The Vienna Parallel Input/Output System. In Euro-Par'98, Southampton, England, Springer-Verlag, September 1998.
14. Thomas Sterling, Donald J. Becker, Daniel Savarese, Michael R. Berry, and Chance Res. Achieving a Balanced Low-cost Architecture for Mass Storage Management Through Multiple Fast Ethernet Channels on the Beowulf Parallel Workstation. In International Parallel Processing Symposium, 1996.
15. Heinz Stockinger. Classification of Parallel Input/Output Products. In Parallel And Distributed Processing Techniques and Applications Conference, July 1998.
16. R. Thakur and A. Choudhary. An Extended Two-Phase Method for Accessing Sections of Out-of-Core Arrays. Scientific Programming, vol. 5(4), 1996.
17. Ying Chen. Automatic Parallel I/O Performance Optimization in Panda Ph.D. thesis, University of Illinois at Urbana-Champaign, 1998
18. Yong E. Cho. Efficient Resource Utilization for Parallel I/O in Cluster Environments Ph.D. thesis, University of Illinois at Urbana-Champaign, 1999
19. Szu-Wen Kuo. Effcient Data Organization and Load Balancing on Parallel Disks Ph.D. thesis, University of Illinois at Urbana-Champaign, 1999
20. E. Schikuta and H. Stockinger Parallel I/O for Clusters: Methodologies and Systems In High Performance Cluster Computing: Architectures and Systems, Ed. Rajkumar Buyya, Prentice Hall, 1999

7. High Availability

Ira Pramanick, Sun Microsystems, USA

7.1 Introduction

The importance of a system being highly available not only for "traditional mission critical applications", but an increasing need for almost all commercial (and many academic) applications and systems to provide users with minimal downtime. This even extends, for example, to customers demanding minimal downtime for regular system maintenance/upgrades.

We consider the follo9wing:

- *Continuous availability* – This implies non-stop service, representing an ideal state.
- *Fault tolerance* – This provides very high levels of availability by employing redundant, custom hardware for the various components of a computer system.
- *High availability* [1, 2] - This provides a cheaper alternative to fault tolerance by providing software solutions using off-the-shelf hardware components. Highly available solutions typically survive single points of failure in a system.

7.2 Background and Overview

High Availability (HA) and clustering are very closely related. A typical HA solution consists of software modules/enablers which run on a cluster consisting of two or more off-the-shelf computer nodes, that together form the HA system. The failure of a node results in all HA applications that were running on that node to be migrated/failed over to another node in the system. Similarly, failure of other components in the system may result in appropriate failovers of that resource. Current HA solutions range from clusters consisting of two to eight nodes, although there are two or three HA products that are advertised to work on much larger clusters. Since the target market for most HA applications involves nodes that are typically servers, the need for much larger than 32 node clusters for HA is not deemed very important, at least not currently. This has implications about the scalability of HA solutions; rather the (lack of) significance of emphasis on the scalability of algorithms employed for the infrastructure of HA systems.

Broadly speaking, an HA solution consists of:

(i) An HA infrastructure that has software components that enable the cluster to appear as a single system and that monitor the cluster nodes, and

(ii) HA applications that use the facilities provided by the infrastructure to provide seamless (in the wake of system component failures) services to users.

There are two kinds of HA framework: those that sit on top of the operating system, and those that are integrated into the base operating system, with both having their various pros and cons.

Both from a user's perspective and that of HA applications that sit on top of an HA framework, the single system image (SSI) capability of an HA solution is critical to its ease-of-use. This is a huge area of research, not only for high availability, but also for cluster computing in general. However, from the HA perspective, the most important elements of SSI include global file systems, global devices and global networking in the cluster. Also, SSI ties in very closely to cluster management mentioned below.

(Ease of) Cluster management is extremely important in general and for HA in particular, since HA products are often difficult to understand from a user's perspective, who would rather just treat their HA cluster as a “system” that acts as a server or server group, and provides seamless access to services in the wake of different types of failures. They do not want to be burdened with details of how the HA software works, and the success of an HA solution is largely dependent on how easy it is to administer and maintain. A graphical user interface (GUI) for cluster administration, management and monitoring (and sometimes even controlling) is a good starting point, but the real enabler here is SSI, which allows administration tools, graphical or not, to present a unified picture of the cluster to the user rather than an awkward glue of the cluster components. All this is particularly important for HA, where incorrect cluster management and/or user error should minimally affect system downtime.

HA applications: There is a plethora of these today, but the most common ones on any HA platforms are the various databases (Oracle, Informix, Sybase), file systems, web servers, and mail servers. Many of these comes pre-packaged with HA products and others can be written on top of the HA infrastructure by application vendors.

HA Application Programming Interface: Applications use the HA-API to convert themselves into highly available versions. Each HA-API has its own specific functions, but all the flavours share methods to start an application, stop an application, to monitor and application-specific resource set, to mention a few.

Some notable HA research projects are:

(i) The High Availability Linux Project [3],
(ii) The Horus Project [4],
(iii) The ISIS Project [5],
(iv) The Linux Virtual Server Project [6],
(v) The Solaris-MC Project [7].

Here is a list of some commercial HA solutions in alphabetical order.

(i) Data General's DG UX Clusters [8],
(ii) Digital's TruClusters [9],
(iii) HP's MC/ServiceGuard [10],
(iv) IBM's HACMP [11],
(v) Microsoft Cluster Services [12],
(vi) NCR's LifeKeeper [13],
(vii) Novell's HA solutions [14],
(viii) RSi's RSF [15],
(ix) Sequent's ptx/CLUSTERS [16],
(x) SGI's IRIS FailSafe [17],
(xi) Stratus's CA Solution [18],
(xii) Sun Clusters [19],
(xiii) TurboCluster Server [20],
(xiv) Veritas FirstWatch [21].

7. 3 Technological Scope

HA infrastructure - pros and cons of it sitting on top of the OS versus it being a part of the OS. There are several advantages of integrating an HA solution into the underlying OS, such as better performance and ease of use which contribute to a better HA solution. The main advantages of an HA solution that sits on top of an OS include independence of the solution with respect to the underlying OS leading to portability across various OSs, and often a smaller development cycle.

Importance of Single System Image (SSI) in HA - many enablers of SSI, such as cluster file system, cluster-wide networking and cluster-wide device naming, are in fact almost necessary components of a good HA solution.

Scalability of HA solutions - since typical HA cluster nodes tend to be “fat” server nodes, currently there does not seem to be a compelling reason to support very large (greater than 32 nodes) HA clusters. I.e., algorithms employed for the HA infrastructure do not necessarily scale. As HA becomes increasingly popular and begins to be used by many different kinds of applications, a need for supporting larger clusters may become important. This may happen for instance if highly scalable HPC solutions want HA incorporated into their HPC machines.

Programming with HA API - as HA becomes increasingly important resulting in a growing number of applications needing to become highly available, the traditional “how difficult is it to program” issue that has been a great source of extensive research in parallel processing. Distributed computing and clustering in the high performance computing arena, begins to loom up in the HA world also.

Cluster components - these must be allowed to be heterogeneous since that is one of the advantages of cluster computing, and in fact is almost always a necessity for a customer. In addition to the usual algorithmic issues that arise in heterogeneous systems, this also brings in discussion of features such as rolling upgrades.

7.4 Conclusions

- HA is becoming increasingly important since everybody wants minimal downtime.
- HA solutions typically span clusters of sizes ranging from 2 to 8 server nodes today.
- There are many different HA products out there, each with several HA applications enabled for them in a pre-packaged form, and others being supported by vendors.
- The research and development opportunities in HA are large, and this arena has some ways to go before it attains maturity. This is especially true when one considers its relationship to high performance computing.

7.5 References

1. The D.H. Brown Reports, http://www.dhbrown.com
2. The IMEX High Availability Report, http://www.imexresearch.com
3. High Availability Linux, http://linux-ha.org
4. The Horus Project, http://www.cs.cornell.edu/Info/Projects/Horus/
5. The ISIS Project, http://www.cs.cornell.edu/Info/Projects/ISIS/
6. Linux Virtual Server Project, http://www.webpusher.com/wensong/
7. Solaris-MC, http://www.sunlabs.com/research/solaris-mc/
8. TruClusters, http://www.digital.com/info/SP4417/SP4417HM.HTM
9. DG UX Clusters, http://www.dg.com
10. MC/ServiceGuard, http://www.hp.com/ssg/eservicesontap/index2.html
11. IBM High Availability Services, http://www.as.ibm.com/asus/highavail2.html
12. Microsoft Cluster Services, http://www.microsoft.com/ntserver/
13. LifeKeeper, http://www3.ncr.com/support/nt/lknt/lkntsol.htm
14. Novell High Availability Server, http://www.novell.com/whitepapers/ha/hawp.html

15. Resident Server Facility, http://www.rsi.co.uk/
16. ptx/CLUSTERS, http://www.sequent.com/products/software/layered/ptx_clusters.html
17. IRIS FailSafe, http://www.sgi.com/Technology/FailSafe.html
18. Stratus Continuum Series, http://www.stratus.com/docs/service/products/index.htm
19. Sun Clustesr, http://www.sun.com/clusters
20. TurboCluster Server, http://community.turbolinux.com/cluster/
21. FirstWatch, http://www.veritas.com

8. Numerical Libraries and Tools for Scalable Parallel Cluster Computing

Shirley Browne, University of Tennessee, USA, Jack Dongarra, University of Tennessee, and ORNL, USA and Anne Trefethen, Numerical Algorithms Group Ltd, UK

8.1 Introduction

For cluster computing to be effective within the scientific community it is essential that there are numerical libraries and programming tools available to application developers. Cluster computing may mean a cluster of heterogeneous components or hybrid architecture with some SMP nodes. This is clear at the high end, for example the latest IBM SP architecture, as well as in clusters of PC-based workstations. However, these systems may present very different software environments on which to build libraries and applications, and indeed require a new level of flexibility in the algorithms if they are to achieve an adequate level of performance. We will consider here the libraries and software tools that are already available and offer directions that might be taken in the light of cluster computing.

8.2 Background and overview

There have been many advances and developments in the creation of parallel code and tools for distributed memory machines and likewise for SMP-based parallelism. In most cases the parallel, MPI-based libraries and tools will operate on cluster systems, but they may not achieve an acceptable level of efficiency or effectiveness on clusters that comprise SMP nodes. Little software exists that offers the mixed-mode parallelism of distributed SMPs. It is worth considering the wealth of software available for distributed memory machines; as for many cases this may be entirely suitable. Beyond that we need to consider how to create more effective libraries and tools for the hybrid clusters.

The underlying technology on which the distributed memory machines are programmed is that of MPI [11]. MPI provides the communication layer of the library or package, which may, or may not be revealed, to the user. The large number of implementations of MPI ensures portability of codes across platforms and in general, the use of MPI-based software on clusters. The emerging standard of OpenMP [2] is providing a portable base for the development of libraries for shared memory machines. Although most cluster environments do not support this paradigm globally across the cluster, it is still an essential tool for clusters that may have SMP nodes.

8.3 Numerical Libraries

The last few years have seen continued architectural change in high performance computing. The upside of this development is the continued steep growth in peak performance, but the downside is the difficulty in producing software that is easy to use, efficient, or even correct. We will review a list of these challenges.

A major recent architectural innovation is clusters of shared-memory multiprocessors referred to as a Constellation. (This is to avoid confusion with the term Cluster, which usually is used in the context of a group of PCs connected through a switched network.) These are the architectures of the ASCI machines, and promise to be the fastest general-purpose machines available for the next few years, in accordance with the industrial trend to invest most heavily in large market sectors and use the same building blocks to service the smaller high-end market.

It is the depth of the memory hierarchy with its different access primitives and costs at each level that makes Constellations more challenging to design and use effectively than their SMP and MPP predecessors. Ideally, a programmer should be able to produce high performance parallel code on a Constellation using an abstract programming model that is not dependent on implementation details of the underlying machine. Since users may have different notions of what is acceptable high performance, we expect the layered programming model to allow the underlying machine to be exposed at varying levels of detail. This requires good communication libraries, data structure libraries, and numerical algorithms, which we propose to build.

Although Constellations will be the dominant architecture for high end computing, most programs will be from either SMP or MPP versions, and even new code developments will likely be done on smaller machines. Therefore, users need a uniform programming environment that can be used across uniprocessors, SMPs, MPPs, and Constellations. Currently, each type of machine has a completely different programming model: SMPs have dynamic thread libraries with communication through shared memory; MPPs have Single Program Multiple Data (SPMD) parallelism with message passing communication (e.g., MPI); Constellations typically have the union of these two models, requiring that the user write two different parallel programs for a single application, or else treat the machine as “flat”, with a corresponding performance loss. Well-designed libraries can hide some of these details, easing the user's transition from desktop to SMP to Constellations.

In addition to Constellations, architectures that are likely to be important are distributed networks and IRAM. IRAM stand for Intelligent-RAM, and is also called PIM, or Processor-in-Memory. IRAM consists of a chip containing both processors and a substantial memory, and is a foreseeable architectural change as devices get smaller and devices more integrated. Distributed Networks, which can be and are used now, have very slow and even unreliable access to remote memories, whereas IRAM promises to significantly flatten the memory hierarchy, with the on-chip memory effectively acting like an enormous cache. This profusion of architectural targets makes software development for scientific computing challenging.

One common feature is to build libraries that are *parameterized* for the architectural features most influencing performance. We are used to dealing with cache sizes, latency and bandwidth in our use of performance modelling in previous work, but the complexity of Constellations and other architectures presents new challenges in this regard.

Another architecturally driven algorithmic opportunity arises on Pentium processors, which are not only widely used on desktops, but comprise the ASCI Red machine, and will be widely used in smaller clusters as inexpensive computing platforms. This ubiquity of Intel platforms leads us to ask how to exploit special features of the Intel architecture to do better high performance computing. In addition to excellent support for IEEE standard 754 floating arithmetic, the basic arithmetic is done to 80-bit precision rather than 64-bit. There are a variety of algorithms that perform more quickly and/or more accurately by using these features. These algorithms can be encapsulated within standard libraries, and so do not require user sophistication or even awareness for their use.

Another challenge from the proliferation of computing platforms is how to get high performance from computational kernels like matrix-matrix multiplication, matrix-vector multiplication, FFTs, etc. There are systems such as ATLAS, PhiPac, and FFTW that use a sophisticated search algorithm to automatically find very good matrix-multiply kernels for RISC workstations with C compilers that do good register allocation and basic instruction scheduling; using this approach one can produced matrix multiply and FFT routines that are usually faster than the hand tuned codes from IBM, SGI and some other manufacturers[2].

8.4 Current State-of-the-art

Of the hundred or more parallel numerical packages available some provide a conventional library interface to routines written in C, Fortran or C++. Others provide more of a parallel environment for the application developer. At the moment few, if any, mix distributed and shared-memory parallelism.

Recent surveys on parallel numerical analysis software [3, 19, 20] include approximately 50 different libraries and packages for parallel architectures. All of these would be suitable for cluster computing although they may not be fully efficient for particular configurations. The software discussed in the report forms a subset of all parallel packages that are available either commercially or distributed on Netlib[3] or on the National HPCC Software Exchange, NHSE[4].

As one might expect, the majority of available packages in this area are in linear algebra. Both direct solvers and iterative are well represented. Direct solver packages include ScaLAPACK [4] and PLAPACK [5] both of which are based on the parallelization of LAPACK [5] but by different approaches.

Iterative solver packages include Aztec [7], Blocksolve [8] and also PSPARSLIB [9]. Each of these provides a selection of iterative solvers and preconditioners with MPI providing the underlying communications layer. Similarly there are packages and libraries for eigenvalue problems including PARPACK [10] and PeIGS [11]; the first based on Arnoldi iterations and the second on Cholesky decomposition; both using MPI.

Other areas covered by existing parallel libraries include optimization and PDE solvers. All of the libraries mentioned above are available from Netlib. Commercial products are provided by many of the machine vendors and NAG provides a commercial, supported, general Parallel Library [12] based on MPI and also an SMP library [13,14] based on OpenMP.

The libraries that have been mentioned so far all have a traditional library interface. One of the packages that offer parallel functionality in the setting of an environment is PETSc [15,16]. PETSc provides an object-based interface to C-coded algorithms. In this case the application developer does not need to be aware of the message-passing or underlying mechanisms. PETSc provides a set of tools for solving PDE problems including iterative methods and other underlying functionality. This requires that the user develop their application in the PETSc style rather than calling a particular routine from PETSc. This provides an object-oriented interface that of course is also the natural interface for C++ -based packages such as ISIS++ and ELLPACK [17].

It is clear that there is a wealth of software available for clusters; however, as noted above this is in general either developed for heterogeneous distributed memory architectures or

[2] See: http://www.netlib.org/atlas/ and http://www.fftw.org/

[3] Netlib – http://www.netlib.org

[4] NHSE – http://www.nhse.org

SMP machines. We still have some distance to travel before we can provide effective and efficient numerical software for the general cluster.

8.5 Future work

To achieve transparent cluster computing requires algorithms that can adapt to the appropriate configuration to match the cluster requirements. Recent studies show that there are benefits to be gained by rethinking the parallelization issues combining distributed and shared-memory models [18]. It is unlikely that libraries that provide this effective mixed-mode parallelism will be available for some time. It is more likely that libraries that are inherently designed for distributed memory but will be adapted have appropriate SMP-based kernels. This is likely to provide the natural progression to general numerical libraries for clustered computing.

8.6 Program Development and Analysis Tools

Rapid improvements in processor performance and multiprocessing capabilities of PC-based systems have led to widespread interest in the use of PC clusters for parallel computing. The two major operating systems that support multiprocessing on such systems are Linux and Windows NT. Scalable parallel computing on PC clusters requires the use of a message passing system such as MPI, although OpenMP and other forms of thread-based parallelism may also be used on SMP nodes. Programming languages of interest for scientific computing on PC clusters include Fortran, C, and C++. Appendix B presents a survey of available program development and analysis tools for PC cluster environments. These tools include compilers and pre-processors, MPI implementations, and debugging and performance analysis tools, as well as integrated development environments (IDEs) that combine these tools. Some IDEs are developed entirely by one vendor or research group, while others are designed to work together with third party tools, for example with different compilers and MPI implementations.

8.6 Conclusions

A good base of software is available to developers now, both publicly available packages and commercially supported packages. These may not in general provide the most effective software, however they do provide a solid base from which to work. It will be some time in the future before truly transparent, complete efficient numerical software is available for cluster computing. Likewise, effective programming development and analysis tools for cluster computing are becoming available but are still in early stages of development.

8.7 References

1. Snir, M., Otto, S. Huss-Lederman, S. Walker, D., and Dongarra, J., MPI: The Complete Reference, MIT Press, Boston, 1996.
2. Dagum, L., and Menon, R.,: OpenMP: An Industry-Standard API for Shared-Memory programming, in IEEE Computational Science & Engineering, Vol. 5, No. 1, January/March 1998.
3. Allan R.J, Hu Y.F., and Lockey P.: Parallel Application Software on High Performance Computers. Survey of Parallel Numerical Analysis Software, http://www.cse.clrc.ac.uk/Activity/HPCI

4. Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I, Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Walker, D., Whaley, R.C., ScaLAPACK Users' Guide, *SIAM*, Philadelphia, 1997 - http://www.netlib.org/scalapack
5. P. Alpatov, G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, R. van de Geijn, and Y.J. Wu, PLAPACK: Parallel Linear Algebra Libraries Design Overview, *SC97* - http://www.cs.utexas.edu/users/rvdg/plapack
6. Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenny, A., Ostrouchov, S., and Sorenson, D., Lapack Users' Guide, Release 2.0. SIAM, Philadelphia, 1995.
7. Hutchinson, S. A., Prevost, L. V., Tuminaro, R. S. and Shadid, J. N.: AZTEC Users' Guide: Version 2.0. Sandia National Labs, 1998 – http://www.cs.sandia.gov/CRF/aztec1.html
8. Jones, M.T., and Plassman, P.E., Blocksolve V1.1: Scalable library software for parallel solution of sparse linear systems. ANL Report 92/46, Argonne National Laboratory, December 1992 – http://www-unix.mcs.anl.gov/sumaa3d/BlockSolve
9. Saad, Y and Sosonkina, M.: Solution of distributed sparse linear systems using psparslib. In Applied Parallel Computing, Proc. *PARA'98,* pages 501-9. Springer Verlag, Lecture Notes in Computer Science, Vol. 1541, 1998 – http://www.cs.umn.edu/Research/arpa/p_sparslib/psp-abs.html
10. Lehoucq, R. B., Sorensen, D. C. and Yang, C., ARPACK Users' Guide: Solution of Large-scale Eigenvalue Problems with implicitly-restarted Arnoldi Methods. *SIAM*, Philadelphia, 1998 – http://www.caam.rice.edu/software/ARPACK
11. Elwood D., Fann, G., and Littlefield, R., Parallel Eigensolver System User Manual. *Batelle Pacific Northwest Laboratory*. (Available from anonymous@ftp://pnl.gov)
12. The NAG Parallel Library Manual, Release 2, *Numerical Algorithms Group Ltd*, Oxford (1997).
13. Salvini, S., Waniewski, J., Linear Algebra Subprograms on Shared Memory Computers: Beyond LAPACK. In J. Waniewski, J. Dongarra, K. Madsen, D. Olesen (eds.), Applied Parallel Computing Industrial Computation and Optimization, Third International Workshop, *PARA'96*, Lyngby, Denmark, August 18--21, 1996, Proceedings. *Springer-Verlag Lecture Notes in Computer Science*, Vol. 1184, 1996.
14. The SMP Library User Manual, Release 1, *Numerical Algorithms Group Ltd*, Oxford (1997).
15. Gropp W. and Smith B., Scalable, extensible, and portable numerical libraries. *Technical report*, Argonne National Laboratory.
16. Balay, S., Gropp W., Curfman McInnes, L. and B. Smith: PETSc 2.0 Users' Manual. Argonne National Laboratory, 1996. *Technical Report* ANL-95/11 revision 2.0.17.
17. Weerawarana, S., Houstis, E.N. Rice R.J., Catlin, A.C., Crabill, C.L. and Chui, C.C., Pdelab: an object-oriented framework for building problem solving environments for PDE-based applications. *Technical Report CSD-TR-94-021*, Purdue University, March 1994 - http://ziggurat.ca.sandia.gov/isis and http://www.cs.purdue.edu/research/cse/pellpack/pellpack.html
18. Salvini, S., Smith, B., and Greenfield, J., Towards Mixed Mode Parallelism on the New Model F50-based IBM SP System. *Albuquerque HPCC Report AHPCC98-003*.
19. Eijkhout, V, A Survey of Iterative Linear System Solver Packages, http://www.netlib.org/utk/papers/iterative-survey/
20. Dongarra, J.J., Freely Available Software For Linear Algebra On The Web, http://www.netlib.org/utk/people/JackDongarra/la-sw.html

9. Applications

David Bader, New Mexico, USA and Robert Pennington, NCSA, USA

9.1 Introduction

Cluster computing for applications scientists is changing dramatically with the advent of commodity high performance processors, low-latency/high bandwidth networks and software infrastructure and development tools to facilitate the use the cluster. The performance of an individual processor used in a high-end personal workstation rivals that of a processor in a high-end supercomputer, such as an SGI Origin and the performance of the commodity processors is improving rapidly. For example, MILC [1] executes at:

- 55 Mflop/s on an Intel 300 MHz Pentium II
- 105 Mflop/s on an Intel 550 MHz Pentium III Xeon and
- 165 Mflop/s on a single 250 MHz R10000 processor in an SGI O2000,

In the aggregate, the performance is even more impressive. The excellent performance of the interconnect network and related software is shown by comparing the same application running on a large cluster and a large SGI Origin. On 128 processors using MPI, MILC executes at:

- 4.1 Gflop/s on 128 Intel 300 MHz Pentium IIs
- 8.4 Gflop/s on 128 Intel 550 MHz Pentium III Xeons
- 9.6 Gflop/s on a 128 processor SGI O2000 with 250 MHz R10000 processors

The successful standardization of system-level interfaces and supporting libraries has removed many of the issues associated with moving to a new computational platform. High-performance clusters are one of the new platforms that are coming to the forefront in the computational arena. There are still a number of outstanding problems to be resolved for these systems to be as highly effective on applications as a supercomputer. For the current generation of clusters, the most significant areas for the applications scientists are performance of the I/O systems on a cluster system and compilation, debugging and performance monitoring tools for parallel applications.

9.2 Application Development Environment

It is essential that the applications scientist be able to move the code to new systems with a minimum amount of effort, have the same or similar tools and environment available to use on different systems. As recently as a few years ago, the lack of a canonical high-performance architectural paradigm meant that migrating to a new computer system, even from the same vendor, typically required redesigning applications to efficiently use the high-performance system. As a result, the application had to be redesigned with parallel algorithms and libraries that were optimized for each high-performance system. This process of porting, redesigning, optimizing, debugging, and analyzing, the application is generally prohibitively expensive, and certainly frustrating when the process is completed just in time to greet the next generation system.

The importance of supporting portable code that uses accepted standards, such as MPI [2], on clusters cannot be overstressed. The current generation of clusters are becoming useful to researchers and engineers precisely because of their direct support of these standards and libraries. Research software messaging systems and interfaces from vendors such as the

Virtual Interface Architecture [3] may provide the underlying protocol for a higher level message passing interface such as MPI but are not directly useful to the vast majority of applications scientists.

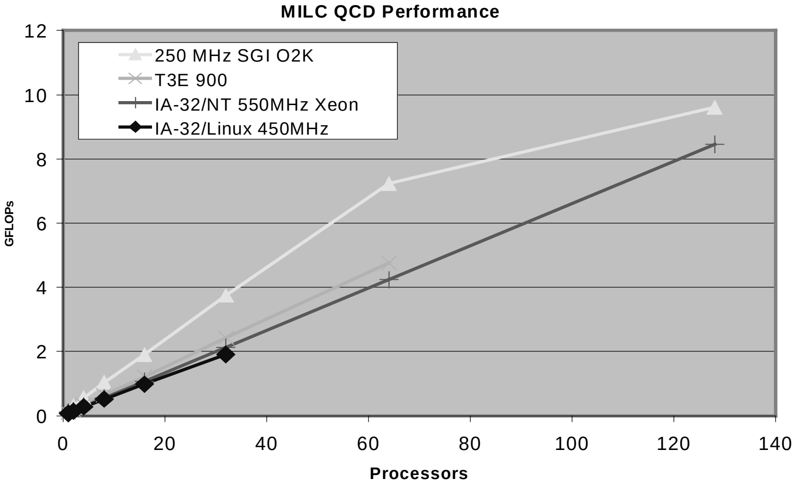
The development environment for clusters must be able to provide the tools that are currently available on supercomputing systems. The current sets of applications running on supercomputing clusters are generally developed, debugged and tested on supercomputing systems. For first generation applications that may be developed for large scale clusters, tools such as debuggers, profilers and trace utilities that work identically or very similarly to those available on supercomputers need to be available to the applications developers. Common tools such as shell, make, tar, and scripting languages also need to be included in the development environment.

9.3 Application Performance

To illustrate the competitive performance of clusters, we show an application, MILC, that has been run on a large-scale cluster in the Alliance, the NT Supercluster [4] at the NCSA. This application is also being used on the Roadrunner Supercluster [5] at the University of New Mexico. In addition, one of the major performance issues confronting clusters is exemplified in the data for the ARPI-3D code, which shows timings for computations, message passing, initialization I/O and I/O at the completion of the application.

9.3.1 MILC

Kostas Orginos and Doug Toussaint did these benchmarks from the University of Arizona. They are for the conjugate gradient calculation of quark propagators in quantum chromodynamics with Kogut-Susskind quarks. The conjugate gradient typically takes 90% of the time in full QCD calculations, so it is the reasonable thing to measure. These are with the simplest version of the action, which is fairly standard. In each case, they have chosen the lattice size so that there are 4096 lattice sites per processor. This is in the range typically used in production runs. For the simplest variants of the code, this works out to about 4MBytes of data per process.



9.3.2 ARPI 3D

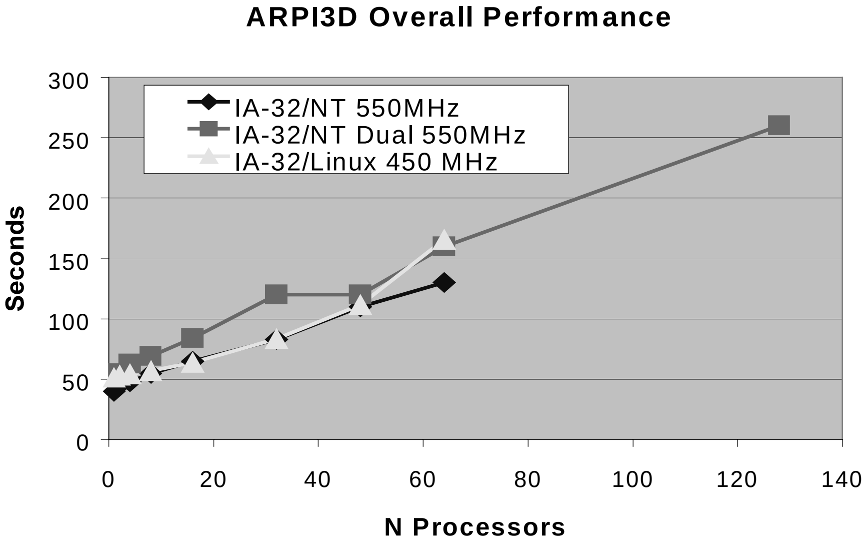
These graphs show the performance of the ARPI 3D weather research model on two large scale clusters, the NCSA NT Supercluster and the UNM Roadrunner Linux Supercluster. These particular tests use a 3D numerical weather prediction model to simulate the rise of a moist warm bubble in a standard atmosphere. The simulations were run to 50 seconds with a 35x35x39 grid, in which time each processor wrote three 2.107 MBYTES/S files to a centralized file system. The data was provided by Dan Weber at the University of Oklahoma from his runs on the two systems.

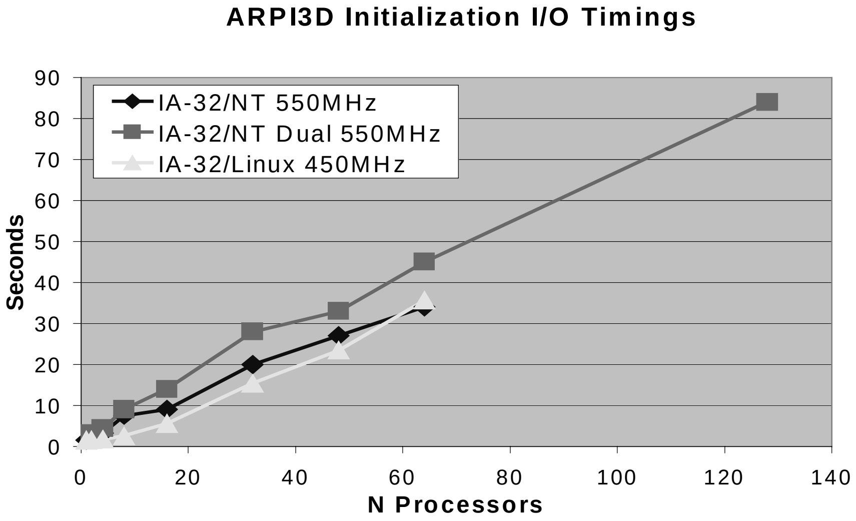
The NT Supercluster data was taken in two different modes. In the first mode, only one processor in the dual processor systems was being used by the application and the second processor was idle. In the second mode, both processors in the dual processor systems were being used by the application. The dual processor runs show approximately a 20% performance degradation compared to runs using the same number of processors in separate systems. The Roadrunner data is available for only the one processor per dual processor system case.

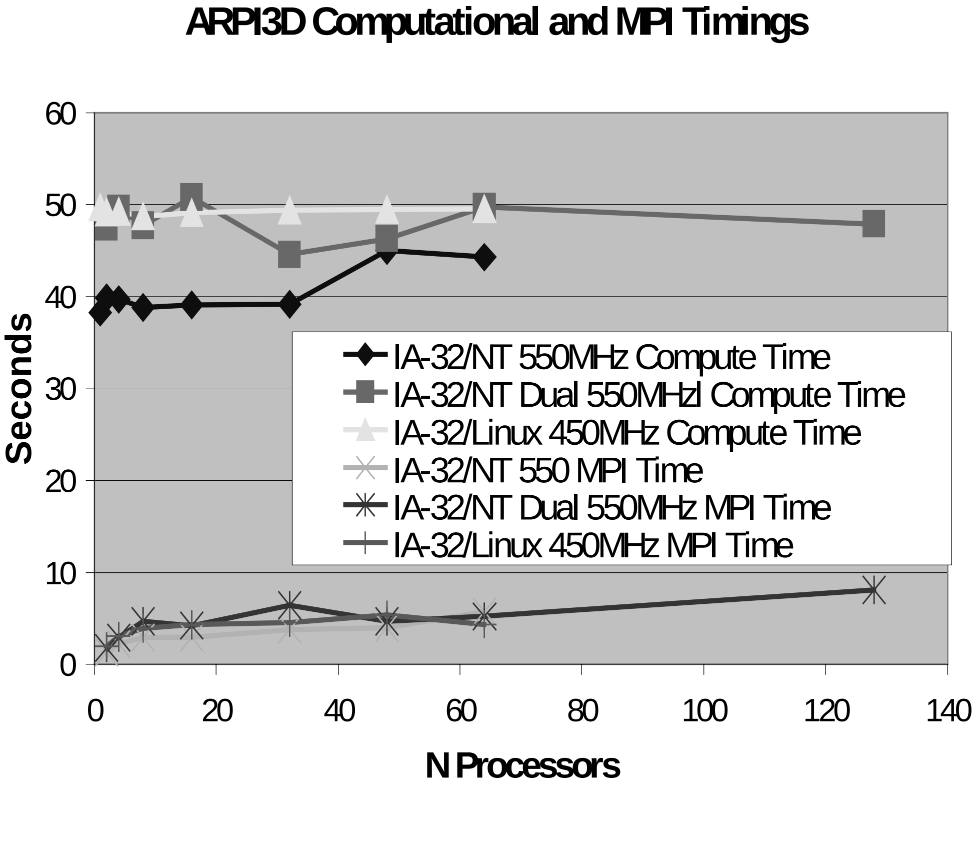
This is a very well instrumented code and it shows where the time is spent on the system to determine where the performance bottlenecks are located in the systems. The instrumentation provides times for the:

- Overall runtime of the application,
- Amount of time spent performing initialization with an input data read from a common file system,
- Amount of time spent doing the computations
- Amount of time spent performing message passing
- Amount of time spent writing out the resultant data files to a common file system.

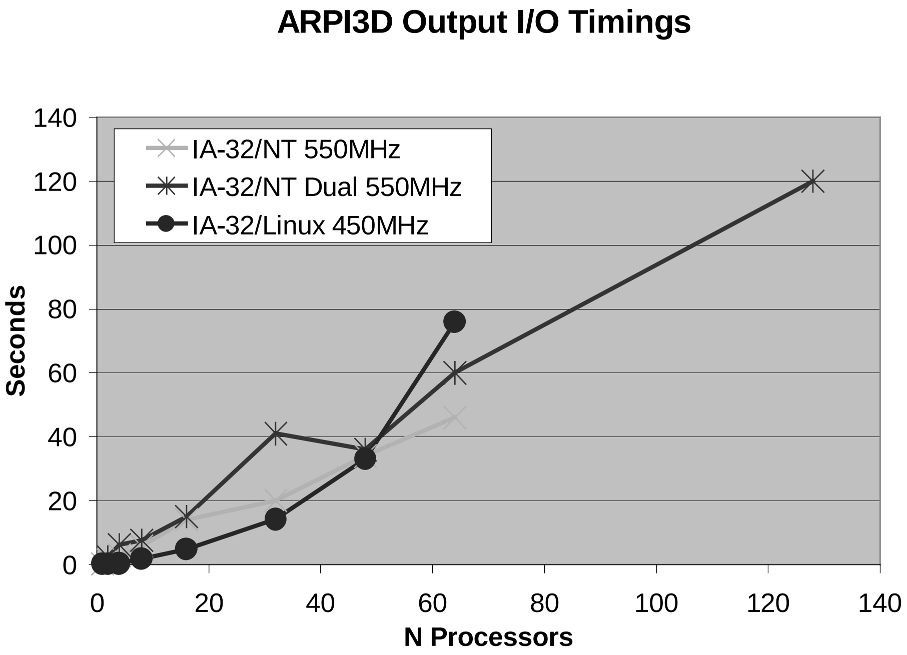
The first graph shows the overall performance of the application on the two systems. The amount of time necessary for the runs increases roughly linearly as the number of processors increases.




ARPI3D Initialization I/O Timings
Seconds
0
10
20
30
40
50
60
70
80
90
N Processors
0
20
40
60
80
100
120
140
IA-32/NT 550MHz
IA-32/NT Dual 550MHz
IA-32/Linux 450MHz


ARPI3D Computational and MPI Timings
Seconds
0
10
20
30
40
50
60
N Processors
0
20
40
60
80
100
120
140
IA-32/NT 550MHz Compute Time
IA-32/NT Dual 550MHzl Compute Time
IA-32/Linux 450MHz Compute Time
IA-32/NT 550 MPI Time
IA-32/NT Dual 550MHz MPI Time
IA-32/Linux 450MHz MPI Time

The size of the computational task per processor remains the same and the time taken for the computations is clearly shown to be nearly constant as a function of number of processors in the second graph, roughly 40 to 50 seconds, depending on the processor speed. Also shown in the second graph is the time spent by the application performing MPI calls and this is insensitive to the number of processors at 5 to 10 seconds for all of the runs, illustrating excellent scalability.



The third and fourth graphs respectively show the time spent reading in an initialization file and writing out an output file. The input data file that is read by all of the processors from the common file system is approximately 8 KB and each processor writes three output data files of 2.107 MBytes/s. The total volume of data moved is proportional to the number of processors in use, up to 800 MBytes/s of output for the 128 processor NT run. Both Superclusters have a similar type of configuration with a common filesystem available from a fileserver to the compute nodes across the commodity Ethernet, differing primarily in the speed of the link from the file server to the switch and the file sharing software.

The IA-32/NT Supercluster file serving configuration:

- Fast Ethernet from the file server to the switch
- Compute nodes are all on individual Fast Ethernet nodes on the switch.
- SMB to all nodes

The IA-32/Linux Roadrunner file-serving configuration:

- Gigabit Ethernet from the fileserver to the switch compute nodes are all on individual Fast Ethernet ports on the switch
- NFS to all nodes.

At smaller numbers of processors, the increased bandwidth of the server to the switch for the Roadrunner Linux system is apparent. At larger numbers of processors, the performance of the NT file server is clearly better. The NT Supercluster I/O times for input and output both increase linearly with the number of processors. In neither case is the I/O performance of the system sufficient for what is actually needed by the application. The overall increase in execution time as a function of number of processors is due almost entirely to the increased time necessary for file I/O as the number of machines increases.

9.4 Conclusions

Applications can perform very well on current generation clusters with the hardware and software that is now available. There are a number of areas where major improvements can be made such as the programming tools to generate new applications on these systems and the I/O systems that are available for clustered systems.

9.5 References

1. C. Bernard, T. Blum, A. De, T. DeGrand, C. DeTar, S. Gottlieb, A. Krasnitz, L. Karkkainen, J. Labrenz, R.L. Sugar, and D. Toussaint. Recent Progress on Lattice QCD with MIMD Parallel Computers, MIMD Lattice Computation (MILC) Collaboration, A Grand Challenge Application Group.
2. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *Technical report*, University of Tennessee, Knoxville, TN, June 1995. Version 1.1.
3. Compaq, Intel, and Microsoft. Virtual Interface Architecture Specification, Version 1.0. December 1997, http://www.viarch.org.
4. A. Chien, M. Lauria, R. Pennington, M. Showerman, G. Iannello, M. Buchanan, K. Connelly, L. Giannini, G. Koenig, S. Krishnmurthy, Q. Liu, S. Pakin, G. Sampemane. Design and Analysis of an HPVM-Based Windows Supercomputer. In *The International Journal of High Performance Computing Applications*, Volume 13, Fall 1999, pages 201-219.
5. D.A. Bader and A.B. Maccabe and J.R. Mastaler and J.K. McIver III and P.A. Kovatch. Design and Analysis of the Alliance/University of New Mexico Roadrunner Linux SMP SuperCluster. In Proceedings of the IEEE *International Workshop on Cluster Computing*, December 1999, Melbourne, Australia.

10. Embedded/Real-Time Systems

Daniel S. Katz, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA and Jeremy Kepner, MIT Lincoln Laboratory, Lexington, MA

10.1 Introduction

Embedded and real-time systems, like other computing systems, seek to maximize computing power for a given price, and thus can significantly benefit from the advancing capabilities of cluster computing. In addition to $/Mflops/s, embedded and real-time systems often have severe constraints on size, weight and power as well as latency and reliability. Satisfying these, constraints are usually achieved by reducing components such as disk drives, memory and access ports. These additional constraints have traditionally led to more customized solutions within a limited marketplace. More recently, embedded computer vendors have adopted the practice of using clusters of mainstream RISC processors configured in rack-mounted cabinets. In addition, many cluster vendors are adopting their rack-mounted systems to more compact environments. Reliability constraints in these markets have traditionally been handled by traditional fault avoidance (e.g. applying higher quality and more expensive fabrication procedures on older designs) and fault tolerance (e.g. replication) techniques. While effective, these techniques tend to work against the goals (cost, power, size and weight) of embedded real time systems.

This section of the paper seeks to provide an overview of the cluster computing technology found in the embedded computing world. We begin with a brief description of some of the major applications that use embedded computers. The next two sections present hardware issues (processors, interconnects and form factors) and software issues (operating systems and middleware). Finally, we conclude with a discussion of the current trend towards embedded systems looking more like clusters and clusters looking more like embedded systems.

10.2 Applications

In this section, we briefly discuss three application areas that are driving the development of embedded clusters.

10.2.1 Space Applications

One area where embedded clusters are being examined is in space. Space-borne instruments are providing data at ever increasing rates, far faster than is feasible to send the data to Earth, and the rate of growth of the data from the instruments is faster than the rate of growth of bandwidth to Earth, so this problem will only get worse in the future. One obvious answer to this is to process the data where it is collected, and to only return the results of the analysis to Earth, rather than the raw data. (A similar argument can be made regarding the signal delay between Earth and space when considering autonomous missions, leading to the same answer: placing the computing where the decisions must be made.)

Traditionally, very little data analysis has been done in space, and what has been done has relied on radiation-hardened processors. These processors are quite old by the time they complete the radiation-hardening process, and do not solve the problem of bandwidth limitation. What is needed is an embedded cluster of COTS (Commercial-Off-The-Shelf) processors, where the processors can be selected and placed in the system shortly before mission launch. In as much as COTS processors are not radiation hardened, this requires software that can detect and correct errors caused by the cosmic ray environment found in space. Such a system is being developed by the Jet Propulsion Laboratory under the Remote Exploration and Experimentation Project [1].

The constraints of the space environment (mass, power, size, resistance to vibration, shock, thermal cycling and natural space radiation) demand work in packaging that is very different than for most ground-based clusters. Additionally, replacement of faulty components is either impossible or extremely expensive, and uploading new or changing existing software is very difficult from a system design as well as operational perspective. Finally, reliability of the software as well as the hardware/system is a significant concern due to the difficulty in validating computational results, and the potential impact of erroneous behavior with respect to decision-making and scientific data analysis. Thus, a different set of requirements exists for space-borne computers.

10.2.2 Signal Processing Applications

The signal processing applications that embedded systems serve are often highly data parallel and naturally lend themselves to the kind of coarse-grained parallelism ideal to clusters. However, the Quality of Service (QoS) and latency requirements of real-time systems usually dictate a need for interconnects with both deterministic performance and higher performance than are typically found in clusters. Thus the solutions developed to serve these needs are similar to conventional rack mounted clusters with high performance interconnects, denser packaging and lighter operating systems.

10.2.3 Telecommunications Applications

In telecommunications, reliability and real-time response have always been critical factors. Early electronic switching systems used specially designed fault-tolerant computers such as the AT&T 3b2. With the current explosion in network servers, in order to achieve acceptable cost it is necessary to use the types of standard commercial machines and operating systems discussed in this section. This industry is currently experimenting with commodity cluster systems to solve the problem of meeting their ultra-high reliability and real-time response constraints while reducing costs.

10.3 Hardware

Form factor is a critical factor in embedded computing. Thus compute density (Gflops/s per cubic foot and Gflops/s per watt) is often as or more important than aggregate processing power. Typically, embedded system vendors are able to achieve roughly a factor of 10 increase in compute density over conventional systems. These gains are achieved by constructing boards containing multiple nodes (typically 2 or 4). Each node consists of a low power processor (e.g., Motorola PowerPC) with a limited amount of memory (e.g. 128 MBytes). In addition, there are no local disks and access to the node is limited to the interconnect, which may be custom or commodity (e.g. Myrinet), and will have been packaged to minimize size and power consumption. These various design tradeoffs allow embedded vendors to fit nearly 100 processing nodes in a volume that can fit underneath a typical office desk.

As mentioned previously, many embedded systems need to withstand much more severe conditions than standard clusters. These systems may be used in the aerospace or military industries, leading to requirements on tolerance to shock, vibration, radiation, thermal conditions, etc. While many of today's commercial components can handle these conditions, they are not packaged to do so, as this increases cost and is not needed by most ordinary users. Thus, for this niche market, different vendors have sprung up to package standard commercial parts with more consideration of these concerns.

There are a variety of vendors that manufacture systems along the above lines. Mercury, CSPI, and Sky are three of the more popular systems. Some of the general capabilities are shown in Table 10.1. For a comparison, a cluster vendor (AltaTech) is also shown.

Vendor	**CPU**	**Interconnect**	**OS**	**CPU/ft^3**
Mercury	PowerPC	Raceway	MCOS	~10
CSPI	PowerPC	Myrinet	VxWorks	~10
Sky	PowerPC	Sky Channel	SKYmpx	~10
Alta	Intel/Alpha	Ethernet/Myrinet	Linux	~1

Table 10.1 – General Capabilities

In addition to the vendors that specialize in embedded systems, a number of other companies build embedded systems, both parallel and distributed for their customers. These vendors may take systems from the standard vendors listed above and ruggedize and/or repackage them, and they include many US defense contractors (Lockheed, Honeywell, General Dynamics, etc.)

Many embedded systems are also targeted for real-time applications with extremely low latency requirements (e.g., radar signal processing). To achieve these requirements it often necessary to adopt a pipeline architecture with different processing occurring at each stage of the pipeline. Typically, each stages exploits coarse grain parallelism but the 'direction' of this parallelism is along different dimensions of the data at different steps. To fully exploit a parallel computer in such circumstances requires transposing (or “corner turning”) the data between steps. Thus, the interconnects provided by embedded systems often have higher bandwidth and lower latencies than those of shared memory supercomputers, let alone clusters of workstations.

10.4 Software

The real-time requirements of embedded systems necessitate special operating systems and middleware to reduce latency and to fully exploit the interconnects and processors. This has resulted in a wide variety of Unix flavored operating systems: VxWorks, Lynx, MCOS, SKYmpx, LinuxRT, IrixRT. Typically, these operating systems trade off memory protection, multi-user and multi-threaded capabilities to get higher performance. Traditionally, VxWorks has been one of the most common in many industries. However, the need for portable software has led many to examine alternatives operating systems for embedded clusters. These include LinuxRT and IrixRT. The advantage of these choices is that software can be developed (at least to some point) on common desktop machines, and easily transferred to the embedded clusters. Lynx and other POSIX-compliant systems are used similarly, under the assumption that software developed on one POSIX-compliant operating system can be easily ported to another. The primary distinction between these operating systems and VxWorks is that VxWorks does not provide process-based memory protection, which may be important in prevention of fault propagation from one process to another.

One of the most significant positive trends in embedded computing has been the adoption of common middleware libraries to ease portability between systems. The two major areas where this has occurred are in communications and math libraries. In the past, special vendor libraries were required to fully exploit their custom networks. Recently vendors have adopted MPI. With careful optimization, they are able to achieve performance similar to that of their proprietary libraries. MPI has been a large step forward for the embedded community, but it does not address all of the communication needs of these systems. This has led to the development of MPI/RT [2], which provides critical features such as quality of service. In addition to MPI/RT, the Data Reorganization Interface (DRI [3]) has been another standardization effort to provide a common interface to large data movements.

The mathematics libraries developed by embedded vendors are similar to other optimized math libraries in that they provide a variety of standard mathematical operations that have been tuned to a particular processor. The functional focus of the embedded libraries has primarily been on basic signal processing operations (e.g. FFT, FIR filters, linear algebra) for complex floating-point data. Because Fortran compilers for these systems are hard to find, these libraries usually only have a C implementation. One additional feature of these libraries has been the ability to pre-allocate memory. For most operations, this eliminates potential latency. Optimized math libraries are critical to achieving real-time performance and thus these libraries are heavily used in embedded real-time software. This heavy reliance can lead to a significant portability bottleneck. Fortunately, one of the most successful efforts of this community has been the adoption of a standardized Vector, Signal, and Image Processing Library (VSIPL [4]). In addition to being valuable to embedded systems, this library has a significant potential benefit to the general high performance computing community.

Creating an optimized implementation of a standard can be a significant undertaking for embedded systems vendors. Increasingly, these libraries are being implemented by third party software developers (e.g. MPI Software Technologies, Inc). However, as the embedded community is a niche market, software vendors generally do not have enough demand to optimize their products for the variety of embedded systems, and thus products that can optimize themselves, such as ATLAS and FFTW (as mentioned in section 7 of this white paper) will be increasingly important.

10.5 Conclusions

While there are certainly a number of differences between embedded clusters and standard clusters that have been brought out in this section, there are also a number of similarities, and in many ways, the two types of clusters are converging. Mass-market forces and the need for software portability are driving embedded clusters to use similar operating systems, tools, and interconnects as standard clusters. Additionally, as traditional clusters grow in size and complexity, there is a growing need to use denser packaging techniques and higher bandwidth, lower latency interconnects. Additionally, fault-tolerance is becoming more important for standard clusters: first, as they are increasingly accepted into machine rooms and subject to reliability and up-time requirements; and second, as feature sizes and operating voltages are reduced, cosmic-ray upsets will occur more frequently. The only areas that promise to continue to separate the two cluster worlds is the general need for ruggedized packaging for many embedded clusters.

10.6 References

1. REE – http://www-ree.jpl.nasa.gov/
2. MPI/RT – http://www.mpirt.org/
3. DRI – http://www.data-re.org/
4. VSIPL – http://www.vsipl.org/

11. Education

Daniel C. Hyde, Bucknell University, USA and Barry Wilkinson, University of North Carolina at Charlotte, USA

11.1 Introduction

Clusters offer an exciting opportunity for *all* institutions of higher education to teach and use high performance computing for the first time without access to expensive equipment. The Task Force on Cluster Computing (TFCC) can help in many ways.

Educators need help in answering such practical questions as: Is a cluster the best computing platform for undergraduate parallel programming courses? How can a cluster be set-up? What is the best software to use and where can it be obtained? How do I to manage a cluster? TFCC can facilitate the sharing of the experiences of others.

Teaching about clusters is new. Educators in both university and industrial settings (industrial seminars and short courses) need direction in selecting course contents and the fundamental principles to be taught. TFCC can help by providing a forum and leadership in developing and sharing curricular materials.

Academics at universities can provide industry with conceptual frameworks for clusters, explore the fundamental characteristics of clusters and devise models on how to model cluster behaviour and performance. TFCC can facilitate positive interactions between the cluster computing industrial research and the academic research communities.

11.2 Background

A cluster is a type of parallel or distributed system that consists of a collection of interconnected whole computers used as a single, unified computing resource. Clusters come in at least the following major flavours depending on their purpose, briefly:

1. **High Performance Computing Flavour**. An example is a Beowulf [1]. The purpose is to aggregate computing power across nodes to solve a problem faster. For example, high performance scientific computing typically spreads portions of the computation across the nodes of the cluster and uses message passing to communicate between the portions.
2. **High Throughput Computing Flavour**. These clusters harness the ever-growing power of desktop computing resources while protecting the rights and needs of their interactive users. These systems are a logical extension of the batch job environments on old mainframes. Condor is a good example [2].
3. **High Availability Computing Flavour**. These clusters are designed to provide high availability of service. Many vendors (e., g., IBM, DEC, SUN) provide high availability (HA) systems for the commercial world. Most of these systems use redundancy of hardware and software components to eliminate single points of failure. A typical system consists of two nodes, with mirrored disks, duplicate switches, duplicate I/O devices and multiple network paths. See Pfister's book [3].
4. **High Performance Service Flavour**. A cluster of nodes is used to handle a high demand on a web service, mail service, data mining service or other service. Typically, a request spawns a thread or process on another node.

11.3 What is Now Happening in Universities?

11.3.1 Graduate Level

At the graduate level, universities often offer a variety of parallel computing courses on architectures, algorithms, and programming. These courses have much in common with flavour 1 above, but ignore characteristics of the other three. These offerings are aimed at a small (less than 1%) and shrinking scientific computing market. Courses are also offered in distributed *something*, e.g., distributed operating systems, distributed computing, distributed systems, and distributed databases. Historically, graduate-level courses have used supercomputers available remotely, or local multiprocessors systems provided for research activities. Nowadays, high performance clusters are replacing such systems, for both research and teaching.

11.3.2 Undergraduate Level

There has been parallel computing and distributed computing courses at the undergraduate level for a few years, although distributed computing was barely mentioned in the ACM Curriculum '91 document. The newly formed IEEE/ACM Curriculum 2001 Committee has a Netcentric Task Force as part of their effort [4]. Members of TFCC should provide valuable input to the dialog that will surround the creation of this new curriculum.

Undergraduate parallel computing courses have typically used inexpensive transputer systems or multiprocessor simulators. A computing cluster offers a particularly attractive solution, and courses are appearing based upon clusters [5].

In addition to specific courses, the concepts to cover all four of the above flavours of clusters are scattered in bits and pieces in many courses. For example, topics needed for cluster computing such as scheduling, performance measurement, communication protocols, client-server paradigm, reliability theory, security, monitoring, and concurrency control are either not taught or scattered throughout a university's curriculum.

11.4 What is Needed to Improve the Situation?

The following need to be addressed:

1. A consensus should be established on the concepts to be taught and on the fundamental principles pertaining to clusters that will serve our students.
2. Examples of high quality courses are needed, including teaching strategies, lecture notes and laboratory exercises.
3. Material is needed to prepare our students to be able to design, analyze, implement, test and evaluate clusters effectively. A model of the behaviour of clusters is needed that can be used to model performance and availability.
4. More good textbooks are needed, especially for undergraduates. A recent textbook specifically for undergraduate teaching parallel programming on clusters is by Wilkinson and Allen [6].
5. More good reference books are needed for practitioners in the field and for advanced graduate students to acquire background. A recent excellent example is the two-volume set edited by Buyya [7, 8].

11.5 How can the TFCC help?

The TFCC will play a leadership role in education of clusters. To support educators, the TFCC will:

1. Collect educational materials and freely disseminate them
2. Serve as a forum for discussion.
3. Sponsor workshops and educational sessions at conferences.
4. Encourage the creation of resources such as books, bibliographies and web pages.
5. Encourage the writing of reviews of books, especially textbooks on cluster computing.
6. Be an active participant of the research culture on clusters.
7. Facilitate positive interactions between the industrial research community and the academic research community on cluster computing.

11.6 Conclusions

In only a few months of existence, the IEEE Computer Society's Task Force on Cluster Computing (TFCC) has made significant contributions to supporting cluster computing. It has brought together a large number of researchers and educators who would not otherwise had the opportunity to interact (including the two authors of this section). The work of the task force encompasses all aspects of cluster computing research as well as education.

To support educators, a web site has been created [9] which contains links to educational sites, related books and journals, freely available software, projects from both academia and industry, white papers, and descriptions of hardware components. A book donation program has been established with the generous cooperation of major publishers for the donation of some current books on clusters. Also, the TFCC has established an electronic list server and an on-line web-based newsletter for all its members.

The TFCC is committed to its efforts within the education arena of cluster computing. The success of these efforts requires active participation by the TFCC members from the education community. We strongly encourage you to become involved.

11.7 References

1. Beowulf Web Site: http://cesdis.gsfc.nasa.gov/linux/beowulf/beowulf.html
2. Condor Web Site: http://www.cs.wisc.edu/condor/
3. Pfister, Gregory F., *In Search of Clusters*, second edition, Prentice Hall, 1998.
4. IEEE/ACM Curriculum 2001 Web Site: http://computer.org/educate/cc2001/
5. Wilkinson, Barry and Michael Allen, "A State-Wide Senior Parallel Programming Course", *IEEE Transaction on Education*, Vol. 43, No. 3 (August), 1999, pp. 167-173.
6. Wilkinson, Barry and Michael Allen, *Parallel Programming Techniques and Application Using Networked Workstations and Parallel Computers*, Prentice Hall, 1999.
7. Buyya, R., (editor), *High Performance Cluster Computing: Architectures and Systems*, Vol. 1,. Prentice Hall, 1999.
8. Buyya, R., (editor), *High Performance Cluster Computing: Programming and Applications*, Vol. 2, Prentice Hall, 1999.
9. TFCC Education Web Site: http://www.coe.uncc.edu/~abw/parallel/links.html

Appendix A

Linux

In the public research community, Linux is the most frequently used operating system simply for two reasons: it is released as true open source (in differentiation on the conditions under which the source code for Sun Solaris and Microsoft Windows NT is available), and it is cheap in terms of both software and required hardware platforms. It offers all the functionality that is expected from standard Unix, and it is developing fast as missing functionality can be implement by anyone who needs it. However, these solutions are usually not as thoroughly tested as releases for commercial Unix variants. This requires frequent updates, which do not ease the job of the administrators to create a stable system, as it is required in commercial environments. The large-scale commercial use as a cluster OS is also severely limited by the missing HA functionality and unverified SMP scalability on CPU numbers greater than 2 or 4.

For scientific and research applications, these functional gaps are not really problematic, and the Linux concept is a great success, as can be seen from the well-known Beowulf project. This leads to a big and growing user community developing a great number of tools and environments to control and manage clusters, which are mostly available for free. However, it's often hard to configure and integrate these solutions to suit ones own, special set up as not all Linuxes are created equal: the different distributions are getting more and more different in the way the complete system is designed. Even the common base, the kernel, is in danger, as solution providers like TurboLinux [1] start to integrate advanced features into the Kernel.

In general, in has to be noted that the support for Linux by commercial hardware and software vendors is continuously improving, so it can no longer be ignored as a relevant computing platform.

Windows NT

Although NT contains consistent solutions for *local* system administration in a workstation or small-server configuration (consistent API, intuitive GUI-based management, registry database), it lacks several characteristics that are required for efficient use in large or clustered servers. The most prominent are standardized remote access and administration, SMP scaling in terms of resource limits and performance, dynamic reconfiguration, high-availability and clustering with more than a few nodes.

Through Microsoft's omnipresence and market power, the support for NT by the vendors of interconnect hardware and tools developers is good, and therefore a number of research projects in clustering for scientific computing is based on Windows NT (HPVM [2] and MPI [3, 4, 5]. Microsoft itself is working hard to extend the required functionality and will surely improve. The so-called "Datacenter Edition" of its forthcoming Windows2000 (which was delayed and postponed several times) has to prove how far NT has caught up with the established UNIX solutions.

AIX

IBM's AIX operating system, running on the SP series of clusters, is surely one of the most advanced solutions for commercial and scientific cluster computing. It has proven scalability and stability for several years and a broad number of applications in both areas. However, it is a closed system, and as such the development is nearly totally in the hand of IBM, which however has enough resources to create solutions in hardware and software that commercial customers demand, such as HA and DFS. Research from outside the IBM laboratories is very limited, though.

Solaris

In a way, Solaris can be seen as a compromise or merge of the three systems described above: it is not an open system, which ensures stability on a commercial level and a truly identical interface to administrators and programmers on every installation of the supported platforms. Of course, easy kernel extensions or modifications such as with Linux are not possible, making it less interesting for research in this area. It should be noted that the source code for Solaris and also Windows NT is available upon request, but not for free extension and publication of modifications.

It offers a lot of functionality required for commercial, enterprise-scale cluster-based computing like excellent dynamic reconfiguration and fail-over and also offers leading inter- and intra-nodes scalability for both, scientific and commercial clustering. Its support by commercial vendors is better for high-end equipment, and the available software solutions are also directed towards a commercial clientele. However, Solaris can be run on the same low-cost off-the-shelf hardware as Linux as well as on the original Sun Sparc-based equipment, and the support for relevant clustering hardware like inter-connects is given (Myrinet, SCI). Software solutions generated by the Linux community are generally portable to the Solaris platform with little or no effort.

The advantage of this situation is that solutions developed on simple, low-cost systems can easily be adapted for use in a computing environment of commercial-quality level. Of course, such an approach requires support and engagement of the manufacturer, Sun in this case.

References

1. *TurboLinux TurboCluster Server 4.0*, TurboLinux, Inc – http://www.turbolinux.com/products/tltcs.html
2. HPVM – http://www-csag.ucsd.edu/projects/hpvm.html
3. PATENT – *PaTENT MPI 4.0*, Genias Software – http://www.genias.de
4. *Multi-Platform MPICH*, Lehrstuhl für Betriebssysteme, RWTH Aachen – http://www.lfbs.rwth-aachen.de/~joachim/MP-MPICH.html
5. *MPICH for Windows NT*, Argonne National Laboratories – http://www.mcs.anl.gov/mpi/mpich/mpich-nt)

Appendix B

Compilers and Preprocessors

The Absoft Pro Fortran toolset [1] for Windows 95/98/NT/2000 includes globally optimizing Fortran 77, Fortran 90, and C/C++ compilers integrated with a development environment designed for Fortran users. Pro Fortran 6.2 is a Fortran toolset optimized for single processor Windows95/98 and Windows NT/2000 systems. Pro FortranMP 6.2 includes in addition a thread-safe runtime library and the VAST-F/Parallel preprocessor for automatically restructuring Fortran code for execution on dual processor systems. The Fx source-level debugger supporting Fortran and C/C++ is included with both toolsets, as is a performance profiler. All compilers and tools can be accessed via an integrated development environment (IDE) that automatically builds make files and generates header dependencies. The Absoft compilers are also available for Linux.

Compaq Visual Fortran [2] (formerly Digital Visual Fortran) is available for Windows 95/98 and Windows NT along with a development environment.

Visual KAP [3] is a preprocessor that automatically parallelizes Fortran 77/90/95 and ANSI C source code. Visual KAP runs on PentiumII/Pentium Pro/Pentium based machines under Windows NT 4.0 or Windows 95, and targets the Compaq Visual Fortran compiler. Since Windows 95 does not support parallelism, parallelism optimizations are available under Windows NT. Visual KAP works with optimizing compilers to provide additional speedups beyond what the compiler's built-in optimizer provides. Visual KAP has both command-line and graphical user interfaces.

Microsoft Visual C++ [4] is part of the Microsoft integrated development environment (IDE) called Visual Studio 6.0. The Visual C++ compiler can process both C source code and C++ source code. The compiler is compliant with all ANSI standards and has additional Microsoft extensions. The C++ Standard Template Library (STL) is included. The Visual C++ debugger has a graphical user interface for setting breakpoints, viewing classes and variables, etc. The debugger has an edit and continue feature that allows the user to change an application and continue debugging without manually exiting the debugger and recompiling.

VAST/Parallel [5] from Pacific-Sierra Research Corporation includes the VAST-F/Parallel and VAST-C/Parallel automatic parallelizing preprocessors for Fortran and C, respectively. VAST/Parallel transforms and restructures the input source to use parallel threads so that the program can automatically make use of parallel processors. They also support the OpenMP standard for user-directed parallelization. VAST/Parallel can optionally produce a diagnostic listing that indicates areas for potential improvement to assist the developer in tuning performance further. VAST/Parallel works with the DEEP development environment by gathering compile-time data for DEEP and inserting instrumentation code for run-time data gathering. DEEP uses this information to display compile-time optimization notes (e.g., which loop nests have been parallelized, which data dependencies are preventing parallelization) and run-time performance data (e.g., which loop nests use the most wallclock time, which procedures are called the most.

MPI Implementations

MPICH [6] is a freely available reference implementation of MPI developed by Argonne National Laboratory and Mississippi State University. Version 1.1.2 of MPICH is available for Linux on PCs and is in use on the Linux RoadRunner Supercluster at the University of New Mexico. Problems with MPICH programs on LINUX suddenly failing with lost TCP connections are on the list of things to fix in future MPICH releases. The MPICH development team at Argonne is involved in a research effort with LBNL to develop MPICH on top of VIA, the Virtual Interface Architecture, a specification of an industry-standard architecture for scalable communication within clusters. MPICH for Windows NT is now a beta released.

MPI-FM [7], developed by the Concurrent Systems Architecture Group at the University of Illinois at Urbana-Champaign and the University of California, San Diego, is a version of MPICH built on top of Fast Messages. MPI-FM requires the High Performance Virtual Machine (HPVM) runtime environment that is available for both Linux and Windows NT. MPI-FM is in use on the NCSA NT Supercluster[8].

MPI/Pro [9] is a commercial MPI implementation from MPI Software Technology, Inc. The current release of MPI/Pro is for Windows NT on Intel and Alpha processors, but MPI/Pro for Linux will be released soon. MPI/Pro is based on a version of MPICH for Win32 platforms that was developed at Mississippi State. MPI Software Technology is working on a new version of MPI/Pro that does not include any MPICH code and that supports VIA.

WMPI (Windows Message Passing Interface) [10] was the one of the first implementations for Win32 systems. WMPI is an ongoing effort by The University of Coimbra, Portugal. The first version was released in 1996 and was a port of the MPICH/P4 to Win32 systems. More recently a new internal architecture has been developed that provides greater functionality to the runtime library and enables WMPI to move towards a dynamic environment based on the MPI-2 standard. The latest released version, with the new internal design, is able to use multiple devices simultaneously and is also thread-safe.

PaTENT MPI 4.0 [11] is a commercial MPI implementation for Windows NT available from Genias GmbH. PaTENT stands for Parallel Tools Environment for NT. Genias plans to release a suite of development tools for MPI programs on NT, including debugging and performance analysis support.

FT-MPI [12] developed as part of the HARNESS [13] project at the University of Tennessee, Knoxville is a subset of the MPI 1.2 and MPI 2 standards that provides additional fault tolerance to allow applications to handle process failures, and subsequent communicator corruption. Failure recovery consists of rebuilding communicators, by either shrinking, padding, re-spawning or user run-time, communicator specific methods. FT-MPI uses the SNIPE_Lite [14] multi-protocol communications library and supports, TCP/UDP sockets, shared memory, BIP [15], GM and ELWMCP.

Development Environments

Some vendors of Linux and Windows products are coming out with integrated development environments (IDEs) that integrate tools such as compilers, preprocessors, source code editors, debuggers, and performance analysis tools. Some IDEs are a complete product from a single vendor. Others provide a framework and some tools but integrate other tools, such as compilers, from other vendors.

DEEP [16] from Pacific-Sierra Research stands for Development Environment for Parallel Programs. DEEP provides an integrated interactive GUI interface that binds performance, analysis, and debugging tools back to the original parallel source code. DEEP supports Fortran 77/90/95, C, and mixed Fortran and C in Unix and Windows 95/98/NT environments.

DEEP supports both shared memory (automatic parallelization, OpenMP) and distributed memory (MPI, HPF, Data Parallel C) parallel program development. A special version of DEEP called DEEP/MPI [17] is aimed at support of MPI programs.

Debuggers

DEEP/MPI [17] from Pacific Sierra Research [18] is a development environment for MPI parallel programs. DEEP/MPI debugging support is available for Linux on PCs. DEEP/MPI provides a graphical interface for parallel debugging of Fortran or C MPI programs. Capabilities include setting breakpoints, watching variables, array visualization, and monitoring process status. DEEP/MPI for Linux has been tested with MPICH and LAM MPI 6.1.

The PGDBG Debugger is part of the Portland Group, Inc. (PGI) PGI Workstation 3.0 development environment for Intel processor-based Linux, NT, and Solaris86 clusters. PGDBG supports threaded shared-memory parallelism for auto-parallelized and OpenMP programs, but does not provide explicit support for MPI parallel programming.

GDB, the GNU Debugger, is available for both Linux and Windows NT on PCs from Cygnus [19]. GDB supports debugging of multithreaded code but does not provide explicit support for MPI parallel programming.

Performance Analyzers

DEEP/MPI [17] from Pacific Sierra Research is a development environment for MPI parallel programs. DEEP/MPI performance analysis support is available for both Windows NT and Linux on PCs. DEEP/MPI provides a graphical user interface for program structure browsing, profiling analysis, and relating profiling results and MPI events to source code. DEEP/MPI for Linux has been tested with and supplies MPI profiling libraries for MPICH and LAM MPI 6.1. A driver called *mpiprof* is used to instrument, compile, and build MPI programs. Running *mpiprof* results in the creation of a subdirectory called *deep* in which the static information files created for each file you instrument are saved. When you execute your MPI program as you normally do, runtime information files are also stored in this subdirectory. To analyze following execution, start DEEP/MPI by executing the command *deep* which will bring up a File Dialog box asking you the location of your program. Currently DEEP requires that all source files be located in the same directory and that all static and runtime created files be located in the *deep* subdirectory, but this restriction is expected to be removed in future releases.

VAMPIR [20] is a performance analysis tool for MPI parallel programs. The VAMPIRtrace MPI profiling library is available for Linux on PCs. VAMPIR provides a graphical user interface for analyzing tracefiles generated by VAMPIRtrace.

The Jumpshot graphical performance analysis tool is provided with the MPICH [6] distribution. Jumpshot analyzes tracefiles produced by the MPE logging library, which is an MPI profiling library also provided with MPICH for Linux. Jumpshot is written in Java and interprets tracefiles in the binary clog format by displaying them onto a canvas object. Jumpshot itself is available for Windows (a JVM for Windows is provided with the Jumpshot distribution), and MPICH for Windows NT is supposed to be available soon. By default, Jumpshot shows a timeline view of the state changes and message passing behaviour of the MPI processes. Clicking any mouse button on a specific state instance will display more information about that state instance. Clicking any mouse button on the circle at the origin of a message will display m ore information about that message. That Jumpshot's performance decreases as the size of the logfile increases is a known bug, and can ultimately result in Jumpshot hanging while it is reading in the logfile. There is a research effort underway to make Jumpshot significantly more scalable. Jumpshot can be run as an application or as an applet using a web browser or applet viewer.

PGPROF [21] is part of the Portland Group, Inc. (PGI) PGI Workstation 3.0 development environment for Intel processor-based Linux, NT, and Solaris86 clusters. PGPROF supports threaded shared-memory parallelism for auto-parallelized and OpenMP programs, but does not provide explicit support for MPI parallel programming. The PGPROF profiler supports function-level and line-level profiling of C, C++, Fortran 77, Fortran 90, and HPF programs. A graphical user interface displays and supports analysis of profiling results.

Paradyn [22] is a tool for measuring and analyzing performance of large-scale long-running parallel and distributed programs. Paradyn operates on executable images by dynamically inserting instrumentation code while the program is running. Paradyn can instrument serial programs running on WindowsNT/x86 but does not yet support any MPI implementation on that platform.

VTune [23] is a commercial performance analyzer for high-performance software developers on Intel processors, including Pentium III, under Windows NT. VTune collects, analyzes, and provides architecture-specific performance data from a system-wide view down to a specific module, function, or instruction in your code. VTune periodically interrupts the processor and collects samples of instruction addresses and matches these with application or operating system routines, and graphically displays the percentage of CPU time spent in each active module, process, and processor. VTune provides access to the Pentium hardware performance counters under Windows NT.

PAPI [24] is a platform-independent interface to hardware performance counters. A PAPI implementation is available for Linux/x86.

References

1 Absoft – http://www.absoft.com/pro.win.html
2 Compaq – http://www.compaq.com/fortran/
3 Visual KAP – http://www.kai.com/visualkap/
4 Microsoft – http://msdn.microsoft.com/visualc/
5 VAST/Parallel – http://www.psrv.com/vast_parallel.html
6 MPICH – http://www-unix.mcs.anl.gov/mpi/mpich/
7 MPI-FM – http://www-csag.ucsd.edu/projects/comm/mpi-fm.html
8 NCSA – NT SuperCluster - http://www.ncsa.uiuc.edu/SCD/Hardware/NTCluster/
9 MPI Software Technology - http://www.mpi-softtech.com/
10 WMPI - http://dsg.dei.uc.pt/wmpi
11 PaTENT MPI – http://www.genias.de/products/patent/
12 FT-MPI: Fault Tolerant MPI for HARNESS, *Technical Report*, Dept of Computer Science, University of Tennessee, Knoxville, December 1999.
13 Beck, Dongarra, Fagg, Geist, Gray, Kohl, Migliardi, K. Moore, T. Moore, P. Papadopoulous, S. Scott and V. Sunderam, HARNESS: a next generation distributed virtual machine, *Journal of Future Generation Computer Systems*, (15), pp. 571-582, Elsevier Science B.V., 1999.
14 G. Fagg, K. Moore, J. Dongarra and A. Geist, Scalable Networked Information Processing Environment (SNIPE), *proc of SuperComputing 97*, San Jose, 1997.

15 Geoffray, L. Lefvre, C. Pham, L. Prylli, O. Reymann, B. Tourancheau, and R. Westrelin, High-speed LANs: New environments for parallel and distributed applications, *In Proceedings of EuroPar'99*, LNCS 1685, pp633-642, August 31-September 3, 1999, Toulouse, France.

16 DEEP – http://www.psrv.com/deep.ht

17 DEEP/MPI – http://www.psrv.com/deep_mpi_top.html

18 PGI Workstation – http://www.pgroup.com/prod_description.html

19 Cygnus – http://www.cygnus.com/

20 Vampir – http://www.pallas.de/pages/vampir.htm

21 PGPROF – http://www.pgroup.com/ppro_prof_desc.html

22 Paradyn – http://www.cs.wisc.edu/paradyn/

23 Vtune – http://www.pts.com/static/Vtune.html

24 PAPI – http://icl.cs.utk.edu/projects/papi/