*A dissertation on*

# Developing a cluster using load balancing approach

*Submitted in partial fulfillment of*
*the requirements for the award of the degree of*

**MASTER OF ENGINEERING**
**(Computer Technology & Applications)**

By
**Anshum Verma**
University Roll No. 8504

Under the guidance of
**Prof. D. Roy Choudhary**

Department Of Computer Engineering

Delhi College of Engineering

Bawana Road, Delhi-110042

## CERTIFICATE

This to certify that the work contained in the dissertation entitled ***"Developing a cluster using load balancing approach"*** by Anshum Verma has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Prof. D. Roy Choudhury
*Department of Computer Engineering*
*Delhi College of Engineering, Delhi*

## Acknowledgement

It is a great pleasure to have the opportunity to extent my heartiest felt gratitude to everybody who helped me throughout the course of this project.

I would like to express my heartiest felt regards to **Prof. D. Roy Choudhary**, Department of Computer Engineering for the constant motivation and support during the duration of this project. It is my privilege and owner to have worked under the supervision. His invaluable guidance and helpful discussions in every stage of this project really helped me in materializing this project. It is indeed difficult to put his contribution in few words.

I would also like to take this opportunity to present my sincere regards to my teachers viz. Dr Goldie Gabrani, Dr. S.K. Saxena, Mr. Rajeev Kumar and Mrs. Rajni Jindal for their support and encouragement.

I am thankful to my friends and classmates for their unconditional support and motivation during this project.

**Anshum Verma**
*M.E.(Computer Technology and Applications)*
*Department of Computer Science*
*Delhi College of Engineering, Delhi*

## Abstract

Parallel computing has seen many changes since the days of the highly expensive and proprietary super computers. Changes and improvements in performance have also been seen in the area of mainframe computing for many environments. But these compute environments may not be the most cost effective and flexible solution for a problem. Over the past decade, cluster technologies have been developed that allow multiple low cost computers to work in a coordinated fashion to process applications. The economics, performance and flexibility of compute clusters makes cluster computing an attractive alternative to centralized computing models and the attendant to cost, inflexibility, and scalability issues inherent to these models.

Many enterprises are now looking at clusters of high-performance, low cost computers to provide increased application performance, high availability, and ease of scaling within the data center. Interest in and deployment of computer clusters has largely been driven by the increase in the performance of off-the-shelf commodity computers, high-speed, low-latency network switches and the maturity of the software components. Application performance continues to be of significant concern for various entities including governments, military, education, scientific and now enterprise organizations.

This dissertation aims at providing cluster as a solution for the processing intensive applications used here in the laboratories of Delhi College of Engineering. This is being done so as to increase the efficiency and productivity of these labs and will also help students to dwell deeper into the subject.

The cluster being designed uses load balancing approach as suggested in openMosix architecture. The advantage of using this approach is that all general purpose applications can be parallelized over the cluster and not only those which are specifically written for the parallel architecture. The following are advantages of using cluster to achieve high performance computing:

- Scalable capacity for compute, data, and transaction intensive applications, including support of mixed workloads
- Horizontal and vertical scalability without downtime
- Ability to handle unexpected peaks in workload
- Central system management of a single systems image
- 24 x 7 availability

This dissertation also provides the performance test of the cluster being designed so as to study the performance advantage being achieved and also realize its feasibility.

## Contents

# List of Figures

# List of Tables

# 1. Introduction

Very often applications need more computing power than a sequential computer can provide. One way of overcoming this limitation is to improve the operating speed of processors and other components so that they can offer the power required by computationally intensive applications. Even though this is currently possible to certain extent, future improvements are constrained by the speed of light, thermodynamic laws, and the high financial costs for processor fabrication. A viable and cost-effective alternative solution is to connect multiple processors together and coordinate their computational efforts. The resulting systems are popularly known as ***parallel computers***, and they allow the sharing of a computational task among multiple processors. The main reason for creating and using parallel computers is that parallelism is one of the best ways to overcome the speed bottleneck of a single processor. In addition, the price performance ratio of a small cluster-based parallel computer as opposed to a minicomputer is much smaller and consequently a better value. In short, developing and producing systems of moderate speed using parallel architectures is much cheaper than the equivalent performance of a sequential system.

Parallel computing has seen many changes since the days of the highly expensive and proprietary super computers. Changes and improvements in performance have also been seen in the area of mainframe computing for many environments. But these compute environments may not be the most cost effective and flexible solution for a problem. Over the past decade, cluster technologies have been developed that allow multiple low cost computers to work in a coordinated fashion to process applications. The economics, performance and flexibility of compute clusters makes cluster computing an attractive alternative to centralized computing models and the attendant to cost, inflexibility, and scalability issues inherent to these models.

***Cluster computing*** is best characterized as the integration of a number of off-the-shelf commodity computers and resources integrated through hardware, networks, and software to behave as a single computer. Initially, the terms cluster computing and high performance computing were viewed as one and the same. However, the technologies available today have redefined the term cluster computing to extend beyond parallel computing to incorporate load-balancing clusters (for example, web clusters) and high availability clusters. Clusters may also be deployed to address load balancing, parallel processing, systems management, and scalability.

Many enterprises are now looking at clusters of high-performance, low cost computers to provide increased application performance, high availability, and ease of scaling within the data center. Interest in and deployment of computer clusters has largely been driven by the increase in the performance of off-the-shelf commodity computers, high-speed, low-latency network switches and the maturity of the software components. Application performance continues to be of significant concern for various entities including governments, military, education, scientific and now enterprise organizations.

## 1.1 Purpose and Problem Definition

The Department of Computer Science at Delhi College of Engineering has a large Linux laboratory which could be used to achieve high performance capabilities. An idea was that these computers could be put together into a so-called cluster, where users can run calculations and simulations. This project is about developing a cluster system and to see what performance scaling could we achieve by using this kind of system.

## 1.2 Assumptions

It is assumed that the system processes already running with the default configuration are not affected by introduction of changes in the kernel. This is in the interest of already running critical processes should not be affected and giving an optimal performance and stability of kernel.

It is also assumed that the applications being run over cluster are distributive nature or are those written using API functions, which are used for designing high performance parallel programs. This is because the cluster is being designed using MPI tools hence applications which could distribute themselves over MPI could only be executed to utilize the cluster to its fullest.

The document is made simple by providing all the details about the deploying of the cluster and details about it user tools are also provided in Appendix that follows the document. But an assumption is made that the reader has working knowledge of Linux upto the user level and a brief knowledge on modifying the kernel to customize it according to user needs. In case of unfamiliarity with Linux it is suggested to keep a Linux handbook handy in case of making references.

## 1.3 Approach

The cluster has been developed using the systems running Linux kernel version 2.6.15. The systems are connected in such a way so as to implement the cluster in homogeneous as well as heterogeneous manner to study the different behaviour patterns of the overall system. The performance of system is analysed using a token ring simulation software which is marginally intensive process in terms of processor utilization. Chapter 2 introduces to cluster and its typical architectural details along with various components used to design and manipulate it. Chapter 3 gives the design and deployment of openMosix architecture on the cluster. Chapter 4 contains the results and their analysis of the performance tests being carried out in different configurations of the system. Finally the conclusion and future work are covered in Chapter5.

## 1.4 Why Cluster?

The problem of achieving high performance could easily be solved if we deploy a parallel computing architecture for such purposes. The main reason for creating and using parallel computers is that parallelism is one of the best ways to overcome the speed bottleneck of a single processor. In addition, the price performance ratio of a small cluster-based parallel computer as opposed to a minicomputer is much smaller and consequently a better value. In short, developing and producing systems of moderate speed using parallel architectures is much cheaper than the equivalent performance of a sequential system.

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

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

The table given below shows comparison of the architectural and functional characteristics of these machines:

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

**Table 1** Comparison of characteristics of different parallel architectures

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

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

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

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

At a basic level a cluster is a collection of workstations or PCs that are inter-connected via some network technology. For parallel computing purposes, a cluster will generally consist of high performance workstations or PCs interconnected by a high-speed network. A cluster works as an integrated collection of resources and can have a single system image spanning all its nodes.

The following list highlights some of the reasons NOW is preferred over specialized parallel computers:

- Individual workstations are becoming increasingly powerful. That is, workstation performance has increased dramatically in the last few years and is doubling every 18 to 24 months. This is likely to continue for several years, with faster processors and more efficient multiprocessor machines coming into the market.
- The communications bandwidth between workstations is increasing and latency is decreasing as new networking technologies and protocols are implemented in a LAN.
- Workstation clusters are easier to integrate into existing networks than special parallel computers.
- Typical low user utilization of personal workstations.
- The development tools for workstations are more mature compared to the contrasting proprietary solutions for parallel computers, mainly due to the nonstandard nature of many parallel systems.
- Workstation clusters are a cheap and readily available alternative to specialized high performance computing platforms.

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

The main benefits of clusters are scalability, availability, and performance. For scalability, a cluster uses the combined processing power of compute nodes to run cluster-enabled applications such as a parallel database server at a higher performance than a single machine can provide. Scaling the cluster's processing power is achieved by simply adding additional nodes to the cluster.

Availability within the cluster is assured as nodes within the cluster provide backup to each other in the event of a failure. In high-availability clusters, if a node is taken out of service or fails, the load is transferred to another node (or nodes) within the cluster. To the user, this operation is transparent as the applications and data running are also available on the failover nodes.

An additional benefit comes with the existence of a single system image and the ease of manageability of the cluster. From the users perspective the users sees an application resource as the provider of services and applications. The user does not know or care if this resource is a single server, a cluster, or even which node within the cluster is providing services.

These benefits map to needs of today's enterprise business, education, military and scientific community infrastructures. In summary, clusters provide:

- Scalable capacity for compute, data, and transaction intensive applications, including support of mixed workloads
- Horizontal and vertical scalability without downtime
- Ability to handle unexpected peaks in workload
- Central system management of a single systems image
- 24 x 7 availability

## 1.5 Overview

There are several cluster architectures that could be used to deploy such system, but the best possible was that of openMosix due to its several advantages: openMosix is under a free license (GPL = Gnu Public License), plus for some other reasons, like more tools, better documentation and more frequent releases it was decided that openMosix should be tested.

The items to be compared were: what hardware and operating system the clustering system could run on, how easy it is to install and administrate and how was it best fitted for a system running computations and simulations. A performance test should also be done on a test system consisting of PC's. A how-to about the clustering system that is considered best for the task should then be written.

The test system used during the project consisted of PC's connected via 100 Mbit full-duplex Ethernet. The computers first had Linux installed on them; the PCs ran the RedHat Enterprise Linux distribution. After that openMosix was compiled on the PC's. When the systems had been installed the different administrative and monitoring tools were tested in order to evaluate the cluster administration. After this was done it was time to start working on the performance tests by searching for information and read about how to do performance testing, looking up what benchmark or benchmarks that should be used and possible tools to see the performance. It was decided to use a token-ring simulation as a benchmark mainly because that it was the only program that could be found where parallel tests on openMosix could be made.

Finally a conclusion on feasibility of system for department was made how-to was written for openMosix. This how-to describe how to install, expand and administrate the clustering systems.

# 2. Cluster and its Architecture

## 2.1 Definition

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

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

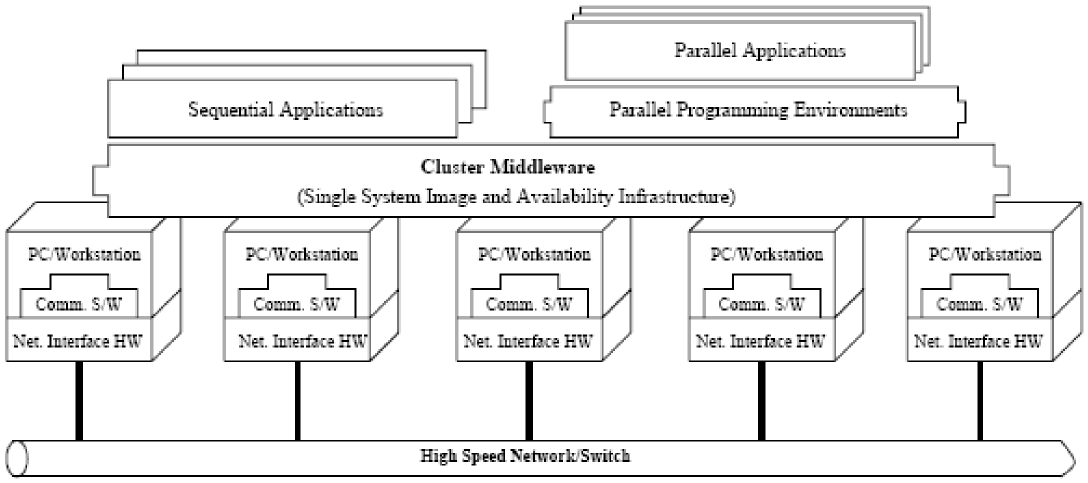


**Figure 1** Typical Architecture of a cluster

## 2.2 Types of Clusters

There are several types of clusters, each with specific design goals and functionality. These clusters range from distributed or parallel clusters for computation intensive or data intensive applications that are used for protein, seismic, or nuclear modeling to simple load-balanced clusters.

### High Availability or Failover Clusters

These clusters are designed to provide uninterrupted availability of data or services (typically web services) to the end-user community. The purpose of these clusters is to ensure that a single instance of an application is only ever running on one cluster member at a time but if and when that cluster member is no longer available, the application will failover to another cluster member. With a high-availability cluster, nodes can be taken out-of-service for maintenance or repairs. Additionally, if a node fails, the service can be restored without affecting the availability of the services provided by the cluster. While the application will still be available, there will be a performance drop due to the missing node.

High-availability clusters implementations are best for mission-critical applications or databases, mail, file and print, web, or application servers.

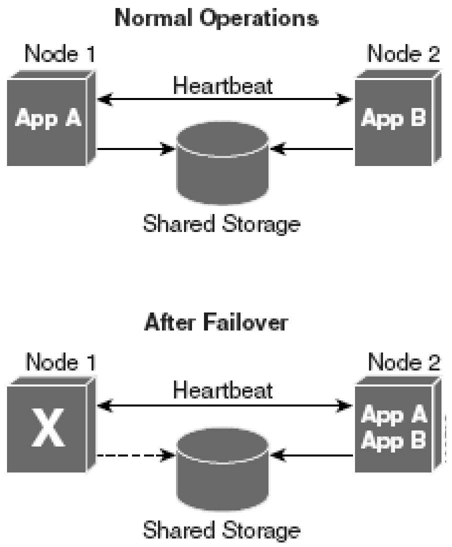


**Figure 2** Failover or High Availability clusters

Unlike distributed or parallel processing clusters, high-availability clusters seamlessly and transparently integrate existing standalone, non-cluster aware applications together into a single virtual machine necessary to allow the network to effortlessly grow to meet increased business demands.

## Cluster-Aware and Cluster-Unaware Applications

Cluster-aware applications are designed specifically for use in clustered environment. They know about the existence of other nodes and are able to communicate with them. Clustered database is one example of such application. Instances of clustered database run in different nodes and have to notify other instances if they need to lock or modify some data.

Cluster-unaware applications do not know if they are running in a cluster or on a single node. The existence of a cluster is completely transparent for such applications, and some additional software is usually needed to set up a cluster. A web server is a typical cluster-unaware application. All servers in the cluster have the same content, and the client does not care from which server the server provides the requested content.

## Load Balancing Cluster

This type of cluster distributes incoming requests for resources or content among multiple nodes running the same programs or having the same content. Every node in the cluster is able to handle requests for the same content or application. If a node fails, requests are redistributed between the remaining available nodes. This type of distribution is typically seen in a web-hosting environment.

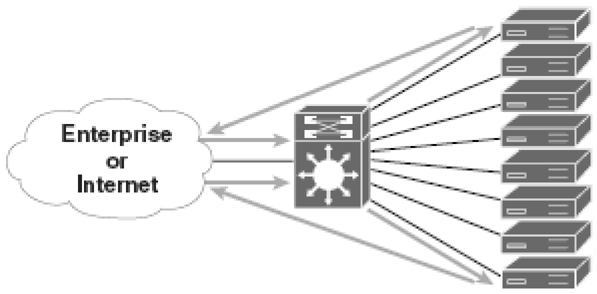


**Figure 3** Load Balancing cluster

Both the high availability and load-balancing cluster technologies can be combined to increase the reliability, availability, and scalability of application and data resources that are widely deployed for web, mail, news, or FTP services.

## Parallel/Distributed Processing Clusters

Traditionally, parallel processing was performed by multiple processors in a specially designed parallel computer. These are systems in which multiple processors share a single memory and bus interface within a single computer. With the advent of high speed, low-latency switching technology, computers can be interconnected to form a parallel-processing cluster. These types of cluster increase availability, performance, and scalability for applications, particularly computationally or data intensive tasks. A parallel cluster is a system that uses a number of nodes to simultaneously solve a specific computational or data-mining task. Unlike the load balancing or high-availability clusters that distributes requests/tasks to nodes where a node processes the entire request, a parallel environment will divide the request into multiple sub-tasks that are distributed to multiple nodes within the cluster for processing. Parallel clusters are typically used for CPU-intensive analytical applications, such as mathematical computation, scientific analysis (weather forecasting, seismic analysis, etc.), and financial data analysis.

One of the more common cluster operating systems is the Beowulf class of clusters. A Beowulf cluster can be defined as a number of systems whose collective processing capabilities are simultaneously applied to a specific technical, scientific, or business application. Each individual computer is referred to as a “node” and each node communicates with other nodes within a cluster across standard Ethernet technologies (10/100 Mbps, GbE, or 10GbE). Other high-speed interconnects such as Myrinet, Infiniband, or Quadrics may also be used.

## 2.3 An alternative classification

Clusters offer the following features at a relatively low cost:

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

Cluster technology permits organizations to boost their processing power using standard technology (commodity hardware and software components) that can be acquired at a relatively low cost. This provides expandability {an affordable upgrade path that lets organizations increase their computing power {while preserving their existing investment and without incurring a lot of extra expenses. The performance of applications also improves with the support of scalable software environment. Another benefit of clustering is a failover capability that allows a backup computer to take over the tasks of a failed computer located in its cluster.

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

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

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

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

    - Dedicated Clusters
    - Non-dedicated Clusters

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

    In non-dedicated clusters, a tension exists between the workstation owners and remote users who need the workstations to run their application. The former expects fast interactive response from their workstation, while the latter is only concerned with fast application turnaround by utilizing any spare CPU cycles.

This emphasis on sharing the processing resources erodes the concept of node ownership and introduces the need for complexities such as process migration and load balancing strategies. Such strategies allow clusters to deliver adequate interactive performance as well as to provide shared resources to demanding sequential and parallel applications.

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

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

4. Node Operating System - Linux, NT, Solaris, AIX, etc.

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

    .

5. Node Configuration - Node architecture and type of OS it is loaded with.

    - Homogeneous Clusters: All nodes will have similar architectures and run the same OSs.
    - Heterogeneous Clusters: All nodes will have di_erent architectures and run different OSs.

6. Levels of Clustering - Based on location of nodes and their count.

    - Group Clusters (#nodes: 2-99): Nodes are connected by SANs (System Area Networks) like Myrinet and they are either stacked into a frame or exist within a center.
    - Departmental Clusters (#nodes: 10s to 100s)
    - Organizational Clusters (#nodes: many 100s)
    - National Meta-computers (WAN/Internet-based): (#nodes: many departmental/organizational systems or clusters)
    - International Meta-computers (Internet-based): (#nodes: 1000s to many millions)

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

## 2.4 Commodity Components for Clusters

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

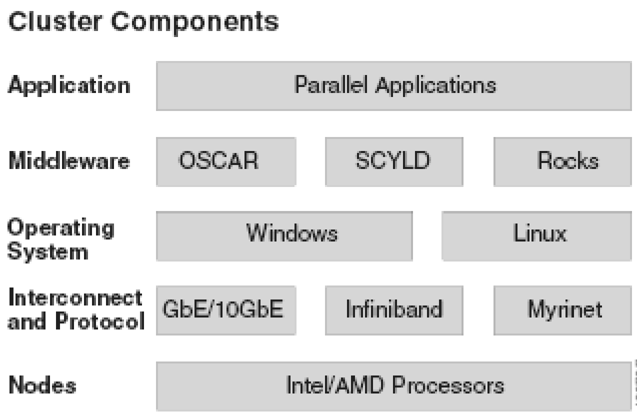


**Figure 4** Cluster Components

### Processors

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

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

Other popular processors include x86 variants (AMD x86, Cyrix x86), Digital Alpha, IBM PowerPC, Sun SPARC, SGI MIPS, and HP PA. Computer systems based on these processors have also been used as clusters; for example, Berkeley NOW uses Sun's SPARC family of processors in their cluster nodes

## Memory and Cache

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

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

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

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

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

## Disk and I/O

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

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

## System Bus

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

A group of PC manufacturers introduced the VESA local bus, a 32-bit bus that matched the system's clock speed. The VESA bus has largely been superseded by the Intel-created PCI bus, which allows 133 Mbytes/s transfers and is used inside Pentium-based PCs. PCI has also been adopted for use in non-Intel based platforms such as the Digital AlphaServer range. This has further blurred the distinction between PCs and workstations, as the I/O subsystem of a workstation may be built from commodity interface and interconnects cards.

## Cluster Interconnects

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

*Ethernet, Fast Ethernet, and Gigabit Ethernet*

Standard Ethernet has become almost synonymous with workstation networking. This technology is in widespread usage, both in the academic and commercial sectors. However, its 10 Mbps bandwidth is no longer sufficient for use in environments where users are transferring large data quantities or there are high traffic densities.

An improved version, commonly known as Fast Ethernet, provides 100 Mbps bandwidth and has been designed to provide an upgrade path for existing Ethernet installations. Standard and Fast Ethernet cannot coexist on a particular cable, but each uses the same cable type. When an installation is hub-based and uses twisted-pair it is possible to upgrade the hub to one, which supports both standards, and replace the Ethernet cards in only those machines where it is believed to be necessary.

Now, the state-of-the-art Ethernet is the Gigabit Ethernet2 and its attraction is largely due to two key characteristics. First, it preserves Ethernet's simplicity while enabling a smooth migration to Gigabit-per-second (Gbps) speeds. Second, it delivers a very high bandwidth to aggregate multiple Fast Ethernet segments and to support high-speed server connections, switched intra-building backbones, inter-switch links, and high-speed workgroup networks.

*Asynchronous Transfer Mode (ATM)*

ATM is a switched virtual-circuit technology and was originally developed for the telecommunications industry [12]. It is embodied within a set of protocols and standards defined by the International Telecommunications Union. The international ATM Forum, a non-profit organization, continues this work. Unlike some other networking technologies, ATM is intended to be used for both LAN and WAN, presenting a unified approach to both. ATM is based around small fixed-size data packets termed cells. It is designed to allow cells to be transferred using a number of different media such as both copper wire and fiber optic cables. This hardware variety also results in a number of different interconnect performance levels.

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

*Scalable Coherent Interface (SCI)*

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

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

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

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

*Myrinet*

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

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

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

## Operating Systems

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

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

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

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

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

*LINUX*

Linux is a UNIX-like OS which was initially developed by Linus Torvalds, a Finnish undergraduate student in 1991-92. The original releases of Linux relied heavily on the Minix OS; however, the efforts of a number of collaborating programmers have resulted in the development and implementation of a robust and reliable, POSIX compliant, OS.
Although Linux was developed by a single author initially, a large number of authors are now involved in its development. One major advantage of this distributed development has been that there is a wide range of software tools, libraries, and utilities available. This is due to the fact that any capable programmer has access to the OS source and can implement the feature that they wish. Linux quality control is maintained by only allowing kernel releases from a single point, and its availability via the Internet helps in getting fast feedback about bugs and other problems. The following are some advantages of using Linux:

- Linux runs on cheap x86 platforms, yet offers the power and flexibility of UNIX.
- Linux is readily available on the Internet and can be downloaded without cost.
- It is easy to fix bugs and improve system performance.
- Users can develop or _ne-tune hardware drivers which can easily be made available to other users.

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

*Solaris*

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

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

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

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

*Microsoft Windows NT*

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

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

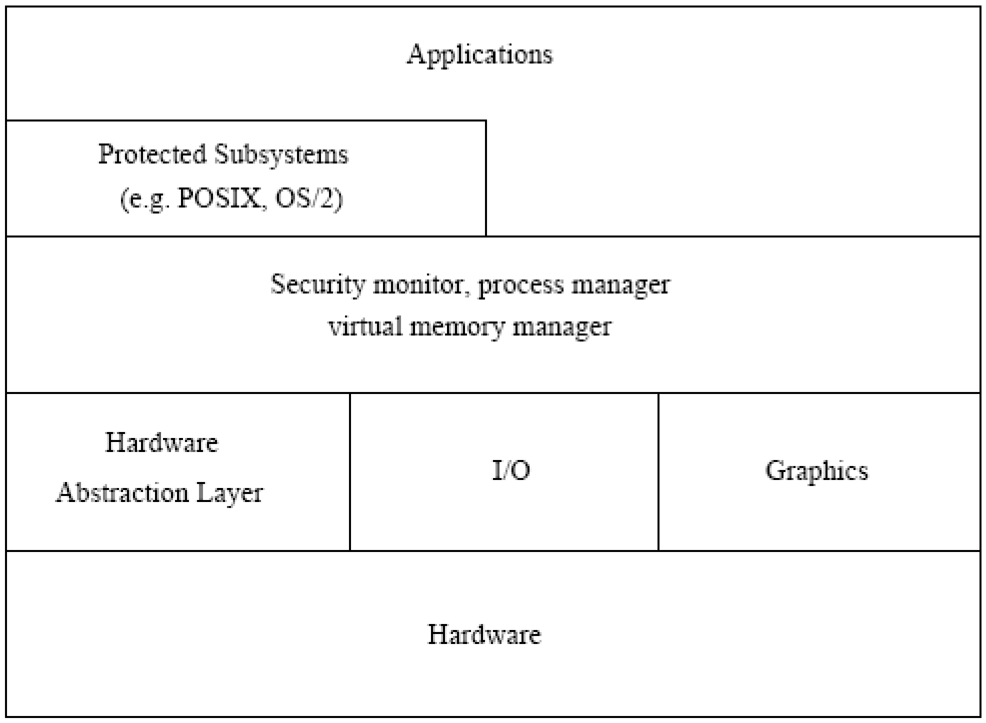

Applications
Protected Subsystems (e.g. POSIX, OS/2)
Security monitor, process manager virtual memory manager
Hardware Abstraction Layer
I/O
Graphics
Hardware

**Figure 5** Schematic Diagram of NT architecture

## Network Services/Communication Software

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

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

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

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

## Cluster Middleware and Single System Image

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

- Single System Image infrastructure.
- System Availability infrastructure

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

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

- It frees the end user from having to know where an application will run.
- It frees the operator from having to know where a resource (an instance of resource) is located
- It does not restrict the operator or system programmer who needs to work on a particular region; the end user interface (hyperlink - makes it easy to inspect consolidated data in more detail) can navigate to the region where a problem has arisen.
- It reduces the risk of operator errors, with the result that end users see improved reliability and higher availability of the system.
- It allows to centralize/decentralize system management and control to avoid the need of skilled administrators for system administration.
- It greatly simplifies system management; actions affecting multiple resources can be achieved with a single command, even where the resources are spread among multiple systems on different machines.
- It provides location-independent message communication. Because SSI provides a dynamic map of the message routing as it occurs in reality, the operator can always be sure that actions will be performed on the current system.

- It helps track the locations of all resources so that there is no longer any need for system operators to be concerned with their physical location while carrying out system management tasks..

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

*Single System Image Levels/Layers*

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

- Hardware (such as Digital (DEC) Memory Channel, hardware DSM, and SMP techniques)
- Operating System Kernel|Underware3 or Gluing Layer (such as Solaris MCand GLUnix)
- Applications and Subsystems|Middleware
    - Applications (such as system management tools and electronic forms)
    - Runtime Systems (such as software DSM and parallel file system)
    - Resource Management and Scheduling software (such as LSF and CO-DINE)

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

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

Hardware Layer

Systems such as Digital (DEC's) Memory Channel and hardware DSM over SSI at hardware level and allow the user to view cluster as a shared memory system. Digital's memory channel, a dedicated cluster interconnect, provides virtual shared memory among nodes by means of inter-nodal address space mapping.

Operating System Kernel (Underware) or Gluing Layer

Cluster operating systems support an efficient execution of parallel applications in an environment shared with sequential applications. A goal is to pool resources in a cluster to provide better performance for both sequential and parallel applications.

To realize this goal, the operating system must support gang-scheduling of parallel programs, identify idle resources in the system (such as processors, memory, and networks), and over globalized access to them. It has to support process migration for dynamic load balancing and fast interprocess communication for both the system and user-level applications. The OS must make sure these features are available to the user without the need of new system calls or commands and having the same syntax. OS kernels supporting SSI include SCO UnixWare and Sun Solaris-MC.

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

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

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

Applications and Subsystems Layer (Middleware)

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

The subsystems offer a software means for creating an easy-to-use and efficient cluster system. Run time systems, such as cluster file systems, make disks attached to cluster nodes appear as a single large storage system. SSI offered by file systems ensures that every node in the cluster has the same view of the data.

Global job scheduling systems manage resources, and enables the scheduling of system activities and execution of applications while offering high availability services transparently.

*SSI Boundaries*

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

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

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

*Middleware Design Goals*

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

Complete Transparency

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

Scalable Performance

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

Enhanced Availability

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

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

*Key Services of SSI and Availability Infrastructure*

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

SSI Support Services

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

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

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

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

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

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

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

Availability Support Functions

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

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

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

## 2.5 Resource Management and Scheduling (RMS)

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

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

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

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

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

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

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

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

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

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

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

## 2.6 Programming Environments and Tools

The availability of standard programming tools and utilities have made clusters a practical alternative as a parallel-processing platform.

### Threads

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

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

## Message Passing Systems (MPI and PVM)

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

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

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

## Distributed Shared Memory (DSM) Systems

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

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

## Parallel Debuggers and Profilers

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

*Debuggers*

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

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

*TotalView*

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

## Performance Analysis Tools

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

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

A particular issue with performance monitoring tools is the intrusiveness of the tracing calls and their impact on the applications performance. It is very important to note that instrumentation affects the performance characteristics of the parallel application and thus provides a false view of its performance behavior.

| **Tool** | **Supports** | **URL** |
|---|---|---|
| AIMS | instrumentation, monitoring library, analysis | http://science.nas.nasa.gov/Software/AIMS |
| MPE | logging library and snapshot performance visualization | http://www.mcs.anl.gov/mpi/mpich |
| Pablo | monitoring library and analysis | http://www.pablo.cs.uiuc.edu/Projects/Pablo/ |
| Paradyn | dynamic instrumentation runtime analysis | http://www.cs.wisc.edu/paradyn |
| SvPablo | integrated instrument or , monitoring library and analysis | http://www.pablo.cs.uiuc.edu/Projects/Pablo/ |
| Vampir | monitoring library performance visualization | http://www.pallas.de/pages/vampir.htm |

**Table 2** Various Performance Analysis Tools

## Cluster Administration Tools

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

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

# 3. Design and Implementation

## 3.1 Introduction

The basic design is a kernel extension for single-system image clustering. Clustering technologies allow two or more Linux systems to combine their computing resources so that they can work cooperatively rather than in isolation. The tool used is for a Unix-like kernel, such as Linux, consisting of adaptive resource sharing algorithms. It allows multiple uniprocessors and symmetric multiprocessors (SMP nodes) running the same kernel to work in close cooperation. The resource sharing algorithms are designed to respond on-line to variations in the resource usage among the nodes. This is achieved by migrating processes from one node to another, preemptively and transparently, for load-balancing and to prevent thrashing due to memory swapping. The goal is to improve the cluster-wide performance and to create a convenient multiuser, time-sharing environment for the execution of both sequential and parallel applications. The standard runtime environment is a computing cluster, in which the cluster-wide resources are available to each node.

The current implementation is designed to run on clusters of X86/Pentium-based uniprocessors workstations that are connected by standard LANs. Possible configurations may range from a small cluster of PCs that are connected by 100Mbps Ethernet, to a high performance system.

## 3.2 The load balancing approach

Basically, the tool includes both a set of kernel patches and support tools. The patches extend the kernel to provide support for moving processes among machines in the cluster. Typically, process migration is totally transparent to the user. However, by using the tools provided with openMosix, as well as third-party tools, you can control the migration of processes among machines.

An illustration of how tool might be used to speed up a set of computationally expensive tasks could be helpful. Suppose, for example, we have a dozen files to compress using a CPU-intensive program on a machine that isn't part of an cluster. We could compress each file one at a time, waiting for one to finish before starting the next. Or we could run all the compressions simultaneously by starting each compression in a separate window or by running each compression in the background (ending each command line with an &). Of course, either way will take about the same amount of time and will load down our computer while the programs are running.

However, if our computer is part of our load balancing cluster, here's what will happen: First, we will start all of the processes running on our computer. With a load balancing cluster, after a few seconds, processes will start to migrate from your heavily loaded computer to other idle or less loaded computers in the clusters. If we have a dozen idle machines in the cluster, each compression should run on a different machine. Our machine will have only one compression running on it (along with a little added overhead) so we still may be able to use it. And the dozen compressions will take only a little longer than it would normally take to do a single compression.

If we don't have a dozen computers, or some of your computers are slower than others, or some are otherwise loaded, the tool will move the jobs around as best it can to balance the load. Once the cluster is set up, this is all done transparently by the system. On the other hand, if we want to control the migration of jobs from one computer to the next, openMosix supplies you with the tools to do just that.

## 3.3 The working

One approach to sharing a computation between processors in a single-enclosure computer with multiple CPUs is *symmetric multiprocessor* (*SMP*) computing. The tool turns a cluster of computers into a virtual SMP machine, with each node providing a CPU. Hence it is potentially much cheaper and scales much better than SMPs, but communication overhead is higher. This could also be termed as *single system image clustering* (*SSI*) since each node in the cluster has a copy of a single operating system kernel.

The granularity for tool is the process. Individual programs, as in the compression example, may create the processes, or the processes may be the result of different forks from a single program. However, if you have a computationally intensive task that does everything in a single process (and even if multiple threads are used), then, since there is only one process, it can't be shared among processors. The best you can hope for is that it will migrate to the fastest available machine in the cluster.

Not all processes migrate. For example, if a process only lasts a few seconds (very roughly, less than 5 seconds depending on a number of factors), it will not have time to migrate. Currently, the tool does not work with multiple processes using shared writable memory, such as web servers. Similarly, processes doing direct manipulation of I/O devices won't migrate. And processes using real-time scheduling won't migrate. If a process has already migrated to another processor and attempts to do any these things, the process will migrate back to its *unique home node (UHN)*, the node where the process was initially created, before continuing.

To support process migration, the processes are divided into two parts or *contexts*. The *user context* contains the program code, stack, data, etc., and is the part that can migrate. The *system context*, which contains a description of the resources the process is attached to and the kernel stack, does not migrate but remains on the UHN.

There is also an adaptive resource allocation policy. That is, each node monitors and compares its own load with the loads on a portion of the other computers within the cluster. When a computer finds a more lightly loaded computer (based on the overall capacity of the computer), it will attempt to migrate a process to the more lightly loaded computer, thereby creating a more balanced load between the two. As the loads on individual computers change, e.g., when jobs start or finish, processes will migrate among the computers to rebalance loads across the cluster, adapting dynamically to the changes in loads.

Individual nodes, acting as autonomous systems, decide which processes migrate. The communications among small sets of nodes within the cluster used to compare loads is randomized. Consequently, clusters scale well because of this random element. Since communications is within subsets in the cluster, nodes have limited but recent information about the state of the whole cluster. This approach reduces overhead and communication.

While load comparison and process migration are generally automatic within a cluster, there are tools to control migration. It is possible to alter the cluster's perception of how heavily an individual computer is loaded, to tie processes to a specific computer, or to block the migration of processes to a computer. However, precise control for the migration of a group of processes is not practical at this time.

The cluster tool uses the values in the flat files in */proc/hpc* to record and control the state of the cluster. If you need information about the current configuration, want to do really low-level management, or write management scripts, you can look at or write to these files.

## 3.4 Components and tools

### Components

There are three main components: process migration, the Mosix File System (MFS) and Direct File System Access (DFSA). These components are described below.

### Process migration

When a system runs the tool a process can be started on one node but it will actually run on another node in the cluster: the process has migrated. Migration means that a process is split in 2 parts, a user part and a system part. The user part can then be moved to a remote node that is faster or have lower load, while the system part will stay on the node that started the process, the so-called unique home node (UHN). The system part is sometimes called the deputy process and takes care of resolving most of the system calls. The communication between the user- and system-parts are taken care of by openMosix. Not all processes can be split, or migrate.

## The Mosix File System (MFS)

MFS is a feature in openMosix that allows the user to access remote file systems in a cluster as if they are locally mounted. The file systems of the other nodes in the cluster can be found mounted under the directory /mfs. For example the files in /var on node 2 can be found in /mfs/2/var on all the machines in the cluster.

## Direct File System Access (DFSA)

The tool provides MFS with the DFSA-option. This option allows remote processes to perform some file system calls locally rather than sending them to their home node.

## Userspace-tools

Userspace-tools is a package of different tools for openMosix. There are a number of useful programs:

- *omdiscd* is a tool to auto-discover new nodes.
- *mps* and *mtop* are openMosix aware version of the Unix tools ps and top.
- *migrate* is a command to manually migrate processes.
- *mosmon* is a terminal program to monitor the status of the different nodes.
- *mosctl* is openMosix main configuration utility.
- *mosrun* is a tool to run a command with certain openMosix settings, for example: make the command run on a certain node. There are several scripts that run the mosmon program with certain settings.
- *setpe* is a manual node configuration utility.
- *tune* is a calibration and optimization utility.

The RPM that is used to install these tools also changes some configuration scripts on the computer so that openMosix runs properly. It also adds an init.d script for openMosix so openMosix is started when the computer is rebooted. This script needs to be disabled and exchanged with omdiscd if auto-discovery is going to be used.

## openMosixview

OpenMosixview is a cluster-management GUI for openMosix. It contains five applications for monitoring and administration of the cluster. These applications are:

- OpenMosixview The main monitoring and administration application.
- OpenMosixprocs A process-box for managing processes on the different nodes.
- OpenMosixcollector A daemon that collects and logs cluster plus node information.

- OpenMosixanalyzer Analyses the data collected by openMosixcollector.
- OpenMosixhistory A process history for the cluster.

All these applications are accessible via the main application and the most common openMosix-commands are also accessible in the same program. Programs with different openMosix settings can also be started in openMosixview. There are priority-sliders for each 7 node so that a system administrator can set the load balancing priorities for the different nodes manually. OpenMosixview has been adapted to work with auto-discovery of nodes.

To be able to administrate openMosix via openMosixview the cluster needs to either have rlogin and rsh or ssh installed. These have to be configured so that the user root can log on all the nodes from the machine running openMosixview without using a password (rlogin and rsh using the .rhosts file and ssh using identity keys and the authorized_keys file). For further security when using ssh a so-called pass-phrase can be set for the identity keys. If the passphrase is set, then the program ssh-agent has to be started before it is possible to log on to the nodes without a password. The QT libraries version 2.30 or above are also needed to install openMosixview

## 3.5 Installation

This section describes the various steps required in installing the tool and configuring it to setup the cluster.

First of all a pure vanilla kernel-source is needed. What also need to be thought of is to use the right openMosix version depending on the kernel version. When the kernel source and the openMosix patch have been downloaded, patch the kernel, configure, compile and install the kernel.

After that, download the openMosix Userland (userspace) tools and unpack the tar.gz-file, edit the configuration file, compile and copy the openmosix startup script. Create a /etc/mosix.map file that lists the computers in the cluster.

Before openMosixview is installed check that QT (and its source files) exists on the system. If it doesn't exist then download and install it. Now it's time to install openMosixview (this only needs to be done on the machine that is going to be used to monitor the cluster). Download and unzip/untar the openMosixview source tar.gz, then run the automatic setup script that comes with the code to configure and compile the different programs. After that openMosixview has been installed, the openmosixprocs program (that comes with openMosixview) should be copied to all the other nodes' */usr/bin* directory so that the processes on these nodes can be administrated via openMosixview. Finally the user root's settings for SSH on the different nodes need to be configured so that a password isn't needed to ssh to the different nodes. This is needed so that openMosixview can change the different openMosix settings on the different nodes.

If we download the source file from the openMosix web site, we'll have an RPM package to install. When we install this, it will place compressed copies of the patches and the source tree (in *gzip* or *bzip2* format) as well as several sample kernel configuration files in the directory */usr/src/redhat/SOURCES*. The next step is to unpack the sources and apply the patches.

Using *gunzip* or *bzip2* and then *tar*, we can unpack the files in the appropriate directory. Moving the patch files into the root directory of source tree, we have all the files in place, we can use the *patch* command to patch the kernel sources.

The next step is to create the appropriate configuration file. In theory, there are four ways we can do this. We could directly edit the default configuration file, typically */usr/src/linux/.config*, or we can run one of the commands *make config*, *make menuconfig*, or *make xconfig*. Figure shows the basic layout with *menuconfig*.

Once we have the configuration file, we are ready to build the image. We use the commands *make dep*, *make clean*, *make bzImage*, *make modules*, and *make modules_install*. If all goes well, you'll be left with a file *bzImage* in the directory *arch/i386/boot/* under your source tree.

The next is to install the kernel, i.e., arrange for the system to boot from this new kernel. After installing kernel, we install the user tools and configure openMosix for our system.

The exact commands used to setup the cluster are detailed in Appendix B.

### Adding nodes to the cluster

There are two ways to add nodes depending on if the auto discovering daemon is used or not. If the auto-discovering daemon is used it’s just to install openMosix with auto-discovering on the new machine and then it will be automatically added. Otherwise the machine has to be added to the file /etc/mosix.map on all the nodes in the cluster.

## 3.6 Administration

The configuration of the cluster exists in files lying in subdirectories in the /proc/hpc directory on the nodes. The flat files in the subdirectories can be changed both manually and via the userspace tools. In this directory there are 5 subdirectories:

- admin Files for setting the different settings that openMosix has. Some files just tell if a setting is on or off for example MFS, others set different variables for example how fast the node should collect load-balancing information (decay statistics).

- decay Keeps the decay statistics settings.

- info Keeps information about the computer in binary format
- nodes Have files with system information about the different nodes on the cluster.
- remote Keeps information about remote processes that has migrated to the node.

There are also files with openMosix settings in the /proc/[PID] directories that keeps information about the processes on the system.

If auto-discovery isn't used then the different nodes should be listed in the /etc/mosix.map file on every node in the cluster, this file can be edited manually or with the setpe utility.

To shutdown openMosix the script /etc/init.d/openmosix which is a common inetd script should be used. For example if openMosix on the node is going to be restarted just type */etc/init.d/openmosix restart* as root[2]. OpenMosix can also be started and stopped in openMosixview.

# 4. Performance Test of Cluster

## 4.1 Test Computers and Programs

The test was performed on three computers running Fedora Core 4 Linux connected via a 100 Mbit full-duplex LAN. The computers were 2 Pentium 4 (HT Technology) 3.0 GHz with 256 MB RAM named node1 and node3 and one AMD Athlon 1800+ with 128MB RAM named node2 Appendix C and E describes the test system and how openMosix was configured on the different test machines.

The program used for the performance test was a token ring simulation. The program runs several simulations, called replications, after each other and then compute different statistical variables. The test was run in such a way that the simulation divided up in smaller jobs (dependent on how many computers were in the test) that ran in parallel on the test cluster. The total number of replications for all the jobs was 600, for example a test with 2 computers ran 2 parallel jobs consisting of 300 replications each.

To get the time that each simulation took to execute the command *time* was used, this command had the accuracy of $10_{-2}$ seconds. The time that was used to compare the different systems was the real time in the system for the small job, of the ones started, in the simulation that got finished last. This time was used since it was that time it took for the entire simulation of 600 replications to finish.

Each test was done 30 times. The test data was then stored and the average, variance and 95% confidence interval for the different systems were calculated. The comparison between the different tests was made with the so called *approximate visual test.*

The tests on the openMosix system was done by creating a small script that started all the small jobs, with the time command in front of each of them. After that openMosix got to take care of the load balancing so that the processes distributed themselves over the computers used in the test.

## 4.2 Tests made

The performance tests were done on 4 different configurations of computers described below:

### 4.2.1 No cluster at all

In this test one simulation with 600 replications were run on the two PC machines. This test was mainly to see how well the clusters were compared to only having one computer to run the program on.

### 4.2.2 Homogeneous cluster

This cluster consisted of the two machines node1 and node2. The tests were made with openMosix.

### 4.2.3 Heterogeneous cluster with two computers

This cluster consisted of the fast Intel machine node3 and the slow AMD machine node2. The tests were done with one test where the script starting the jobs were run on node2 and another where the jobs where started from node3.

### 4.2.4 Heterogeneous cluster with three computers

This test used all the machines (node1, node2 and node3). The tests were done with one test where the script starting the jobs were run on node2 and another where the jobs where started from node1.

## 4.3 Results

The results from the different performance tests are presented. The individual data collected during the tests can be found in Appendix A.

### 4.3.1 No Cluster

In table 3 the average, variance, standard deviation and 95% confidence interval of the data from the performance tests when no cluster existed on the test computers is presented:

| | Node 1 | Node 2 | Node 3 |
|---|---|---|---|
| Average (s) | 469.58 | 1653.74 | 462.44 |
| Variance ($s^2$) | 0.06 | 1.82 | 0.005 |
| Standard Deviation (s) | 0.24 | 1.35 | 0.07 |
| 95% confidence interval (s) | +/-0.08 | +/-0.48 | +/-0.03 |

**Table 3** Statistical values for the results from the performance tests with no cluster

In figures 6, 7 and 8 below are graphs of the test data gotten from the tests together with the average and the confidence boundaries.

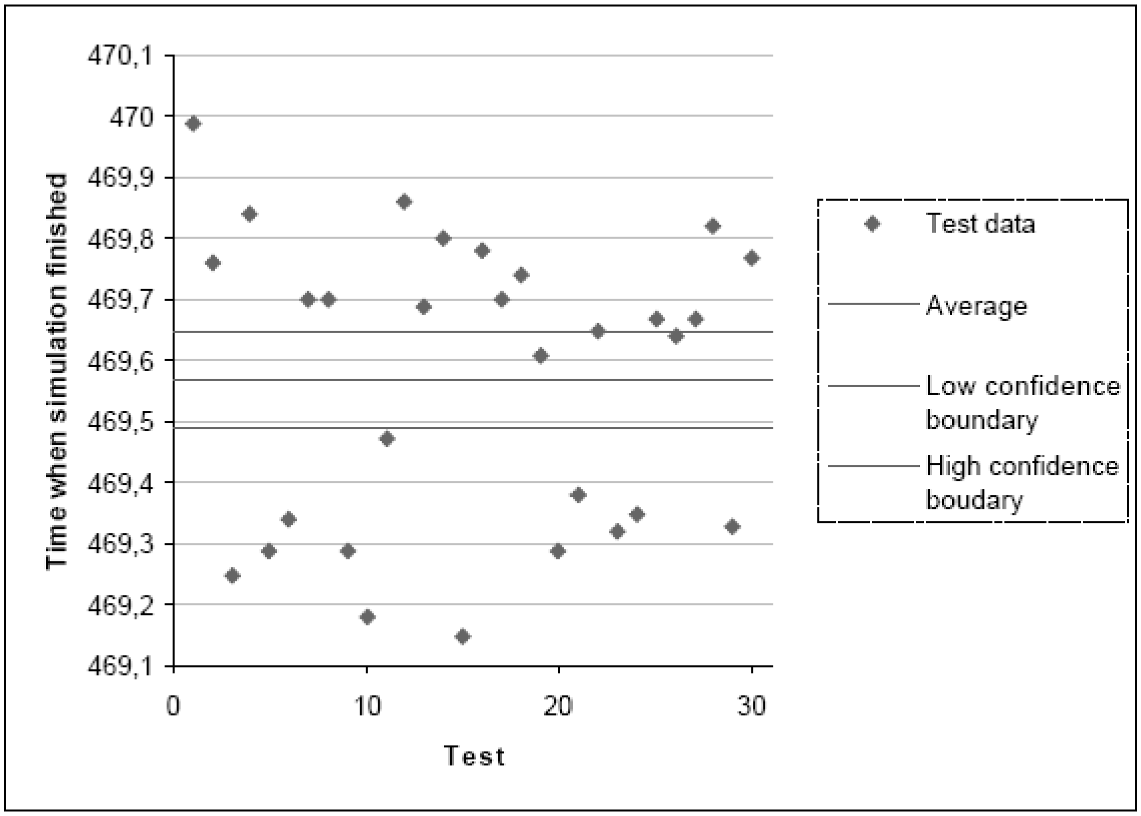


**Figure 6** Graph of test data with average and confidence boundaries with no cluster on node1

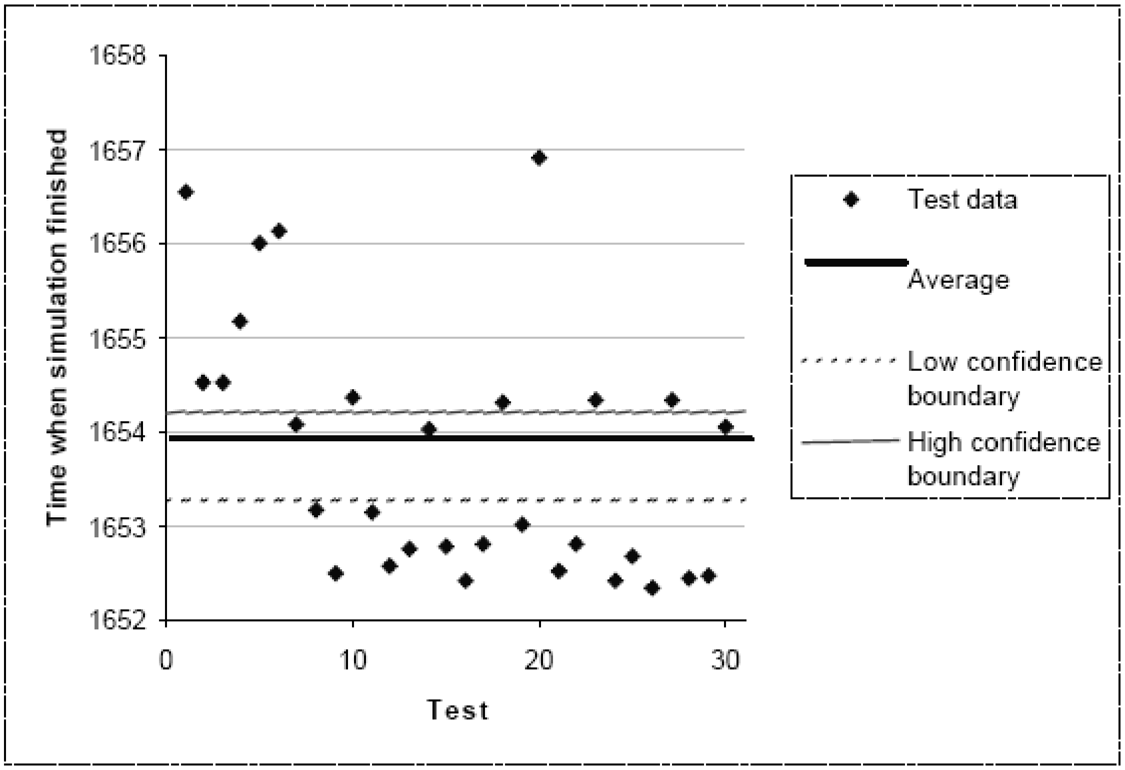


**Figure 7** Graph of test data with average and confidence boundaries with no cluster on node2

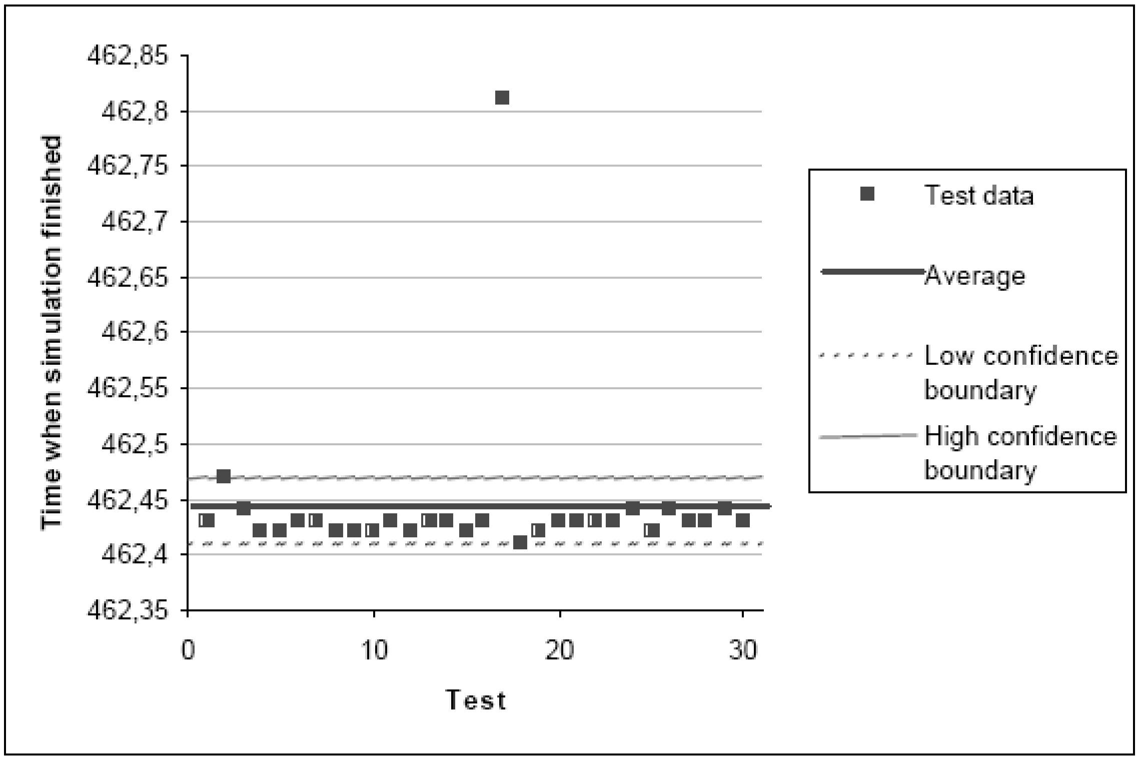


**Figure 8** Graph of test data with average and confidence boundaries with no cluster on node3

That the 17th sample time in figure 8 is higher (by approximately 3-4 ms) than the rest of the sample times probably depends on some other program on the system that took system resources while the test was running. This is also the explanation to the similar differences in time that will be seen in some of the graphs for the sample data for all the other tests in the project.

The confidence values together with the average time are presented in table 4:

| | Node 1 | Node 2 | Node 3 |
|---|---|---|---|
| Low Confidence Boundary | 469.5 | 1653.26 | 462.41 |
| Average | 469.58 | 1653.74 | 462.44 |
| High Confidence Boundary | 469.66 | 1654.22 | 462.47 |

**Table 4** Confidence values (95% confidence interval) for simulation with no cluster

The times for node1 are higher than for node3 even though they have the same hardware. This depends on that node1 is running additional software that takes system resources.

### 4.3.2 Homogenous Cluster

In table 5 the average, variance, standard deviation and 95% confidence interval of the data from the performance tests when the two machines node1 and node3 were connected in a cluster, is presented:

| | OpenMosix |
|---|---|
| Average (s) | 237.3 |
| Variance ($s^2$) | 0.28 |
| Standard Deviation (s) | 0.53 |
| 95% confidence interval (s) | +/-0.19 |

**Table 5** Statistical values for the results from the performance tests with homogenous cluster

In figure 9 is graph of the test data gotten from the performance test together with its average and the confidence boundaries.

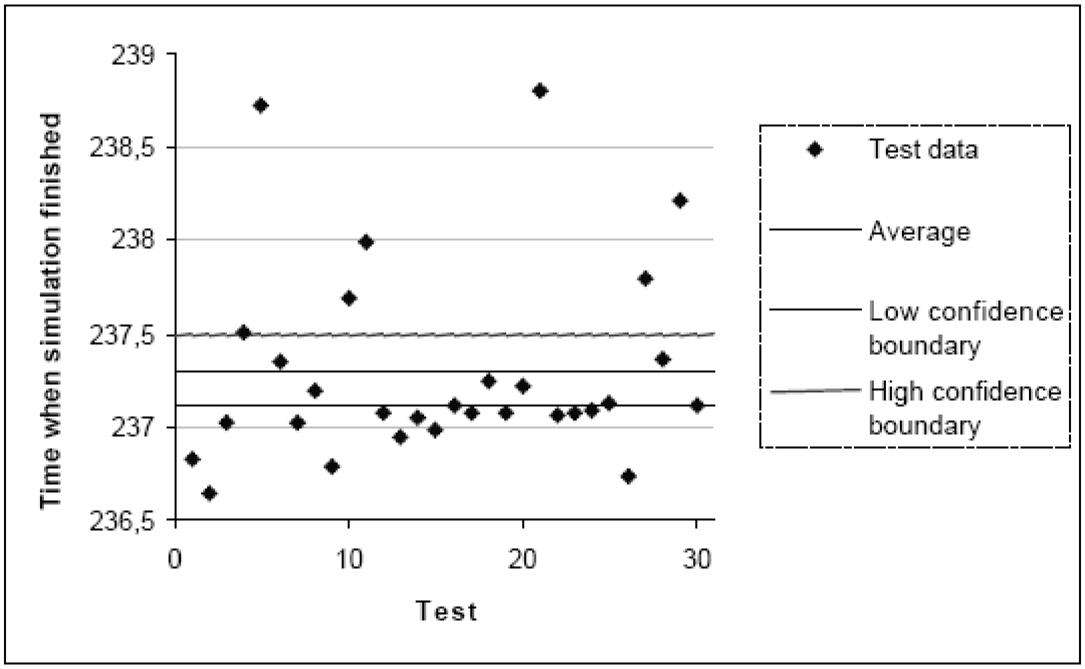


**Figure 9** Graph of test data with average and confidence boundaries on homogeneous openMosix cluster

### 4.3.3 Heterogeneous cluster with two computers

In table 6, the statistics of the tests when the fast machine node3 and the slow machine node2 were connected in a cluster. *Started on node2/node3* in the table means that the simulation-processes were started on that node and then openMosix migrated them between the machines if the cluster system saw the need for it.

|  | OpenMosix started on node 2 | OpenMosix started on node 3 |
|---|---|---|
| Average (s) | 469.99 | 462.81 |
| Variance ($s^2$) | 0.97 | 0.03 |
| Standard Deviation (s) | 0.99 | 0.16 |
| 95% confidence interval (s) | +/-0.35 | +/-0.06 |

**Table 6** Statistical values for the results from the performance tests with heterogeneous cluster.

In figures 10 and 11 are plots of the test data together with their average and confidence boundaries.

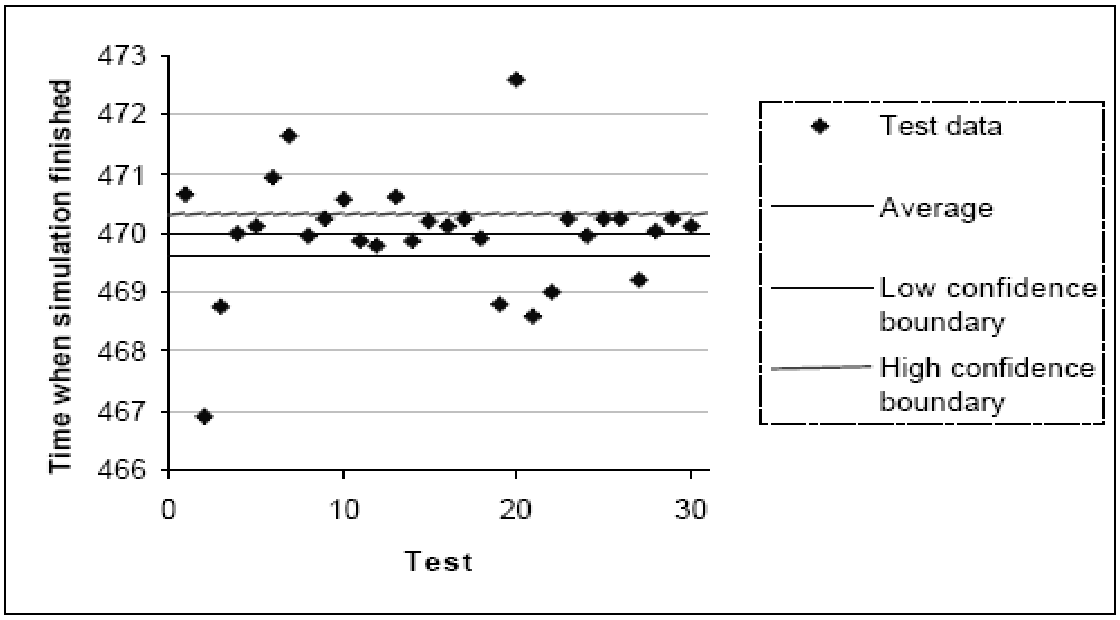


**Figure 10** Graph of test data with average and confidence boundaries on heterogeneous openMosix cluster with 2 computers with simulations started from node2

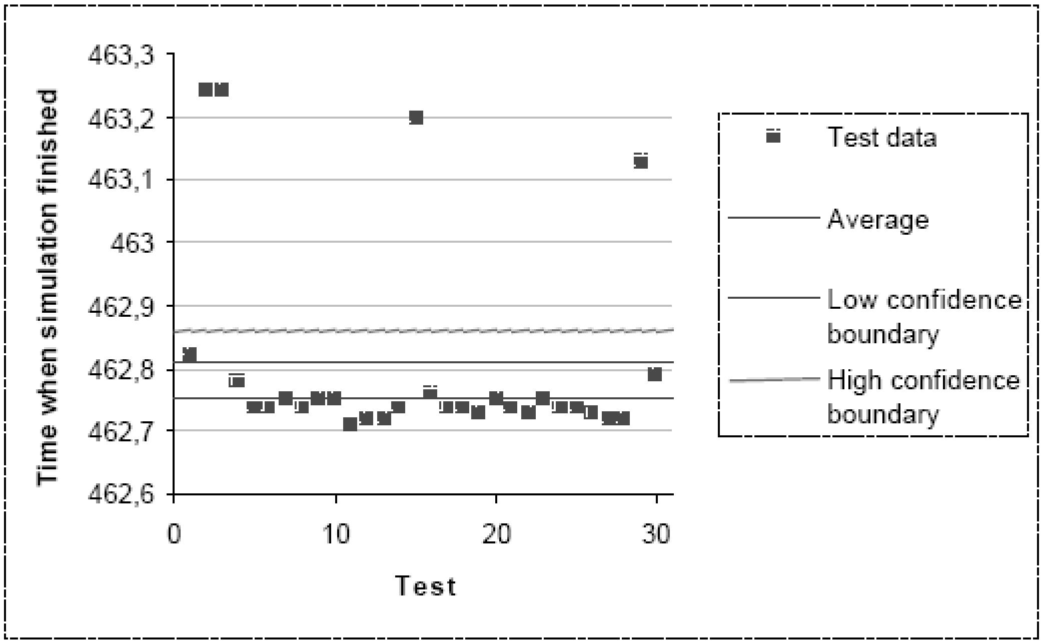


**Figure 11** Graph of test data with average and confidence boundaries on heterogeneous openMosix cluster with 2 computers with simulations started from node3

The confidence values together with the average time are presented in table 7:

| | OpenMosix started on node 2 | OpenMosix started on node 3 |
|---|---|---|
| Low Confidence Boundary | 469.63 | 462.75 |
| Average | 469.99 | 462.81 |
| High Confidence Boundary | 470.34 | 462.86 |

**Table 7** Confidence values (95% confidence interval) for a heterogeneous cluster with 2 computers.

OpenMosix from node3 are better than openMosix from node2 but just with a few seconds as shown in figure 12 below.

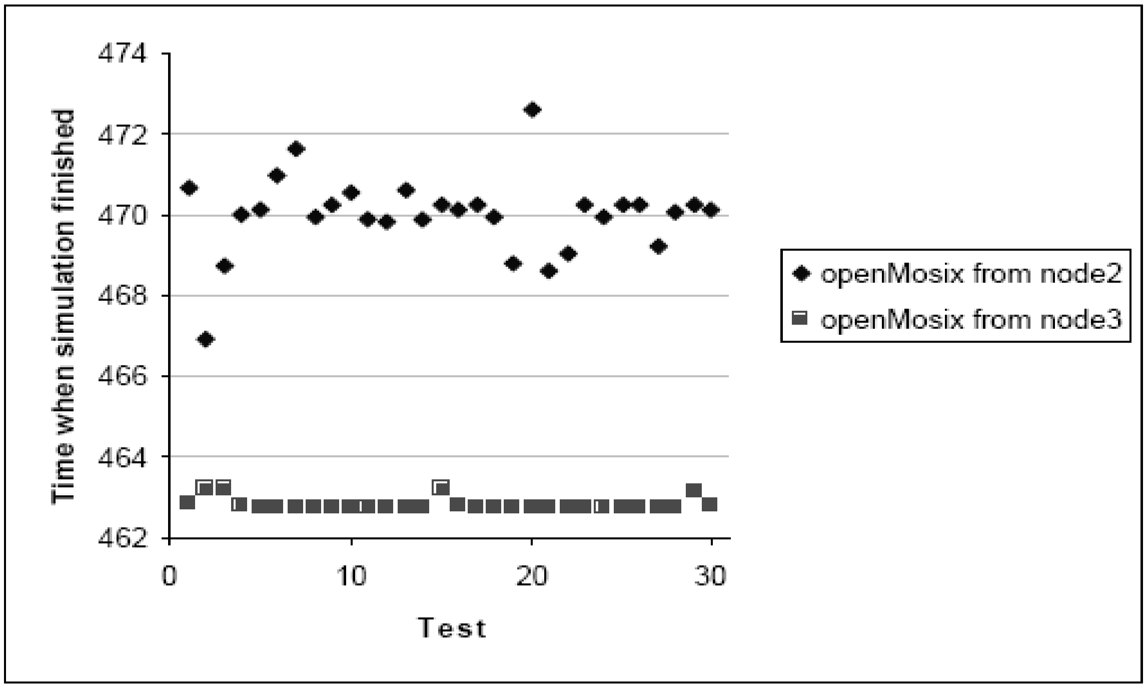


**Figure 12** Graph test data for openMosix from node2 and node3 on a heterogeneous cluster with 2 computers,

### 4.3.4 Heterogeneous cluster with three computers

In table 8 the average, variance, standard deviation and 95% confidence interval of the data gotten from tests when all the 3 PC machines (the two fast machines node 1 and 3 plus the slow machine node2) was connected in a cluster, is presented:

| | OpenMosix started on node 2 | OpenMosix started on node 1 |
|---|---|---|
| Average (s) | 252.00 | 246.17 |
| Variance ($s^2$) | 26.53 | 38.15 |
| Standard Deviation (s) | 5.15 | 6.18 |
| 95% confidence interval (s) | +/-1.84 | +/-2.21 |

**Table 8** Statistical values for the results from the performance tests with heterogeneous cluster of 3 computers.

In figures 13 and 14 are graphs of the test data gotten from the performance tests together with the average and their confidence boundaries.

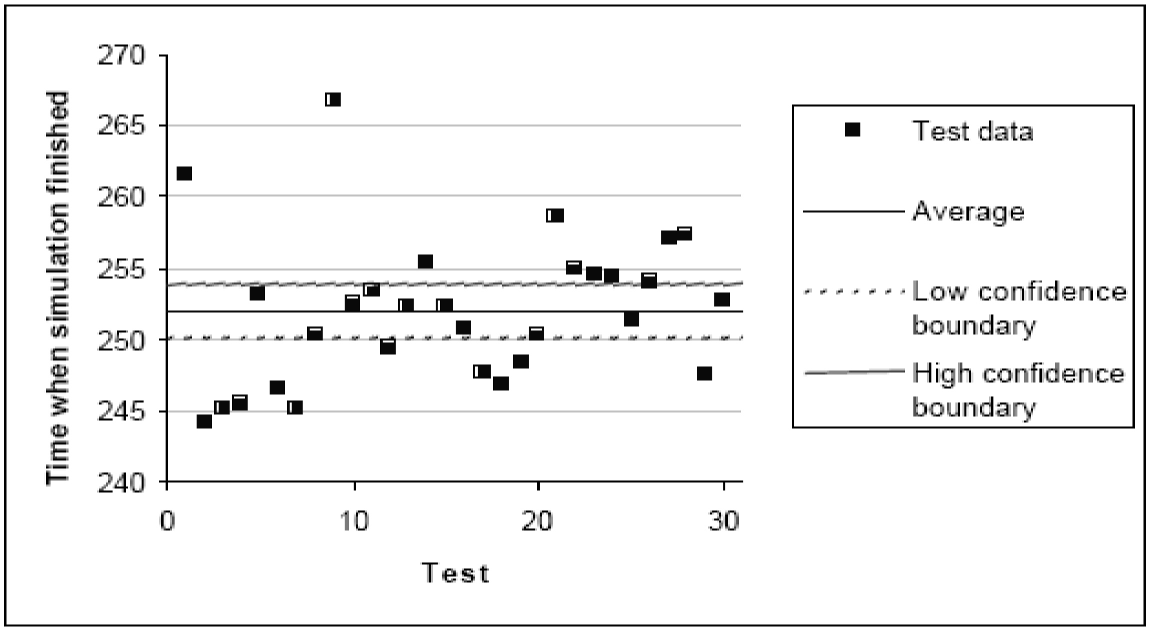


**Figure 13** Graph of test data with average and confidence boundaries on heterogeneous openMosix cluster with 3 computers with simulations started from node2,

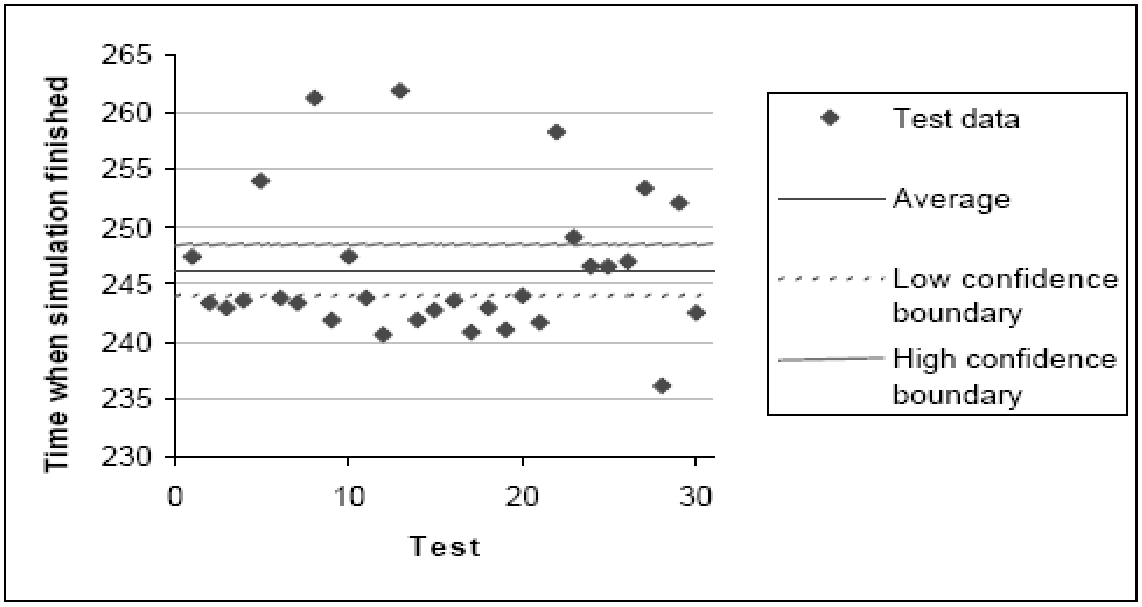


**Figure 14** Graph of test data with average and confidence boundaries on heterogeneous openMosix cluster with 3 computers with simulations started from node1,

The confidence values together with the average times are presented in table 9

| | OpenMosix started on node 2 | OpenMosix started on node 1 |
|---|---|---|
| Low Confidence Boundary | 250.15 | 243.96 |
| Average | 251 | 246.17 |
| High Confidence Boundary | 253.84 | 248.38 |

**Table 9** Confidence values (95% confidence interval) for a heterogeneous cluster with 3 computers.

## 4.4 Summary of Tests

In table 10 the average, variance and confidence intervals for the tests when no cluster was used is presented:

| | Node 1 | Node 2 | Node 3 |
|---|---|---|---|
| Average (s) | 469.58 | 1653.74 | 462.44 |
| Variance ($s^2$) | 0.06 | 1.82 | 0.005 |
| 95% confidence interval (s) | +/-0.08 | +/-0.48 | +/-0.03 |

**Table 10** Average, variance and confidence interval when no cluster is used.

The best time is when the simulation was run on node3 since this was one of the fast machines with the smallest amount of different daemons running on it. If the time for a simulation on a cluster is faster than the time it took on node3 alone then the cluster system should be considered.

In table 11 the average, variance and confidence intervals for all the tests with openMosix is presented.

| | Homogenous cluster | Heterogeneous cluster with 2 computers | Heterogeneous cluster with 3 computers |
|---|---|---|---|
| Average | 237.3 | 462.81 | 246.17 |
| Variance | 0.28 | 0.97 | 38.15 |
| 95% confidence interval | +/-0.19 | +/-0.06 | +/-2.21 |

**Table 11** Average, variance and confidence interval for openMosix clusters.

The best times are from the homogeneous cluster but the heterogeneous cluster with 3 computers isn’t much worse. When adding more computers to a heterogeneous cluster the times for the tests to finish on the cluster will get better. That the time got better might also depend on that the extra machine in the heterogeneous cluster was a fast one and therefore the jobs got migrated to it too (instead of the jobs only running on one machine that seemed to be the case with the heterogeneous cluster with 2 computers). If the third machine had been of the same hardware as the slower machine, then the time might not have been much better than for a homogeneous cluster with 2 computers since the slower machines priority in the cluster is much lower than the priority for the faster machines.

## Comparison with other clusters

The cluster developed by us was put to test against the cluster[14] developed by Department of Telecommunication and Signal Processing, Blekinge Institute of Technology. The cluster developed by them uses high performance Beowulf-like architecture being run over by the use of Ganglia cluster management tool. There results with similar tests are registered and can be viewed publicly on the websites for cluster benchmarking[15][16].

The following table shows the results obtained for the cluster from Blekinge Institute running similar tests:

| | Homogenous cluster | Heterogeneous cluster with 2 computers | Heterogeneous cluster with 3 computers |
|---|---|---|---|
| Average | 231.99 | 827.15 | 555.3 |
| Variance | 0.09 | 0.24 | 0.23 |
| 95% confidence interval | +/-0.11 | +/-0.17 | +/-0.17 |

**Table 12** Average, variance and confidence interval for Ganglia clusters.

When it comes to hardware and operating systems Ganglia’s monitoring core works on a lot more systems than Ganglia's Execution environment and openMosix who only work on Intel x86 and Athlon machines running Linux..The Ganglia execution environment might be ported in the future but there is no information about when or if this will be done.

When it comes to adding nodes Ganglia's auto-discovery is of great help. But since openMosix also have a sort of auto-discovery the difficulty in adding nodes to the systems isn't that different between them either.

When it comes to administration of the systems, all of Ganglia's settings exist in two files (one for gmond and one for gmetad). OpenMosix has a directory structure with administrative information that can either be edited manually or via the tools that exists to openMosix. Monitoring of the systems are taken care of by a web interface on Ganglia, while openMosix uses openMosixview together with openMosixcollector or mosmon if X windows can't be used. Ganglia's monitoring system can monitor more things about the computers than openMosix (and new things to monitor can also be added). There is a possibility to use Ganglia together with openMosix. If the tools in Ganglia’s monitoring core are needed they can be used without too much interference in openMosix operations. When it comes to running programs on the two systems openMosix is better than Ganglia, for parallel execution of programs. Ganglia's execution environment is good for running the same program on several machines at the same time while openMosix can send different sub processes to other machines. There are certain programs that can't migrate to other machines on an openMosix system mainly because of shared memory. Of course if the user really just want to run one instance of the same program on all the machines then Ganglia is better but this is mostly not the case if a system for calculations and simulations are wanted. Another thing with openMosix is that the binary for the program that is going to be executed only needs to be on the machine that the program is started from while in Ganglia the binary needs to be on all the machines that gexec are going to start up the programs on. Finally another positive thing with openMosix is that there are several ways to make (or recode) the programs that the user wants to run parallel on the cluster.

When it came to the parallel performance tests, openMosix is considered be the best. Ganglia is better than openMosix on a homogeneous cluster but on a heterogeneous cluster openMosix work better.

# 5. Conclusion and Future Scope

In this dissertation, a complete framework is provided to facilitate the development of clusters using load balancing approach. The load balancing approach is provided by using the openMosix architecture of developing a cluster. The advantage of using openMosix is that it could easily parallelize those processes which are not specifically designed according to openMosix architecture. Hence the widely used applications and programs which are designed for general purpose users could also be successfully run over cluster and hence desired performance could be achieved. The initial results of the tests performed over the system shows that the approach is surely positive.It is also cost effective since deploying of old systems which are otherwise dumped could hence be used to develop the cluster or a high performance computer. This system could then be deployed for the use in laboratories which run processing intensive applications thus accelerating the work and achieving better results.

The current system is still in its infant stage and hence steps need to be taken to make it a full grown system. This system was merely developed to calculate the feasibility of such a system. The system could now thus be developed over a large scale using this approach.

The algorithm used for migration in openMosix could further be improved by reducing its utilization of the resources in network. This could be achieved by spreading the information required by the process, running in cluster, over whole of the system. This will thus remove the existing stream required from remote node to UHN and thus saving the resources.

Further, improvement could be achieved by deploying as much processors as possible and pooling them under one rack. This could help in utilizing the old unused systems and thus achieving cost effectiveness in designing the cluster.

# Appendix A

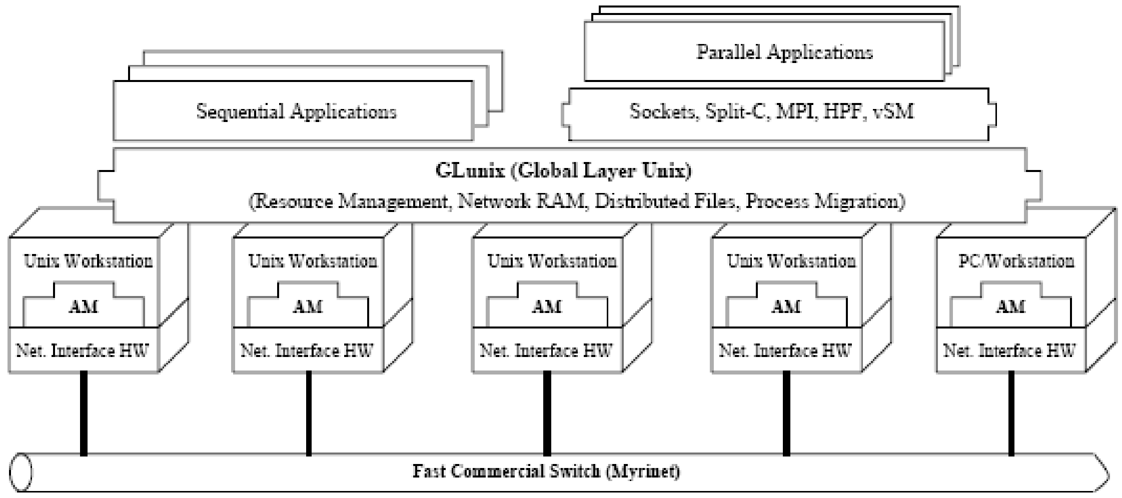
## Representative Cluster Systems

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

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

*The Berkeley Network Of Workstations (NOW) Project*

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



Interprocess Communication

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

Global Layer Unix

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

Network RAM

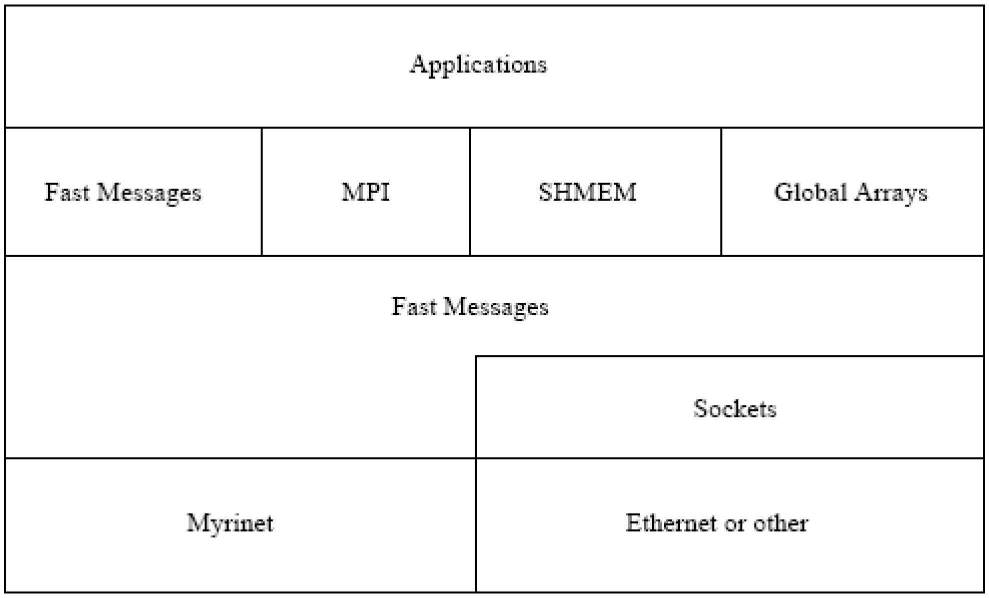
Network RAM allows us to utilize free resources on idle machines as a paging device for busy machines. The designed system is serverless, and any machine can be a server when it is idle, or a client when it needs more memory than physically available. Two prototype systems have been developed. One of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

xFS: Serverless Network File System

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

*The High Performance Virtual Machine (HPVM) Project*

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



The HPVM project aims to address the following challenges:

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

A critical part of HPVM is a high-bandwidth and low-latency communications protocol known as Fast Messages (FM), which is based on Berkeley AM. Unlike other messaging layers, FM is not the surface API, but the underlying semantics. FM contains functions for sending long and short messages and for extracting messages from the network. The services provided by FM guarantees and controls the memory hierarchy that FM provides to software built with FM. FM also guarantees reliable and ordered packet delivery as well as control over the scheduling of communication work.

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

*The Beowulf Project*

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

System Software

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

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

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

*Solaris MC: A High Performance Operating System for Clusters*

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

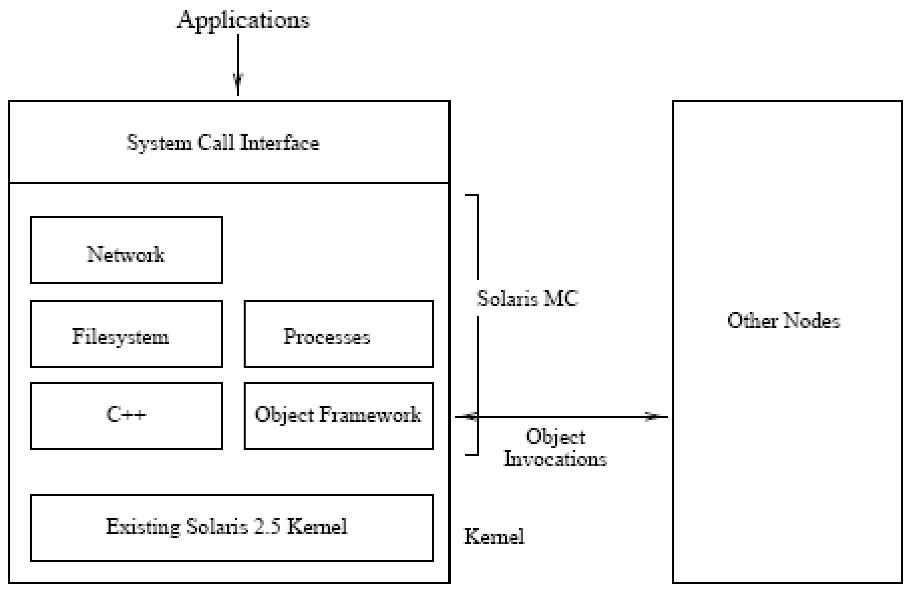
- Extends existing Solaris operating system

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

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

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

The second important component of Solaris MC supporting a single system image is its globalized process management. It globalizes process operations such as signals. It also globalizes the /proc file system providing access to process state for commands such as 'ps' and for the debuggers. It supports remote execution, which allows to start up new processes on any node in the system. Solaris MC also globalizes its support for networking and I/O. It allows more than one network connection and provides support to multiplex between arbitary the network links.



*A Comparison of the Four Cluster Environments*

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

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

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

Each system consists of some middleware interfaced into the OS kernel, which is used to provide a globalization layer, or unified view, of the distributed cluster resources. Berkeley NOW uses the Solaris OS, whereas Beowulf uses Linux with a modified kernel and HPVM is available for both Linux and Windows NT. All four systems provide a wide variety of tools and utilities commonly used to develop, test, and run parallel applications. These include various high-level APIs for message passing and shared-memory programming.

# Appendix B

## How-to for DCEwulf

This how-to describes how to install and administrate cluster using openMosix plus how to use some of the tools that accompany it.

### Requirements

The basic hardware requirements are at least 2 network-connected computers, the faster the network the better performance you will get. The computers also have to have either Intel x86 or Athlon processors.

The system needs a basic Linux installation of any distribution though this how-to is written with RedHat in mind. The network cards needs to be properly configured and you need a quite a lot of swap-space. You have to have the source for 2.6.* kernel. The openMosix tool omdiscd also needs IP multicast support enabled in the kernel.

OpenMosixview needs X, QT 2.3.0 or above, ssh (or rlogin and rsh) on all the nodes (not only the one with openMosixview on it) and the User-space tools installed.

### Installation

*Compiling the openMosix Kernel*

You'll always have to use a pure vanilla kernel-sources to compile an openMosix kernel. The kernel source also needs to be the same version as the openMosix kernel patch you are going to use. For example if you are using openMosix 2.6.15 you have to use the 2.6.15 kernel.

Download the correct kernel source from *ftp://ftp.kernel.org/pub/linux/kernel/v2.6/* and unpack the kernel with: `tar -xzvf linux-2.6.15.tar.gz`

Apply the patch by going into the kernel source directory and then type the following command (if the kernel patch is in the same directory as where you downloaded the kernel source): `zcat ../openMosix-2.6.15.gz | patch -p1`

Now type: `make menuconfig` and enable the openMosix-options in the configuration programs. If the auto discovery daemon is going to be used then IP multicast also has to be enabled. Do also look through the other options shown and configure the kernel the way you want it.

When you exited the configuration utility its time to compile the kernel via:

```
make dep
make bzImage
make modules
```

And then install the kernel with:

```
make modules_install
make install
```

Edit your boot- loader and reboot.

There might show up problems when you try to reboot the kernel. There are some problems with certain settings in the vanilla kernel on RedHat systems. If the new openMosix kernel doesn't boot when reboot the machine with the old kernel and experiment by removing different options in the kernel.

*Compiling openMosix userland tools*

Unpack the openMosix userland source: `tar -xzvf openMosixUserland-0.2.4.tgz`

Go into the directory that is created and edit the file configuration. The lines you only need to touch (most of the times) are (for paths use only **absolute** paths):

| | |
|---|---|
| OPENMOSIX | Shows where the openMosix kernel is. |
| INSTALLDIR | Which is where the system base directory is (usually /). |
| INSTALLEXTRADIR | Where the applications are installed (/usr/local or /usr) |
| INSTALLMANDIR | Is where the man pages should be |
| CC | What C compiler should be used (usually gcc). |
| MONNAME | The filename that openMosix monitoring tool should be called. The recommended name is mosmon (the name used for the program in the rpm). |

When this is done it’s time to compile the programs. You do this by typing: *make all*

After this copy the file *openmosix* in the scripts directory to */etc/init.d* and if you're not going to use the auto discovery tool make so that openMosix is started on reboot by the command: `chkconfig --add openmosix`

If auto discovery is going to be used then the openmosix script shouldn't be added with chkconfig. Instead install the auto discovery by going into the *autodiscovery* directory that lies in the directory with the userland tools source, then remove the line "#define ALPHA" from the files *openmosix.c* and *showmap.c*. After that run the commands: *make clean* and *make*

To make the auto discovery daemon to run on reboot, create the following script

```
#!/bin/bash
#
# chkconfig: 2345 95 5
# description: omdiscd is a tool to automatically discover openMosix nodes.
#
# omdiscd Script to stop/start omdiscd
# Source function library.
[ -f /etc/rc.d/init.d/functions ] || exit 0
. /etc/rc.d/init.d/functions
RETVAL=0
#
# The pathname substitution in daemon command assumes prefix and
# exec_prefix are same. This is the default, unless the user requests
# otherwise.
#
case "$1" in
start)
echo -n "Starting omdiscd: "
daemon /sbin/omdiscd # default interface is eth0 if another network inteface should
# be used use the -i option (ex: /sbin/omdiscd -i eth1)
RETVAL=$?
echo
[ $RETVAL -eq 0 ] && touch /var/lock/subsys/omdiscd
;;
stop)
echo -n "Shutting down omdiscd: "
killproc omdiscd
RETVAL=$?
[ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/omdiscd
echo
;;
6
status)
status omdiscd
;;
restart)
$0 stop
$0 start
;;
*)
echo "Usage: $0 {start|stop}"
exit 1
;;
esac
```

(call the file for example *omdiscd*) in */etc/init.d* and then add it so it gets started at reboot with the command: `chkconfig --add omdiscd`

*Compiling openMosixview*

When you compile openMosixview on the machine you want to run it on you will not only need the QT libraries (that comes with the ordinary QT RPM) but you also need the QT source files available on the system. To install QT from source download the source tar.gz and then unpack it:

```
tar –xvzf qt -x11-free-3-0.5.tar.gz
```

After this you should move the entire directory created to */usr/local* with the new name *qtx.y.z* (there x.y.z is the QT version):

```
mv qt -x11-free-3-0.5 /usr/local/qt-3.0.5
```

When this is done go into the new directory:

```
cd /usr/local/qt-3.0.5
```

Before you run the configure script you have to set the QTDIR variable so that the script find the source files.

```
export QTDIR=”/usr/local/qt-3.0.5”
```

Thereafter run the configure script and compile (*make* will take quite a while):

```
./configure
make
make install
```

Now you will be able to compile openMosixview. Begin by going to the directory where you downloaded the openMosix tar.gz and unpack the downloaded openMosixview source code:

```
tar -xzvf openmosixview-1.2.tar.gz
```

Then go to the directory created and execute the automatic setup-script that comes with the source:

`./setup` *[your_qt_2.3.x_installation_directory]*

When openMosixview is installed copy *openmosixprocs* to */usr/bin/* on all the nodes in /usr/bin directory.

The next step is to configure SSH so that passwords aren't needed when different features in openMosixview is used. First you have to create a RSA key-pair on all the nodes this you do with the command

`ssh-keygen`.

If you use SSH1 you have to write `ssh-keygen -t rsa1` and if you use SSH version 2 you just change *rsa1* to *rsa*. The program will prompt you for what files you want the keys to be in (use the default) and then it will ask you for a pass phrase. You don't need a pass phrase but if you decide to have one you need to use the program *ssh-agent* before you run openMosixview.

The key pairs will be saved in two files in the */root/.ssh* directory: *identity* (private key) and *identity.pub* (public key) for SSH1 or *id_rsa* (private key) and *id_rsa.pub* (public key) for SSH2.

Now copy the entire content in the public key files on the machines that openMosixview has been installed on into */root/.ssh/authorized_keys* on all the nodes in the cluster. In the end all the nodes (both the ones with openMosixview and the ones with just openMosixprocs) should have the public keys of all the nodes with openMosixview in */root/.ssh/authorized_keys*. If the machine has openMosixview on it then its own key should also be in this file

**Administrating openMosix**

The openMosix cluster can be configured by using the */proc/hpc* interface and the user-space tools. OpenMosixview can also be used to configure certain settings and to monitor the cluster.

*The /proc/hpc interface*

In */proc/hpc* there is a number of sub directories with different openMosix settings and statistics. Some of the files in these directories can be set manually others just show statistics and is changed by the system.

The directories in */proc/hpc* are:

*admin* - that presents the current configuration of the system.
*decay* - shows the decay statistics (load balancing information settings).
*info* - shows information about the computer (in binary format).
*nodes* - has information about the different nodes in the cluster.
*remote* - has information about the remote processes that has migrated to the computer

There are also files with openMosix information for a specific process in the */proc/[PID*] directories (where [PID] is the process ID for the process). To view the setting in any of the existing files just use cat, more, less or similar UNIX command. The only directories where the files can be manually edited is */proc/hpc/admin* and */proc/hpc/decay*. You edit these files with the echo command. For example if you want to block the machine, so it doesn't accept remote processes migrating to it, you type:

```
echo 1 > /proc/hpc/admin/block
```

*User-space tools*

The user-space tools have a number of commands that can be used to administrate the cluster. The commands aren't fully described here, for more information read the commands man file with `man` *[command]*

*/etc/init.d/openmosix* is the script that is used to start and stop openMosix. The options to this script are *start*, *stop* and *status*.

The command `migrate` tries to send a process to another machine. The command's syntax is:
`migrate` *[PID] [MOSIX_ID | home | balance]*

The last option can be either a nodes openMosix ID, if the process should go to the machine with the lowest load (balance) or if it should migrate to its home node (home).

The command `mosmon` is a small monitor program that displays a bar chart, normally showing the loads on the various openMosix nodes. Alternatively, it can display the various processor speeds, or the amount of available vs. total memory. The program has an online help that you can get if you type h in the program, to quit just type q. Some of the keys that can be pressed in mosmon to display different variables are:

s shows processor speeds
m shows memory (used out of total)
r shows memory (raw used/free out of total)
u shows utilizability percentage
l go back to showing loads
d show also dead (configured but not-responding) nodes
D stops showing dead nodes
y shows the yardstick in use (e.g. speed of a standard processor)

If all the nodes doesn't fit one screen you can use the left/right arrow keys to move one node to the left or right and the n or p keys to move one screen to the left or right.

The command `mosctl` is the main configuration tool. With this command you can set and read the different settings that exists in the */proc/hpc/* interface. The settings are changed by that. You type the command `mosctl` with different options. The syntax for mosctl is:

`mosctl` *{ stay | nostay | lstay | nolstay | block | noblock | quiet | noquiet | nomfs | mfs | expel | bring | gettune | getyard | getdecay }*

When the command is used with this syntax certain settings (mainly in */proc/hpc/admin*) is set or in the case of the 3 last ones shown. The options mean:

- *stay* sets so that there won't be any automatic process migration (is cancelled with *nostay*).
- *lstay* makes so that local processes doesn't migrate to other nodes but remote processes do (is cancelled by *nolstay*).
- *block* blocks arriving guest processes (is cancelled by *noblock*).
- *quiet* disables gathering of load-balancing information (this is enabled again by *noquiet*).
- *nomfs* disables the MFS (this is enabled again by *mfs*).
- *expel* sends away all the guest processes.
- *bring* brings all the migrated processes home
- *gettune* shows the current overhead parameters used by the kernel to estimate the "I/O factor" in its load-balancing.
- *getyard* shows the current yardstick (the processor speed of the most typical openMosix node).
- *getdecay* shows the current decay parameter (controls the gradual decay of old process statistics for the use of load-balancing).

`mosctl whois` *[ OpenMosix-ID | IP-address | hostname ]* Resolves openMosix ID, IP-addresses and hostnames of the cluster.

`mosctl` *{ getload | getspeed | status | isup | getmem | getfree | getutil } [ OpenMosix-ID ]* Displays different status parameters on a certain node (defined by its openMosix ID).

- *getload* displays the current (openMosix-) load
- *getspeed* displays the current (openMosix-) speed.
- *status* displays the current status and configuration (stay, lstay, block, quiet).
- *isup* tells if a node is up or down.
- *getmem* shows the logical free memory.
- *getfree* shows physical free memory.
- *getutil* displays the utilization.

`mosctl setyard` *[ processor-type | number | this ]* Sets a new yardstick value (can be good if the majority of the nodes are very different from the standard yardstick).

`mosctl setspeed` *[numeric-value ]* Overrides the node's idea of its own speed.

`mosctl setdecay` *[ interval ] [ slow ] [ fast ]* Sets a new decay interval: *interval* in seconds, how much of 1000 to keep for *slow*-decaying processes and how much of 1000 to keep for *fast*-decaying processes. The interval must be within the range of 1-65535 and the slow and fast parameters in the range of 1-1000 (there the slow parameter is greater or equal to the fast parameter).
.
The command `mosrun` is used to execute jobs on the cluster with special openMosix settings on a specific node (or nodes). There are also several scripts that can be used to execute a job with a special pre-configured openMosix configuration. The scripts are: *nomig*, *runhome*, *runon*, *cpujob*, *iojob*, *nodecay*, *slowdecay* and *fastdecay*. Mosrun isn't needed to run processes that uses the openMosix migration but is mainly there if you want to control how the process you are starting should behave.

The command `setpe` is used to configure what nodes are in the cluster by reading a specified file (`setpe -w -f` *[filename]*). You can also read the current configuration with the program (`setpe -r`) plus shutting down openMosix by removing the configuration (`setpe -off`).

The commands `mtop` and `mps` are openMosix aware versions of the standard Linux commands `top` and `ps`. These commands have an extra column that shows what node the programs are running on (in a column marked *n#*). Only the programs that started on the node that mtop/mps is running on is shown and programs that hasn't migrated to another node has node number 0. The options used by these two programs are the same as the ones used for ordinary ps/top.

The tool *omdiscd* is an auto discovery daemon that can be used instead of */etc/mosix.map*. If you start omdiscd openMosix has to be shutdown:

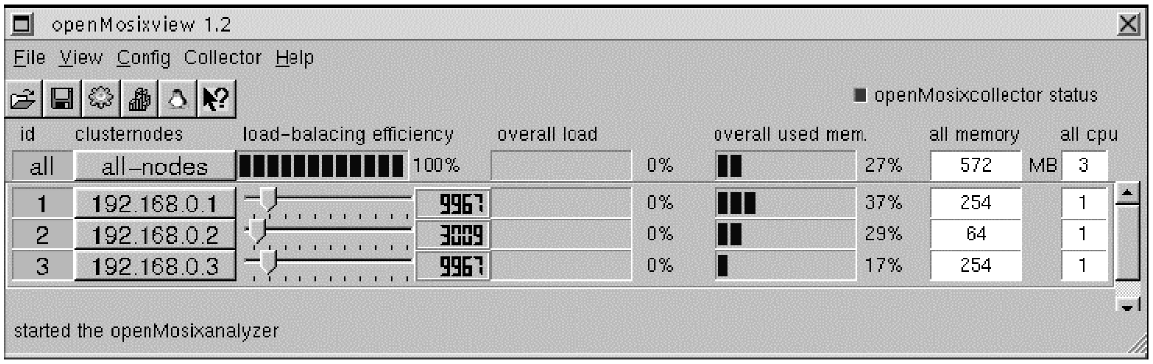
```
s/etc/init.d/openmosix stop
```

Then you start the auto discover daemon with the command `omdiscd`. If the machine has more than one network interface the -i option should be used. You can also start the daemon so it runs in the foreground sending messages and debugging output to standard error. The option is -n and without it all output should go to the syslog.

The auto discovery daemon comes with another command called `showmap`. This command shows the current nodes in the cluster.

*openMosixview*

The tool openMosixview is a graphical interface for doing certain administration and monitoring tasks on an openMosix cluster. OpenMosixview consists of five programs there among a main program (called just openMosixview) that is used to access the other four.



| id | clusternodes | load-balacing efficiency | overall load | overall used mem. | all memory | all cpu |
|---|---|---|---|---|---|---|
| all | all-nodes | 100% | 0% | 27% | 572 MB | 3 |
| 1 | 192.168.0.1 | 9967 | 0% | 37% | 254 | 1 |
| 2 | 192.168.0.2 | 3009 | 0% | 29% | 64 | 1 |
| 3 | 192.168.0.3 | 9967 | 0% | 17% | 254 | 1 |

started the openMosixanalyzer

When you run the command *openmosixview* do this as root (so you can change the different settings) and don't start the program in the background since it wants to use the console that the program started from for error messages etc.

When openMosixview has started you will get a window that shows the nodes on your cluster and their status.

The status information is shown in several columns, with the top row showing the entire clusters status, these columns are:

- Id**:** The node number with a background color that shows if the node is up (green) or down (red).
- Clusternodes**:** A button with the nodes IP-address on, if the button is pressed a window shows up. This new window can be used to set some of the nodes settings (the window that shows up if the button *all-nodes* are pushed sets these settings for all the nodes). In this window you can:
  - Turn auto migration on and off.
  - Tell the node if it should talk to other nodes.
  - Set the node so local processes should stay on it.
  - Tell the nodes guest processes to leave the node.
  - Start and stop openMosix.

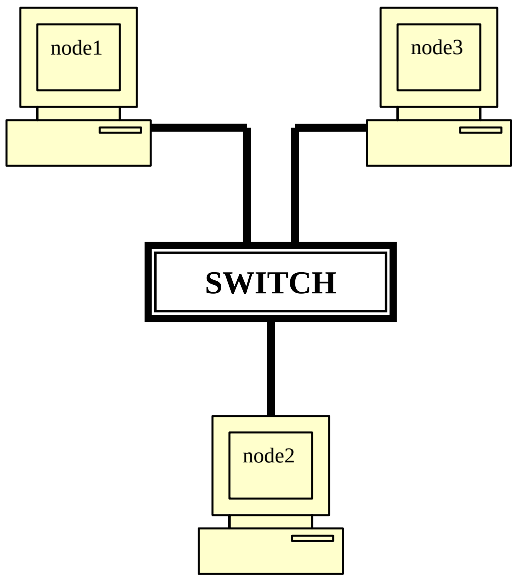
    - Open a console on the node.
    - Open the program openMosixprocs on the node.
    - Tell openMosixview on what machine the console and the openMosixprocs program should be displayed on (preferably the machine where you sit at the monitor and run X windows).

- Load-balancing efficiency**:** On the first row where the entire cluster information is shown displays how effective the load balancing is. On the rows that show information about specific nodes a slider exists that shows the current speed of the node and if the slider is moved this speed is changed. Next to the slider is the numerical value of the speed.
- Overall load**:** The load on the entire cluster and the different nodes.
- Overall used mem**:** The overall used memory on the entire cluster and the different nodes.
- All memory**:** The physical memory on the entire cluster and the specific nodes in megabytes.
- All CPU**:** The first row shows the number of CPU's on the entire cluster. The other rows show the number of CPU's on a specific node.

# Appendix C

## Configuration of DCEwulf

How the system was physically connected is presented in the picture below. Data about the nodes are listed in the table.



| Hostname | IP address on network | Hardware |
|---|---|---|
| node1 | 192.168.0.1 | Pentium 4 (HT Technology) 3.0 GHz with 256 MB RAM |
| node2 | 192.168.0.2 | AMD Athlon 1800+ with 128MB RAM |
| node3 | 192.168.0.3 | Pentium 4 (HT Technology) 3.0 GHz with 256 MB RAM |

The hardware for connecting network used was:

- Cisco Catalyst 1900 switch with 8-port configuration.
- 100-MBps full duplex LAN cable.

When the performance tests was made the auto-discovery daemon had been shut down and therefore the file /etc/mosix.map file was the one that was edited to get the cluster configured. This file was also the only file that was edited when openMosix was configured. Below it can be seen how that file looked like for the different cluster types that were used in the performance tests.

**Homogeneous cluster**

When doing the performance tests on a homogeneous cluster openMosix on node2 was shut down and the text in the file /etc/mosix.map on the two nodes (node1 and node3) was:

```
# MOSIX CONFIGURATION
# ===================
#
# Each line should contain 3 fields, mapping IP addresses to MOSIX
# node-numbers:
# 1) first MOSIX node-number in range.
# 2) IP address of the above node (or node-name from /etc/hosts).
# 3) number of nodes in this range.
#
# Example: 10 machines with IP 192.168.1.50 - 192.168.1.59
# 1 192.168.1.50 10
#
# MOSIX-# IP number-of-nodes
# ============================
1 192.168.0.1 1
2 192.168.0.3 1
```

**Heterogeneous cluster with 2 computers**

When doing the performance tests on a heterogeneous cluster openMosix on node1 was shut down and the text in the file /etc/mosix.map on the two nodes (node2 and node3) was:

```
# MOSIX CONFIGURATION
# ===================
#
# Each line should contain 3 fields, mapping IP addresses to MOSIX
# nodenumbers:
# 1) first MOSIX node-number in range.
# 2) IP address of the above node (or node-name from /etc/hosts).
# 3) number of nodes in this range.
#
# Example: 10 machines with IP 192.168.1.50 - 192.168.1.59
# 1 192.168.1.50 10
#
# MOSIX-# IP number-of-nodes
# ============================
1 192.168.0.2 2
```

**Heterogeneous cluster with 3 computers**

When doing the performance tests on a heterogeneous cluster the text in the file /etc/mosix.map on the three nodes (node1, node2 and node3) was:

```
# MOSIX CONFIGURATION
# ===================
#
# Each line should contain 3 fields, mapping IP addresses to MOSIX
# nodenumbers:
# 1) first MOSIX node-number in range.
# 2) IP address of the above node (or node-name from /etc/hosts).
# 3) number of nodes in this range.
#
# Example: 10 machines with IP 192.168.1.50 - 192.168.1.59
# 1 192.168.1.50 10
#
# MOSIX-# IP number-of-nodes
# ============================
1 192.168.0.1 3
```

# Appendix D

## Data from the performance tests

**No cluster**

In this test there was 1 simulation with 600 replications. These tests were made on the AMD Athlon 1800+ machine, node2, and the two Pentium 4 (HT Technology) 3.0 GHz machines, node1 and node3, without any migration between the machines. These data are to see how much the clustering of computers help when it comes to the performance. If the time of a test on a cluster is lower than the times that the job takes on one machine then the cluster isn't good to use.

| | On node1 | On node2 | On node3 |
|---|---|---|---|
| Test | Time | Time | Time |
| 1 | 469.99 | 1656.54 | 462.43 |
| 2 | 469.76 | 1654.53 | 462.47 |
| 3 | 469.25 | 1654.53 | 462.44 |
| 4 | 469.84 | 1655.16 | 462.42 |
| 5 | 469.29 | 1656.01 | 462.42 |
| 6 | 469.34 | 1656.12 | 462.43 |
| 7 | 469.7 | 1654.09 | 462.43 |
| 8 | 469.7 | 1653.18 | 462.42 |
| 9 | 469.29 | 1652.49 | 462.42 |
| 10 | 469.18 | 1654.37 | 462.42 |
| 11 | 469.47 | 1653.15 | 462.43 |
| 12 | 469.86 | 1652.56 | 462.42 |
| 13 | 469.69 | 1652.75 | 462.43 |
| 14 | 469.8 | 1654.02 | 462.43 |
| 15 | 469.15 | 1652.79 | 462.42 |
| 16 | 469.78 | 1652.42 | 462.43 |
| 17 | 469.7 | 1652.81 | 462.81 |
| 18 | 469.74 | 1654.31 | 462.41 |
| 19 | 469.61 | 1653.02 | 462.42 |
| 20 | 469.29 | 1656.91 | 462.43 |
| 21 | 469.38 | 1652.51 | 462.43 |
| 22 | 469.65 | 1652.81 | 462.43 |
| 23 | 469.32 | 1654.33 | 462.43 |
| 24 | 469.35 | 1652.41 | 462.44 |
| 25 | 469.67 | 1652.68 | 462.42 |
| 26 | 469.64 | 1652.35 | 462.44 |
| 27 | 469.67 | 1654.35 | 462.43 |
| 28 | 469.82 | 1652.44 | 462.43 |
| 29 | 469.33 | 1652.46 | 462.44 |
| 30 | 469.77 | 1654.06 | 462.43 |

## Homogeneous cluster

In this test there were 2 simulations with 300 replications per simulation. These tests were made on the two Pentium 4 3.0 GHz machines, node1 and node3.

| Test | OpenMosix Time |
|---|---|
| 1 | 236.83 |
| 2 | 236.64 |
| 3 | 237.03 |
| 4 | 237.51 |
| 5 | 238.73 |
| 6 | 237.35 |
| 7 | 237.03 |
| 8 | 237.2 |
| 9 | 236.79 |
| 10 | 237.69 |
| 11 | 237.99 |
| 12 | 237.08 |
| 13 | 236.95 |
| 14 | 237.05 |
| 15 | 236.99 |
| 16 | 237.12 |
| 17 | 237.08 |
| 18 | 237.24 |
| 19 | 237.07 |
| 20 | 237.22 |
| 21 | 238.8 |
| 22 | 237.06 |
| 23 | 237.08 |
| 24 | 237.09 |
| 25 | 237.13 |
| 26 | 236.74 |
| 27 | 237.79 |
| 28 | 237.36 |
| 29 | 238.21 |
| 30 | 237.12 |

## Heterogeneous cluster with 2 computers

In this test there were 2 simulations with 300 replications per simulation. These tests were made on a cluster consisting of the Pentium 4 3.0 GHz machine, node2, and the AMD Athlon 1800+ machine, node3.

|  | OpenMosix from node2 | OpenMosix from node3 |
|---|---|---|
| Test | Time | Time |
| 1 | 470.64 | 462.82 |
| 2 | 466.9 | 463.24 |
| 3 | 468.74 | 463.24 |
| 4 | 469.99 | 462.78 |
| 5 | 470.1 | 462.74 |
| 6 | 470.94 | 462.74 |
| 7 | 471.66 | 462.75 |
| 8 | 469.94 | 462.74 |
| 9 | 470.23 | 462.75 |
| 10 | 470.56 | 462.75 |
| 11 | 469.88 | 462.71 |
| 12 | 469.79 | 462.72 |
| 13 | 470.61 | 462.72 |
| 14 | 469.86 | 462.74 |
| 15 | 470.22 | 463.2 |
| 16 | 470.12 | 462.76 |
| 17 | 470.26 | 462.74 |
| 18 | 469.93 | 462.74 |
| 19 | 468.81 | 462.73 |
| 20 | 472.59 | 462.75 |
| 21 | 468.58 | 462.74 |
| 22 | 469.01 | 462.73 |
| 23 | 470.23 | 462.75 |
| 24 | 469.95 | 462.74 |
| 25 | 470.23 | 462.74 |
| 26 | 470.23 | 462.73 |
| 27 | 469.2 | 462.72 |
| 28 | 470.04 | 462.72 |
| 29 | 470.26 | 463.13 |
| 30 | 470.13 | 462.79 |

## Heterogeneous cluster with 3 computers

In this test there were 3 simulations with 200 replications per simulation. These tests were made on a cluster consisting of two Pentium 4 3.0 GHz machine, node1 and node3, and the AMD Athlon 1800+ machine, node2.

| | OpenMosix from node2 | OpenMosix from node1 |
|---|---|---|
| Test | Time | Time |
| 1 | 261.53 | 247.3 |
| 2 | 244.03 | 243.37 |
| 3 | 245.11 | 242.98 |
| 4 | 245.57 | 243.57 |
| 5 | 253.21 | 254.04 |
| 6 | 246.55 | 243.72 |
| 7 | 245.09 | 243.3 |
| 8 | 250.29 | 261.11 |
| 9 | 266.67 | 241.97 |
| 10 | 252.48 | 247.35 |
| 11 | 253.42 | 243.75 |
| 12 | 249.39 | 240.51 |
| 13 | 252.35 | 261.76 |
| 14 | 255.48 | 241.91 |
| 15 | 252.33 | 242.79 |
| 16 | 250.75 | 243.65 |
| 17 | 247.64 | 240.84 |
| 18 | 246.81 | 242.91 |
| 19 | 248.4 | 241.04 |
| 20 | 250.29 | 244.03 |
| 21 | 258.67 | 241.65 |
| 22 | 254.94 | 258.18 |
| 23 | 254.54 | 249.14 |
| 24 | 254.32 | 246.48 |
| 25 | 251.29 | 246.44 |
| 26 | 254.01 | 247.06 |
| 27 | 257.19 | 253.35 |
| 28 | 257.32 | 236.14 |
| 29 | 247.61 | 252.11 |
| 30 | 252.62 | 242.55 |

## References

[1] B. Knox, *openMosix, an Open Source Linux Cluster Project,* http://www.openmosix.org/
[2] A. Barak, *MOSIX*, http://www.mosix.org/
[3] *Licenses - Gnu Project - Free Software Foundation (FSF),* http://www.fsf.org/licenses/licenses.html#GPL
[4] *RedHat Linux*, http://www.redhat.com/
[5] *Debian GNU/Linux*, http://www.debian.org/
[6] H-G. Hegering, S. Abeck, B. Neumair, *Intergrated Management of Networked Systems: Concepts, Architectures, and Their Operational Application,* 1999, ISBN 1-55860-571-1
[7] G. Pfister. *In Search of Clusters*. Prentice Hall PTR, NJ, 2nd Edition, NJ, 1998.
[8] K. Hwang and Z. Xu. *Scalable Parallel Computing: Technology, Architecture, Programming.* WCB/McGraw-Hill, NY, 1998.
[9] M.A. Baker, G.C. Fox, and H.W. Yau. *Review of Cluster Management Software*. NHSE Review, May 1996. http://www.nhse.org/NHSEreview/CMS/
[10] *The Beowulf Project.* http://www.beowulf.org
[11] *QUT Gardens Project*. http://www._t.qut.edu.au/CompSci/PLAS/
[12] *MPI Forum*. http://www.mpi-forum.org/docs/docs.html
[13] *The PVM project*. http://www.epm.ornl.gov/pvm/