Gollach — Configuration of a Cluster Based Linux Virtual Server

by

Tinashe Gwena

A thesis submitted to the Department of Electrical Engineering,
University of Cape Town, in fulfilment of the requirements
for the degree of

Master of Applied Science

at the

UNIVERSITY OF CAPE TOWN

February 14, 2003

Declaration

I declare that this thesis is my own, unaided work. It is being submitted for the degree of Master of Applied Science in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author

Signed by candidate

Department of Electrical Engineering
Cape Town, February 14, 2003

Abstract

This thesis describes the Gollach cluster. The Gollach is an eight machine computing cluster that is aimed at being a general purpose computing resource for research purposes. This includes image processing and simulations. The main quest in this project is to create a cluster server that gives increased computational power and a unified system image (at several levels) without requiring the users to learn specialised tricks. At the same time the cluster must not be tasking to administer.

The making of this cluster aims to give a single image on several levels of the computer, which are: processes, I/O (Input/Output) and storage. To achieve this the cluster computer is built together on a unified backbone that ensures sharing on the levels mentioned above. It makes use of diskless booting, NFS-root (Network File System), MOSIX and LVS (Linux Virtual Server) to achieve this. MOSIX migrates processes and balances load. LVS distributes users on the cluster server. Diskless booting and NFS-root file system ensure that all the machines in the cluster share a single disk image. Mass storage has been aggregated with the help of NBD (Network Block Device) and software RAID (Redundant Array of Independent Disks).

The Gollach has been built and it provides a general computing resource for multiple users. Administration turns out to be at almost the same level as a single Linux server due to the use of ClusterNFS. ClusterNFS is an NFS server which provides name translation. Therefore machine specific files can be kept unique on the file server. User distribution using LVS has worked as expected, users' requests are always attached to the machine with the least connections. The mass storage device has been tested to work with RAID-0. The cluster has been made to run conventional programs as a conventional server, here it works to expectation with no visible difference to a single machine server. Parallelism with the help of MOSIX has been observed and non-I/O bound processes have experienced a speed-up due to the process migration. Disk I/O bound processes suffer though, from the speed descaling found with multiple simultaneous reads/writes.

Acknowledgements

I would like to thank the supervisor, Professor Michael Inggs, for guiding the progress of this project. Over this project's time period he was helpful with useful advice and criticism.

I would also like to thank Leon Alexander and Dr. Richard Lord for giving aid when it was required at different parts of the project. I am also grateful for their running through most of the documentation and providing educated opinions of the work done.

I am very grateful to my family and friends who would throw in the extra encouragement and provide the courage to push on. Every ounce of it was most necessary.

Lastly I would like to thank the members of the RRSG (Radar Remote Sensing Group) in the Department of Electrical Engineering at the University of Cape Town for providing a second home in which I could do my research.

Contents

List of Figures

List of Tables

List of Acronyms

3DES	Triple DES See DES
ACL	Access Control Lists A list of data that specifies which permissions, each user or group has to a specific system object, such as a directory or file.
AES	Advanced Encryption Standard
AFS	Andrew File System
ARP	Address Resolution Protocol a TCP/IP protocol used to convert an IP address into a physical address (called a DLC address), such as an Ethernet address.
BIOS	Basic Input Output System Basic Input Output System, which loads and executes operation functionality usually stored on the computer's hard disk (in ROM mode); also may be accessible from compact disk or floppy disk at install time. See ROM
BIT	Binary Digit
BSD	Berkeley Software Distribution The name of distributions of source code from the University of California, Berkeley, which were originally extensions to AT&T's Research UNIX operating system.
CAST	Carlisle Adams/Stafford Tavares
CAST-128	CAST 128 bit Encryption See CAST
CM5	Connection Machine 5 A massively parallel processor made by Thinking Machines Inc. The number of processing nodes can range from 32 up to 16,384. Each processing node consists of a SPARC processor operating at 32 MHz or 40 MHz. See MPP
CODA	Constant Data Availability A fully featured distributed file system developed at Carnegie Mellon.
COTS	Commercial Off The Shelf
COW	Cluster of Workstations

CPU	Central Processing Unit	The element of all computation in a computer. Usually this is implemented as a single integrated circuit.
DES	Digital Encryption Standard	
DFS	Distributed File System	A file-system that resides on one or more computers and is accessible remotely on other computer computers.
DFSA	Direct File System Access	The system of moving processes to data on MOSIX.
DHCP	Dynamic Host Configuration Protocol	A protocol for assigning dynamic IP addresses to devices on a network. See IP
DHCPD	DHCP daemon	The DHCP server daemon. See DHCP
DOS	Disk Operating System	An operating system developed by Microsoft Corporation that was popular in the 1980's and early 1990's. It was the predecessor of the now almost ubiquitous Windows Operating System.
DR	Direct Routing	
DRBD	Distributed Replicated Block Device	A means of mirroring block devices over networks
EPROM	Erasable Programmable ROM	A type of ROM which allows reprogramming after erasure by exposure to ultra violet light through a glass window.
EXT2	Second Extended File System	A file system used mainly with the Linux Operating System.
EXT3	Third Extended File-system	A journalling extension to EXT2. See also EXT2
FAT16	16 bit FAT	The original Microsoft DOS FAT file system. See DOS
FAT32	32 bit FAT	The file system that was used in Microsoft's Windows 95 operating system. It was based on the DOS FAT file system. See DOS
FFS	Fast Filing System	The file system used in BSD Unix. See BSD
FS	file system	
GB	Gigabyte	1,024 Megabytes. Sometimes referred to as 1,000 Megabytes. See MB
Gb	Gigabit	1,024 megabits. Sometimes referred to as 1,000 megabits. See Mb, BIT

GB/S	Gigabit per second See Gb
GRUB	Grand Unified Boot-loader An fully featured operating system loader. It includes provision for network booting and has the ability to read several file-systems in order to locate operating software.
HA	High Availability A server made specially for little or no down-time by employing redundancy.
HPC	High Performance Computing A field of highly demanding computing applications that require powerful computers. Examples are weather simulations.
HTTP	Hyper-Text Transfer Protocol The protocol most often used to transfer information from Web servers to browsers.
I/O	Input/Output
IBM	International Business Machines A large computer corporation well known for creating the first desktop personal computer (PC).
IDE	Integrated Drive Electronics A type of hard disk drive implementation
IDL	Interactive Data Language A programming language designed by Research Systems Inc. It is mainly suited for image data processing.
IP	Internet Protocol The protocol used to route a data packet from its source to its destination via the Internet.
IPv4	IP version 4 See IP
JFS	Journalling File System A file system that records its transactions in a journal. Therefore on unexpected shutdowns, e.g. system crash. Recovery time during file system check is minimised.
KB	Kilobyte 1024 bytes.
KB	kilobit 1,024 bits. Sometimes referred to as 1,000 bits. See BIT
KB/S	Kilobyte per second See KB
LAN	Local Area Network A computer network that covers a relatively small area. Most LANs cover a single building or group of buildings. A system of LANs can be connected over any distance through telephone lines and radio waves, creating a wide-area network. See WAN
LVM	Logical Volume Manager A software means of combining storage devices into logical volumes as opposed to physical volumes.

LVS	Linux Virtual Server	A virtual server based on the Linux Operating System.
MAC	Media Access Control	A hardware address that uniquely identifies each node of a network. I
MB	Megabyte	1,024 kilobytes. Sometimes referred to as 1,000 kilobytes. See KB
Mb	Megabit	1,024 kilobits. Sometimes referred to as 1,000 kilobits. See Kb
MB/s	Megabyte per second	See MB
Mb/s	Megabit per second	See Mb
MBR	Master Boot Record	A section of a disk containing information necessary for boot-up.
MCOTS	Modified COTS	See COTS
MFS	MOSIX File System	A file system specially made to take advantage of DFSA on MOSIX. See MOSIX, DFSA
MHz	MegaHertz	A measure of frequency equalling 10^6 cycles per second. It is also used as a measure of base clock speed in microprocessors.
MOPI	MOSIX massive Parallel I/O	A set of libraries that allow the programming of parallel I/O to be used in MOSIX clusters. See I/O
MOSIX		A software package that takes advantage of the power of a cluster by migrating processes and thus balancing load. In operation it turns a cluster to something like an SMP. See SMP
MPI	Message Passing Interface	Software for parallel processing that adds message passing semantics to either C/C++ or FORTRAN.
MPP	Massively Parallel Processor	A computer system which contains a large number of computing elements. An example is the Connection Machine with 65536 processors.
NAT	Network Address Translation	A system of changing the IP address of a machine on a packet in transit so as to create a virtual server or to allow masquerading. See IP
NBD	Network Block Device	A means of networking storage devices.
NFS	Network File System	A protocol used to access files over a network regardless of machine, operating system, or architecture.

NIC	Network Interface Card
NOW	Network Of Workstations A group of computers on a network that are made to perform parallel computations.
OS	Operating System The most important program that runs on a computer. Every general-purpose computer must have an operating system to run other programs. Operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers.
PC	Personal Computer
PII	Pentium II Part of the Intel Pentium series of microprocessors.
PVM	Parallel Virtual Machine A parallel computing paradigm that treats a cluster of computers as a single virtual computer.
PXE	Pre-Boot Execution Environment
RAID	Redundant Array of Independent Disks A means of combining two or more disk drives into an array for the sake of performance and/or fault tolerance.
RAM	Random Access Memory The working memory of the computer. RAM is the memory used for storing data temporarily while working on it, running application programs, etc.
RJ-45	Registered Jack 45 An 8 line physical connector used in networking.
RMT	Remote MagTape
ROM	Read Only Memory Read-Only Memory, non-volatile memory that can be read, but not written, by the microprocessor, pre-recorded, often holding critical programs, such as boot data.
RPM	Redhat Package Manager A means of distributing files for the RedHat Operating System.
RPM	Revolutions Per Minute
RRSG	Radar Remote Sensing Group
RVM	Recoverable Virtual Memory
SCSI	Small Computer Systems Interface An interface mainly used for storage devices in computers. It is especially popular with servers since it allows for daisy-chaining multiple devices on a single cable.

SMP	Symmetric Multi-Processor A computer system which has two or more processors connected in the same cabinet, managed by one operating system, sharing the same memory, and having equal access to input/output devices. Application programs may run on any or all processors in the system.
sRAID	software RAID An implementation of RAID using software. See RAID
SSH	Secure Shell Remote terminal access software which includes encryption for security.
SSI	Single System Image
TCP	Transmission Control Protocol The most common Internet transport layer protocol.
TCP/IP	TCP over IP See TCP, IP
TFTP	Trivial File Transfer Protocol A simple form of the File Transfer Protocol (FTP). TFTP uses the User Datagram Protocol (UDP) and it provides no security features. It is often used by servers to boot diskless workstations, X-terminals, and routers.
TUN	IP Tunelling A means of encapsulating an IP datagram in another IP datagram to allow a machine behind a fire-wall to access an outside network. See IP
UNFSD	Universal NFS Daemon An implementation of NFS. See NFS
VFS	Virtual File System An abstract file system layer implemented in Linux to simplify file system programming.
VI	Visual editor A popular text editor for Unix.
VIP	Virtual IP The IP number used to access a virtual server.
VSG	Volume Storage Group
VSTA	Valencia Simple Tasker An experimental operating system that makes use of a microkernel and attempts to be POSIX compliant.
WAN	Wide Area Network A network of computers connected to each other over a long distance, via telephone lines and satellites.
X	X-window A graphical windowing system for Unix and Unix-like Operating Systems.

x86	Intel 80x86	A popular range of processor that are made by Intel Corporation. The x stands for a number from 1 to 4 depending on when it was made. The 8086 and Pentium range are also members of the range. They are primarily used in most desktop personal computers.
XFS	Extensible File System	A file system for Silicon Graphics Irix OS. See OS

Glossary

back-plane bandwidth The internal bandwidth capability of a network switch. This figure determines the maximum number of network devices that can communicate simultaneously without any losses.

Beowulf A means of interconnecting conventional workstations in order to achieve performance matching that of a supercomputer.

bit *Binary Digit* A computational quantity that can take on one of two values, such as true and false or 0 and 1. A bit is said to be set if its value is true or 1, and reset or clear if its value is false or 0. One speaks of setting and clearing bits. To toggle or invert a bit is to change it, either from 0 to 1 or from 1 to 0.

boot The process of loading a computer's operating system. If a system is working properly, the operating system boots when the computer is turned on.

Bootp BOOTstrap Protocol. A TCP/IP protocol allowing a diskless workstation to find its own IP address at start-up. See TCP, IP

boot-ROM A ROM device that is installed on an ethernet card to enable the machine to boot remotely. See ROM, ethernet

BSD *Berkeley Software Distribution* The name of distributions of source code from the University of California, Berkeley, which were originally extensions to AT&T's Research UNIX operating system.

byte A sequence of eight bits. See bit.

Cisco A networking products manufacturing company.

Cluster Refers to a group of computing components interconnected over a network that collaborate to give a single system image for some service.

ClusterNFS A version of NFS that allows for name translation. See NFS

coaxial cable A communication cable which has a core surrounded by a grounded shield in order to attenuate noise and improve transmission efficiency.

cp A Unix command meaning copy. Used for copying files and directories on a local file system.

C++ An object oriented language based on the popular C compiled programming language. C is central to most source code distributed.

Cray A manufacturer of supercomputers that are mostly used in weather forecasting and bio-medical research.

Cray T3D A super computer made by Cray Research. See Cray

Cray T3E A super computer made by Cray Research. See Cray

Convex SPP A scalable parallel processor that had the ability to share memory directly to processors and was scalable up to 128 processors.

daemon A background process that provides a system service.

Deep Blue A computer system specially made to play chess at grand-master level.

distributed computing Computing services supplied by two or more computers connected over a network.

DOS *Disk Operating System* An operating system developed by Microsoft Corporation that was popular in the 1980's and early 1990's. It was the predecessor of the now almost ubiquitous Windows Operating System.

Emacs A highly extensible text editor from GNU. Its central core is a Lisp interpreter which then carries macros defining all the rest of he features.

ethernet A local area network (LAN), that sends communications through radio frequency signals carried by a coaxial cable. Each computer on the network checks to see if another computer is transmitting and "gets in line." If two computers transmit at the same time and their messages collide, they wait and send again in turn. See LAN

fork-and-forget A means of invoking parallelism in a MOSIX cluster from merely forking processes and waiting for the MOSIX system to do the rest. See MOSIX

Fast Ethernet Ethernet that gives speeds of up to 100 Mb/s.

optical fibre A communication cable that transmits data using light.

filter To dispose of unwanted packets

firewall A network security gateway that filters out undesirable packets that are destined for a client.

flash A type of electrically erasable non-volatile storage. See EPROM

FreeBSD A free version of BSD Unix. See BSD, Unix

FTP *File Transfer Protocol* A protocol for transferring files from one computer to another over a network.

`ftp` A Unix command that invokes an FTP client. See FTP

Galeon An Internet web browser.

Gigabit-Ethernet Ethernet that gives speeds of up to 1000 Mb/s or 1 Gb/s

GNU *GNU is Not Unix* An organisation that promotes the free distribution of software and thus encouraging software development.

Gollach A cluster server used by the Radar Remote Sensing Group. See RRSG

HA *High Availability* A server made specially for little or no down-time by employing redundancy.

Hurd The GNU Hurd is the GNU project's replacement for the Unix kernel. It is a similar concept to the Linux kernel. See Linux

IBM *International Business Machines* A large computer corporation well known for creating the first desktop personal computer (PC).

`inittab` A file containing initialisation information in a Unix type OS. See Unix, OS

Intel An intergrated circuits (IC) manufacturing company. It is well known for making the first IC in the world. It makes the x86 line of processors. See x86

Intermezzo A distributed file system that is especially made to address device replication.

Internet A world wide network of computers that links computers sharing the TCP/IP protocol. See TCP/IP

IP *Internet Protocol* The protocol used to route a data packet from its source to its destination via the Internet.

IPChains A system for creating fire-walls on the Linux platform.

IP Masquerading A means of providing Internet access to computers behind a gateway by having the gateway machine masquerade as the client.

IPTables A system for creating fire-walls on the Linux platform.

Kerberos Kerberos is a network authentication protocol. It is designed to provide strong authentication for client/server applications by using secret-key cryptography

Kernel The core part of a program or operating system; performs the basic functions that more advanced functions depend on.

LAN *Local Area Network* A computer network that covers a relatively small area. Most LANs cover a single building or group of buildings. A system of LANs can be connected over any distance through telephone lines and radio waves, creating a wide-area network. See WAN

Linux Linux is a free Unix-type operating system originally created by Linus Torvalds with the assistance of developers around the world.

Lyx A front end graphical user interface for the LaTeXdocument preparation system. It gives a WYSIWYG (what you see is what you get) mode of operation during the creation of documents.

Mach GNU Mach is the microkernel of the GNU system. A microkernel provides only a limited functionality, just enough abstraction on top of the hardware to run the rest of the operating system in user space.

Matlab A mathematical application based on a language specially built for matrix manipulation. The name means Matrix Laboratory.

migrate To move to another computer via a network.

Minix A free Unix clone intended for educational purposes that was a predecessor to Linux. See Linux, Unix

mirror A means of creating a highly resilient data array by a means that constantly duplicates contents and transactions between two devices.

`mon` The MOSIX monitoring program. See MOSIX

MOSIX A software package that takes advantage of the power of a cluster by migrating processes and thus balancing load. In operation it turns a cluster to something like an SMP. See SMP

`mospe` A file in the MOSIX system that determines the node identity of the current computer. See MOSIX

`mosrun` A command available in MOSIX to allow the changing of running conditions of another file. See MOSIX

motherboard A central circuit-board in a computer which carries the most necessary basic hardware.

mount A process of making a device a part of the file system true so that its contents are accessible

message passing A parallel processing model that sends information between processors as messages.

MPP *Massively Parallel Processor* A computer system which contains a large number of computing elements. An example is the Connection Machine with 65536 processors.

`mtab` A file in the MOSIX system that determines which machines are part of the cluster. See MOSIX

Multiboot A specification of the interface between a boot loader and a operating system, such that any complying boot loader should be able to load any complying operating system.

nCube A provider of scalable streaming media infrastructure to broad-band, Internet and telecommunications carriers worldwide. It historically used to produce MPPs. See MPP

NetBSD NetBSD is a free, highly portable UNIX-like operating system. It is based on BSD Unix. See BSD

NetDispatcher A software solution to balance request loads to TCP/IP based servers. See TCP/IP

Netscape An Internet browser.

network A group of interconnected computers, and their connecting cables and hardware.

NFS *Network File System* A protocol used to access files over a network regardless of machine, operating system, or architecture.

OpenBSD OpenBSD is a freely available, multi-platform BSD-based UNIX-like operating system. See BSD

OS *Operating System* The most important program that runs on a computer. Every general-purpose computer must have an operating system to run other programs. Operating systems perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices such as disk drives and printers.

OS/2 An operating system made by IBM Corporation with the intent of replacing DOS. See IBM, DOS

packet A packet is the unit of data that is routed between an origin and a destination on the Internet or any other packet-switched network

parallel programming A programming style that takes advantage of multiple processors by doing several simultaneous computations. This leads to a speed-up of the overall calculation.

parity Extra information that is stored about a small piece of data on wether the sum of the bits in the collection produces an even or an odd number. Thus defining either even or odd parity. It is a popular scheme in error detection and correction. See RAID

personalities The different types of RAID configuration that are available in a typical RAID system.

process A Unix process is one execution of a program or command.

proxy cache server A proxy server is a server that acts as an intermediary between a workstation user and the Internet so that the enterprise can ensure security, administrative control, and caching service.

RAID *Redundant Array of Independent Disks* A means of combining two or more disk drives into an array for the sake of performance and/or fault tolerance.

`raidtab` A configuration file in Linux software RAID that determines the layout of RAID devices on that computer. See Linux, sRAID

RAM *Random Access Memory* The working memory of the computer. RAM is the memory used for storing data temporarily while working on it, running application programs, etc.

`rcp` A Unix command meaning remote copy that allows files to be transferred files to and from another system over the network.

RAM-disk A part of the RAM that is made to behave as if it were a disk storage system. See RAM

RedHat A company that distributes a version of the Linux OS known as RedHat Linux. See Linux

ReiserFS A journalled file system that is based on balanced trees. It is implemented on Linux to solve the problem of having many small files on a large hard drive. See Linux

ROM *Read Only Memory* Read-Only Memory, non-volatile memory that can be read, but not written, by the microprocessor, pre-recorded, often holding critical programs, such as boot data.

router A device which routes packets from one network to another.

RRSG *Radar Remote Sensing Group*

shared memory A means of interconnected several processors to use common RAM in a parallel processor. See RAM

SMP *Symmetric Multi-Processor* A computer system which has two or more processors connected in the same cabinet, managed by one operating system, sharing the same memory, and having equal access to input/output devices. Application programs may run on any or all processors in the system.

sRAID *software RAID* An implementation of RAID using software. See RAID

stripe A means of combining data storage by inter-leaving storage devices.

Sun Microsystems A manufacturer of workstation servers that are mainly intended for networking applications.

share/unique A problem of deciding wether a file should be shared or should be unique to a particular machine in a cluster. This is a problem where the root device is shared by several computers.

TCP *Transmission Control Protocol* The most common Internet transport layer protocol.

TCP/IP *TCP over IP* See TCP, IP

Twofish Twofish is a block cipher by Counterpane Labs. It was one of the five Advanced Encryption Standard (AES) finalists.

Unix A popular server operating system made by AT&T.

voltmeter A device for measuring the potential difference between two points in an electrical circuit.

Virtual Server A group of networked computers made to respond to the same IP address and provide the same services such that they appear to be single computer. They are mainly intended to address high availability issues. See IP, HA

WAN *Wide Area Network* A network of computers connected to each other over a long distance, via telephone lines and satellites.

Windows An operating system built by Microsoft Corporation. It is the most widely used OS on x86 based computers. See OS, x86

workstation A machine with the computing power of a minicomputer or a mainframe but in a personal computer type package.

X *X-window* A graphical windowing system for Unix and Unix-like Operating Systems.

x86 *Intel 80x86* A popular range of processor that are made by Intel Corporation. The x stands for a number from 1 to 4 depending on when it was made. The 8086 and Pentium range are also members of the range. They are primarily used in most desktop personal computers.

Xfig A graphics program for the X platform that is mainly used in the creation of illustrations. See X

X-terminal A window for typing system commands in X. See X

Chapter 1

Introduction

Clustering in computing terms is the practice of interconnecting computers with the help of suitable hardware, software and networking technology. The aim of such a combination is to bring together the resources that are provided by the individual machines and use them collectively. Resources in this case means:

- processing power from the CPU (Central Processing Unit),
- volatile, non-volatile storage and
- other computer related services like web servers, cache servers, data-basing etc.

These resources are provided by the individual machines that make up the cluster.

1.1 Background on Clustering

1.1.1 Clustering History

Historically clustering technology made big news with the advent of Beowulf Clusters [3]. This type of cluster was mainly fuelled by research institutions as a means of solving large real world problems using COTS (Commercial Off The Shelf) components. These were mainly scientific applications like simulations related to physics, biology, chemistry and other demanding situations where supercomputers are usually used.

The main requirement for these scientific applications was parallel programming and distributed computing. Parallel computing is the use of several computing elements simultaneously on one computation. These computing elements may be CPUs or whole workstations interconnected by some network.

With the use of appropriate middle-ware i.e. PVM (Parallel Virtual Machine), MPI (Message Passing Interface) or other Beowulf packages, data is split according to the type of problem at hand and shipped out across the cluster. This data can then be serviced by all the machines in the cluster in a piece-wise fashion. After every node has completed its part of the computation, the results are recombined for analysis or for further computation. Therefore a computational speed-up is realised allowing the processing of very large data sets.

1.1.2 Commercial Cluster Application

The above application of clustering technology proved to be quite a success with some notable clusters being listed among the worlds top supercomputers for their capabilities. On the other hand, clustering technology is also finding purpose within commercial and industrial applications. Multiple computing nodes can provide advantages and uses which substitute large expensive custom made servers. Some of these advantages are server storage redundancy, high availability and scalable servers.

High Availability

HA (High Availability) has been a recent industrial attraction for cluster computers [46]. This is a more commercial application of clusters where server applications are put on a collection of machines in order to increase throughput, resilience and reliability.

HA clusters are usually intended for use in situations where large numbers of users require service simultaneously. At the same time the server cannot afford any down-time. This is especially true for Internet, email, data-base servers and other places where a small service is in huge demand and thus becomes a large problem.

The use of multiple computers increases flexibility in terms of scalability and cost. When more power is required from the server, the means of solving the problem of increased demand is not to purchase a bigger, more expensive specialised server. Instead, the means of upgrading is to add low cost nodes to the cluster and these extra machines will take up the extra slack.

Distributed File Systems

A useful component in HA clusters is a DFS (Distributed File System) [50]. A distributed file system allows the cluster to have a combined storage solution. This is a major requirement in maintaining data consistency, reliability and in some cases in improving data I/O (Input/Output) rates. To increase reliability would probably imply some kind of data redundancy on the server.

Apart from data redundancy, processor redundancy is also usually employed to prevent down-time caused by processor failure.

1.2 The Radar Remote Sensing Group

The RRSG (Radar Remote Sensing Group) is a research group comprised of researchers that possess large computational requirements due to the nature of their work. These people all require computing resources from some powerful computing system. This is required for the many types of applications which are run by the members. The main applications in use are Matlab, IDL (Interactive Data Language), C++, PVM and MPI based applications. All of these applications are mainly intended for radar image processing and radar related simulations.

Apart from the somewhat more specialised applications, there is also a whole host of conventional Unix applications which include Xfig , Lyx, Netscape , Emacs and VI (Visual editor). All of these have their varying uses for the users.

The nature of image processing is, that it is a highly demanding field in terms of computing power. The people who need to run these applications have relatively slow computers on their desks. Thus the requirement is to concentrate energy on a small collection of more powerful machines which users can access as a cluster server.

The research group would thus benefit from a Cluster based Virtual Server. Such a server will be composed of several individual computers, but on the outside will appear as one computer. Thus it should give increased computational power without advertising itself as a cluster. Further to this, the cluster would need a scalable architecture and a simplified administration viewpoint. This is a necessity for maintenance and upgrade of the system.

1.3 The Gollach Cluster

The Gollach is an 8 machine cluster. It consist of 7 PII (Pentium II) computers running at 350 MHz (MegaHertz) plus 1 dual PII 350 MHz. These computers have on-board 100 Mb/s (Megabit per second) ethernet together with SCSI (Small Computer Systems Interface) controllers that have a maximum transfer rate of 40 MB/s (Megabyte per second) (See Appendix D).

The naming convention for the machines in the Gollach cluster is **Gollach-1** to **Gollach-8** in later sections of this document.

For the communication within the cluster there is a Cisco 2912 XL ethernet switch. This is a 100 Mb/s switch and the manufacturer claims that it has a back-plane bandwidth of 3.2 Gb/s (Gigabit per second). Thus we can expect to have 12 machines talking to each other without any losses in performance. This is assuming interconnections where each machine is connected to one other machine in the network and not a contending situation with several machines trying to talk to a certain one machine.

Apart from the on-board ethernet, the machine **Gollach-1** is also fitted with an extra ethernet interface for outside communication. Thus our cluster is accessible and can access the outside network.

1.4 Purpose of Cluster

The aim of this cluster is to use the combined storage and computational power of the nodes included in order to give a noticeable increase in computing speed and a better storage facility.

The main purpose of this cluster is that a variety of people use it at the same time. Due to the many available node machines, we would like to get a speed advantage with the combined processing power.

The cluster will be a general computational resource for conventional Unix applications. At the same time it should be possible to use it as a parallel computational resource for the benefit of users who have large computational problems. The cluster will also give an opportunity to provide large storage facilities to the users who may require it.

Therefore the main purpose of the cluster is to be able to provide a much bigger computer for general use.

1.5 Aim of the Project

The aim of this project is to present the concept of using a cluster of computers to be used and viewed as one computer in all its aspects. This includes access, processor and storage. Added to this the project aims to show that this can be done in a way which actually simplifies the setup rather than complicate it.

This project also aims to investigate the various technologies surrounding clustering hardware and software which may be of use in varied circumstances to those in this project.

1.6 Project Scope

This project looks at the hardware, software and issues surrounding the creation of a general purpose *Cluster Based Virtual Server.* The main purpose of this cluster is to enable the running of everyday Unix type applications that are used by a research group.

This will look into issues surrounding the creation and administration of cluster computer servers. We will try and find out about other implementations and research on other technologies that are available. This project will show how one such a system is put together and find out the problems which are faced in this field of computation. It will discuss other methods of dealing with these issues and also touch upon some future possibilities for one such a system.

The project will not delve into how a powerful commercial server is built out of specialised components. Instead, this project will concentrate on off the shelf hardware that is not startling in terms of performance and will show that in a cluster combination, a powerful system can be built.

1.7 Organisation of this Document

The organisation of the rest of this document is outlined as the following :

- The next chapter, i.e. Chapter 2 is the *Problem Statement.* This outlines the problem that we are trying to solve with this project.
- Chapter 3 is the *Literature Review and Theory.* In this chapter there is the background on related systems and discussion on the insides of the components used in this project.

- Chapter 4 with the title *Implementation* describes the building of the project in detail
- Chapter 5 is the *Tests and Results* chapter. It discusses the types of tests conducted after building up the project.
- Finally there is *Conclusions* as Chapter 6.

1.8 Text Conventions

In this document, file names, command entries and file listings are written in typewriter mono-space style. For example :

```
an example of typewriter monospace style
```

Console commands are written with a prompt. The prompt in this case is a hash (#) symbol. The hash is not to be confused with a comment line in script file listings. An example is :

```
#  cp file1 file2
```

File listings generally carry an incomplete frame to mimic a screen. In the case of a listing starting from the beginning of the file, then the frame is incomplete on the bottom only. This is shown below :

```
default  = 0
timeout  = 10

#image   0
title    Mosix 1.6.0 ( 2.4.18 )
         root (sd0,0)
         kernel /vmlinuz-2.4.18 root=/dev/md0 \
         md=0,/dev/sda3,/dev/sdb3 ro
```

In the case that the listing begins in the middle of the file, then the frame is open on the top and on the bottom. The following is an example :

```
.
.
host g1 {
hardware ethernet 00:a0:c9:ec:04:90;
fixed-address 192.168.0.1;
option root-path "/";
}
.
```

Chapter 2

Problem statement

2.1 Administrative Problem

Cluster computers are composed of several machines making up the nodes. These nodes can become quite a task to manage, especially if they are composed of large numbers of machines. This would especially be true for clusters with 25 or more nodes [44]. With such large numbers of machines, an administrator would find that they have to continuously run around the machines ironing out problems that constantly sprout unexpectedly.

2.2 User Related Issues

Users on such a cluster would require a means of accessing the power of the cluster. This means that the users have to have a means of using all the machines in the cluster. User access in this case refers to either:

- a per-user access to the entire cluster's power or,
- a group-wise access to the cluster's services.

Either way, there has to be a simple user view of the cluster. A poor means of providing such user access would be to give each user a list of IP numbers for all the nodes in the cluster. If a user feels dissatisfied with their service, they could try their luck on another server.

The above illustration is a poor means of providing clustering technology, especially with a large collection of machines. It is clear that in a server farm it could prove to be a greater inconvenience than it is worth.

2.3 Conceptual Issues

The Gollach is a collection of independent computers. These computers are linked by a communications network. Ideally many hands make light work thus there is more power available from more machines. A lot of computing problems exist where the job can be split. Also with Linux being multitasking, the kernel deals with many processes at "once". Thus there is opportunity for distribution on a process level. On the other hand we expect many users to use the cluster at once, thus we reach yet another level of distribution, the user level.

Our cluster has to be able to deal with these varying levels of distribution, so as to best reduce the load per node and increase processing power. On the other hand having the individual nodes in the cluster being independent entities presents problems with installation, administration and repair on the management level. It also has problems with usability on the user level.

2.4 Overall Problem

Thus the challenge is to try and get the best of both worlds. This means have the processing power of a cluster computer together with the administration of a single computer. At the same time requiring little or no special cluster oriented knowledge from the users who need to use the cluster.

Chapter 3

Literature Review and Theory

This chapter details the components that were useful in this project. It also outlines some of the alternatives technologies useful in building up a similar project.

3.1 Linux Operating System

The Linux operating system is central to our operation. Linux is a free software operating system that took the world by storm in the last decade [33][34]. Apart from the cost, Linux is fully featured in every way. Due to its Unix roots, it is very well suited to network server uses. Thanks to its open source nature, it allows for easy modification of system properties due to the availability of information and source code.

3.1.1 The Kernel

The most useful component of the Linux operating system is the Linux Kernel [29][30]. This is the heart of the entire system. It carries the whole core of operations, the drivers and the behaviour. The Kernel can be modified by a reconfiguration process. This will enable the use of other features available in the kernel, or disable unrequired services. The purpose of this can be to change the processor type, the ethernet cards, the file system, or virtually any

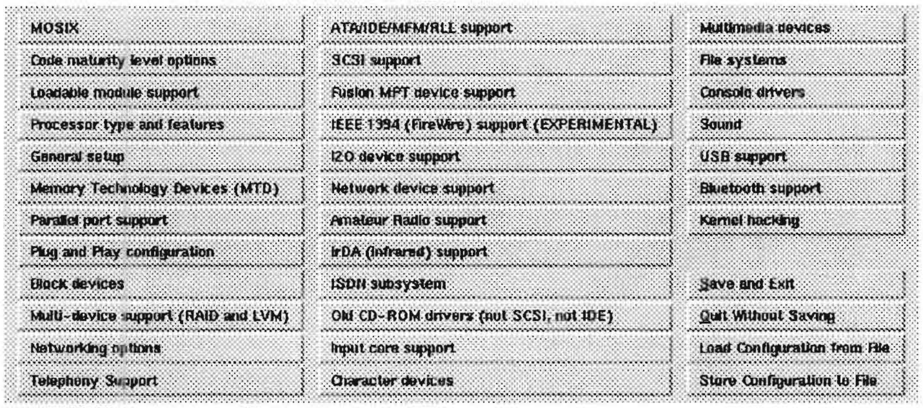


Figure 3.1: Typical Kernel Configuration Screen

other system feature. Apart from mere reconfiguration, the kernel can also be patched to add in non-standard features or to install bug fixes.

After patching and re-configuring, the kernel is recompiled. When this is done then the machine can be restarted with the new kernel. Once restarted then the newly compiled in features can then be used.

3.1.2 Kernel Drivers

Linux drivers are commonly used as part of the kernel. In order to do this they are either statically compiled in, otherwise they are dynamically loaded as modules.

Static Compilation

The kernel allows for static compilation of drivers. This is where the drivers are compiled into the kernel and thus will be initialised whenever the kernel begins to run. Even if they are not necessary, they will occupy system memory anyway. If several drivers are compiled in, (e.g. several different ethernet card drivers), but only one is used, then there will be a lot of unnecessary drivers loaded. Thus for a thinner kernel requirements, this is not always ideal.

Dynamic Loading

The other method for loading in drivers is dynamic compilation. This is where the drivers are compiled as kernel modules. Thus allowing their insertion and removal at will. All that is compiled-in is a group of kernel symbols that will attach to the kernel module during dynamic operation. This method allows for a very simple kernel and minimises the re-compilations that are necessary if system requirements change often.

3.1.3 Kernel Boot-up Parameters

The Linux kernel also allows for command line parameters during boot-up. These parameters allow for options to some of the drivers which cannot determine certain information during machine start-up. This can be information like IP (Internet Protocol) numbers of the machine, root directory location and whether the file system is read/write enabled etc.

3.1.4 Linux File System Structure

The Linux file system structure is defined in layers of abstraction. Instead of having the file system operations talking directly to the storage devices, it is instead separated by a software layer known as the VFS layer[32][30]. This allows the definition of a uniform file system structure for use by the kernel. File systems like EXT2 (Second Extended File System) connect through this abstract layer. Since it has a preset design, then it makes file system definition easier and only requires one driver to implement a file system type. The VFS system is then connected to the actual drivers for the low level storage systems. Thus specific work for a certain device only needs to be done on the specific driver since the VFS system has a uniform interface for access to the kernel.

The Figure 3.2[32] illustrates how the EXT2 file system is laid out.

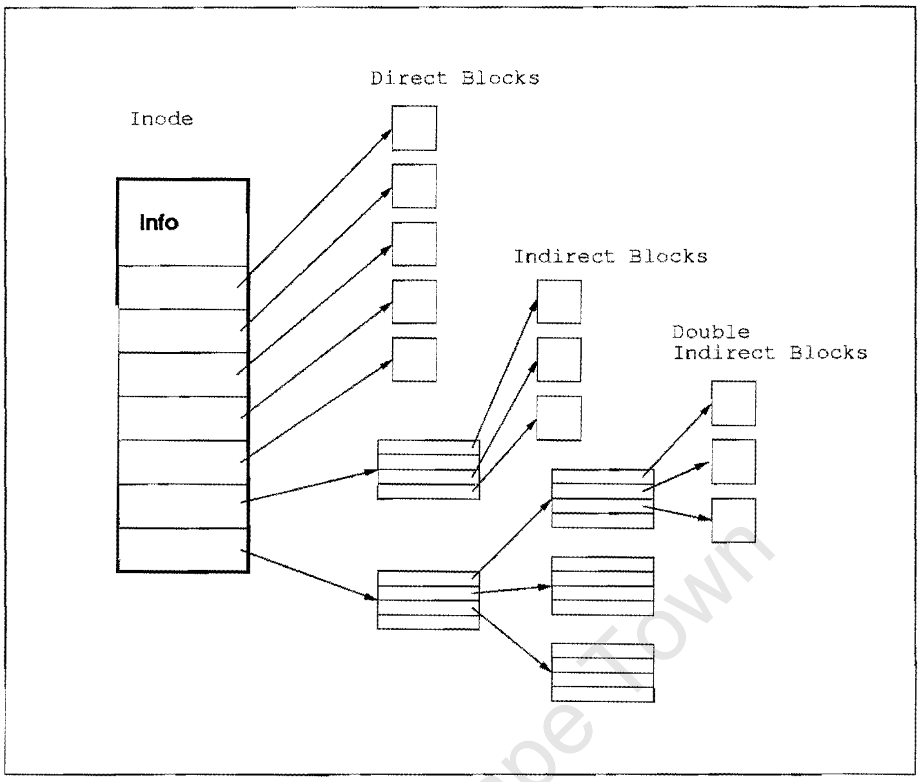


Figure 3.2: EXT2 File-system Structure

The Virtual File System Principle

According to Remy Card et. al [32], when a process issues a file oriented system call, the kernel calls a function contained in the VFS. This function handles the structure independent manipulations, then redirects the call to a function contained in the physical file-system code, which is responsible for handling the structure dependent operations. File-system code uses the buffer cache functions to request I/O on devices. This scheme is illustrated in the Figure 3.3

The VFS Structure

VFS defines a set of functions that every file-system has to implement [32]. This interface is made up of a set of operations associated to three kinds of objects, i.e. *file-systems, inodes,* and *open files.*

The VFS knows about file-system types supported in the kernel. It uses a table defined

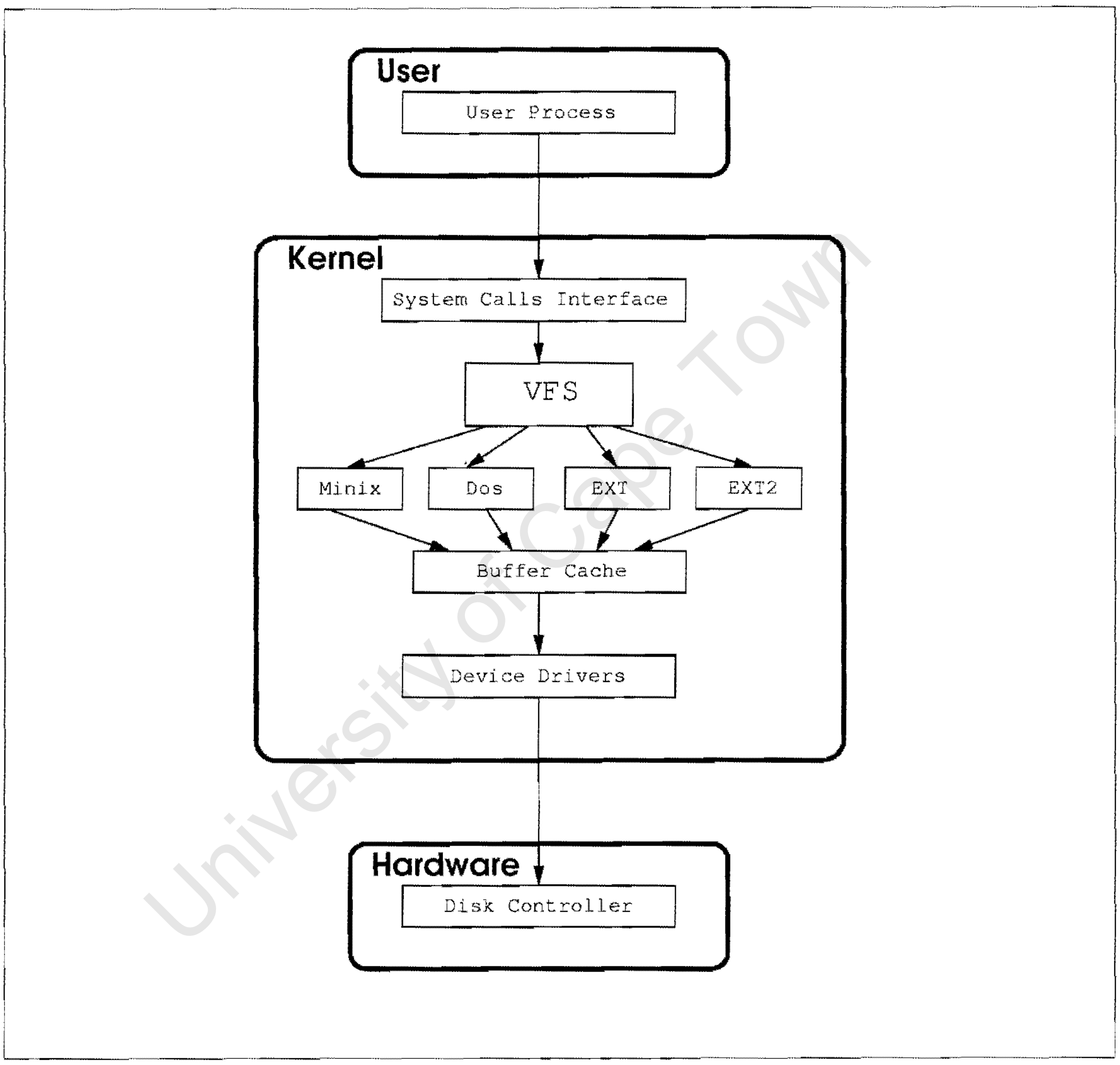


Figure 3.3: VFS Layout

during the kernel configuration. Each entry in this table describes a file-system type, it contains the name of the file-system type and a pointer to a function that is called during the mount operation. When a file-system is to be mounted, the appropriate mount function is called. This function is responsible for reading the super-block from the disk, initializing its internal variables, and returning a mounted file-system descriptor to the VFS. After the file-system is mounted, the VFS functions can use this descriptor to access the physical file-system routines.

A mounted file-system descriptor contains several kinds of data. This includes information that is common to every file-system type, pointers to functions provided by the physical file-system kernel code, and private data maintained by the physical file-system code. The function pointers contained in the file-system descriptors allow the VFS to access the file-system's internal routines.

Two other types of descriptors used by the VFS are : an inode descriptor and an open file descriptor. Each descriptor contains information that is related to the files in use and a set of operations provided by the physical file-system code. While the inode descriptor contains pointers to functions that can be used to act on any file (e.g. create, unlink), the file descriptors contain pointers to functions which can only act on open files (e.g. read, write).

3.2 Diskless Booting

Diskless booting is useful for computing clusters where the operating system is shared. This method of operation simplifies the administration and installation of multiple computers. There are several ways of achieving diskless booting over networks. Some of the ones outlined here are PXEs, Etherboot and GRUB .

Diskless booting can be useful in various situations [25]:

- An X-terminal.
- Clusters of compute servers.

- routers.
- Various kinds of remote servers, e.g. a tape drive server that can be accessed with the RMT (Remote MagTape) protocol.
- Machines doing tasks in environments unfriendly to disks.
- A user platform where remote partitions are mounted over the network and a lower speed compared to disk is acceptable.
- Maintaining software for a cluster of equally configured workstations centrally.

3.2.1 PXE

PXE is a protocol that was designed by Intel. According to Antoine Ginies[37], its major purpose is to allow computers to boot over a network. PXE is mostly supported by new generation network cards.

The PXE is stored in the ROM (Read Only Memory) of the network card. When the computer starts up, the BIOS (Basic Input Output System) loads the PXE ROM into main memory and executes it.

The PXE system will display a menu of options. A user can then select a desired operating system image to load over the network. Once a selection is made, an image is loaded up and thus the system boots up over the network.

3.2.2 Etherboot

Etherboot [25] is a software package that is useful for creating ROM images that enable diskless booting of computers. It downloads code over an ethernet network to be executed on an Intel x86 (Intel 80x86) computer. Etherboot makes use of the boot-ROM socket that is found on most ethernet cards. The code produced by Etherboot is written to a boot-ROM and then the ROM chip is slotted into a socket to enable network booting.

Among some of the advantages of using Etherboot are convenience and simplification of cluster architecture. Etherboot also has an advantage of booting computers faster than from a disk because there are no spin-up delays as those found with disks. A simple calculation will show that even with a 10 Mb/s ethernet, sending a 500 KB (Kilobyte) kernel will only take a couple of seconds typically. With 100 Mb/s ethernet it gets even better.

Etherboot is ideal for centralised administration since it loads up an operating system from a remote machine. In a cluster situation, this means that all the nodes can obtain their OS (Operating System) from a single server. The drawback is that this also centralises failure. The incidence of centralised failure can be countered somewhat by the use of a redundant server.

Etherboot can work with RAM-disks, NFS file-systems, or even local disks, if desired. It's a component technology and can be combined with other technologies to texture a desired model of operation.

Etherboot is usually used to load Linux, FreeBSD or DOS (Disk Operating System). However the Etherboot protocol and the boot file formats are general, so there is no reason why it could not be used to load arbitrary images to a workstation, including other OSes.

The components needed by Etherboot are:

- A bootstrap loader, usually in an EPROM (Erasable Programmable ROM) on a network card or a flash BIOS. Alternatively a boot-loader can be run from a floppy disk for test purposes.

- A DHCP or Bootp server, for returning an IP address in exchange of a MAC (Media Access Control) (on ethernet card) address.

- A TFTP (Trivial File Transfer Protocol) server, for sending the kernel images and other files required in the boot process. Alternatively, Etherboot can boot from an NFS mount.

- A Linux or FreeBSD kernel.

- Optionally, a NFS server, for providing the disk partitions that will be mounted if Linux or FreeBSD is being booted.
- Optionally, a RAM (Random Access Memory) disk contained in the loaded image. This can be the initial RAM disk if desired.
- Software tools for building the download image, and tools for debugging.

3.2.3 DHCP

DHCP provides configuration parameters to network nodes [26]. DHCP consists of two components:

1. a protocol for delivering host-specific configuration parameters from a DHCP server to a host and
2. a mechanism for allocation of network addresses to hosts.

DHCP is built on a client-server model. The DHCP server allocates network addresses and delivers configuration parameters to dynamically configurable hosts.

Throughout the remainder of this section, the term "server" refers to a host providing initialization parameters through DHCP, and the term "client" refers to a host requesting initialization parameters from a DHCP server.

DHCP supports three mechanisms for IP address allocation. These are shown in Table 3.1.

Protocol

The format of DHCP messages is based on the format of Bootp messages [26]. This allows inter-operability of existing Bootp clients with DHCP servers. Using Bootp relay agents eliminates the necessity of having a DHCP server on each physical network segment.

Table 3.1: DHCP mechanisms

Mechanism	What it Does
Automatic allocation	DHCP assigns a permanent IP address to a client.
Dynamic allocation	DHCP assigns an IP address to a client for a limited period of time (or until the client explicitly relinquishes the address).
Manual allocation	A client's IP address is assigned by the network administrator, and DHCP is used simply to convey the assigned address to the client.

Thus DHCP provides centralized configuration and maintenance of IP addresses. It enables the configuration of clients from a single location. DHCP permits IP address "leases" to be dynamically assigned to workstations, eliminating the need for static IP address allocation. Servers implementing DHCP server functionality maintain pools of available IP addresses.

3.2.4 GRUB

From Yoshinori K. Okuji [28], a boot loader is the first software program that runs when a computer starts. It is responsible for loading and transferring control to operating system kernel software (such as Linux or GNU -Mach). The kernel, in turn, initializes the rest of the operating system (e.g. a GNU Hurd system). GNU GRUB is a very powerful boot loader, which can load a wide variety of operating systems.

One of the important features in GRUB is flexibility; GRUB understands file-systems and kernel executable formats, so an arbitrary operating system can be loaded as desired, without recording the physical position of your kernel on the disk.

Thus you can load the kernel just by specifying its file name and the drive (and the partition) where the kernel resides. To let GRUB know the drive and the file name, either they can be typed in manually via the command-line interface, or by use of the menu interface, through which an OS boot can be selected. To allow customization of the menu, GRUB will load a pre-existing configuration file.

GRUB Features

According to Erich Boleyn[27], some important goals, listed in approximate order of importance, are:

- Basic functions must be straightforward for end-users.
- Rich functionality to support kernel experts and designers must be available.
- Backward compatibility for booting FreeBSD, NetBSD, OpenBSD, and Linux. Proprietary kernels (such as DOS, Windows NT, and OS/2) are supported via a chain-loading function.

Except for specific compatibility modes (chain-loading and the Linux piggyback format), all kernels will be started in much the same state as in the Multiboot Specification. Only kernels loaded at 1 megabyte or above are presently supported though.

Some of the other important features that GRUB has are:

- A menu interface listing the preset boot commands, with a programmable time-out, is available. There is no fixed limit on the number of boot entries, and the current implementation has space for several hundred.
- It has a flexible command-line interface that is accessible from the menu. It is available to edit any preset commands, or write a new boot command set from scratch. If no configuration file is present, GRUB drops to the command-line.
- It supports multiple filesystem types transparently, plus a useful explicit blocklist notation. The currently supported file-system types are BSD (Berkeley Software Distribution) FFS (Fast Filing System), DOS FAT16 (16 bit FAT) and FAT32 (32 bit FAT),

Minix, Linux EXT2, ReiserFS, JFS (Journalling File System), XFS (Extensible File System), and VSTa (Valencia Simple Tasker) .

- It supports network booting . GRUB is basically a disk-based boot loader but also has network support. You can load OS images from a network by using the TFTP protocol.

3.3 Clustering Technology

3.3.1 SSI

Single System Images are a means of combining resources in a cluster server [23] [24]. Cluster in this case means a NOW (Network Of Workstations), COW (Cluster of Workstations) or Beowulf type of combination. The main purpose of this is to reduce cognitive overhead required by the end users of one such a system. SSI is aimed at several levels of a cluster server. These are :

1. Hardware,
2. Operating system,
3. Message passing,
4. Language/compiler and
5. Tools

Single system images are supplied at one or more of these levels. Operating systems can be made to supply a single image on the process level with migration systems like MOSIX. Hardware can be dealt with using the help of distributed/parallel file systems or disk exportation tools like NBD. The availability of consistent message passing frameworks like MPI and PVM help in combining the view the programmer has of the system.

SSIs have numerous benefits. Some of these are: users are freed from knowing where an application will run, an operator does not have to know where a resource is located. SSIs can also greatly simplify management requirements.

3.3.2 MOSIX

From Amnon Barak et. al[16], we get that MOSIX is a software solution that can transform a cluster of PC's (workstations and servers) to run almost like an SMP (Symmetric Multi-Processor). The main advantages are simplicity of use and near optimal resource usage, i.e., when you create (one or more) processes in your log-in node MOSIX will distribute (and redistribute) your processes (transparently) among the nodes, to improve the performance [17].

According to Moshe Bar[21] the core of MOSIX are adaptive management algorithms that continuously monitor the activities of the processes versus the available resources, in order to respond to uneven resource distribution and to take advantage of the best available resources. The algorithms of MOSIX use pre-emptive process migration to provide:

- Automatic work distribution - for parallel processing or to move a process from a slower to a faster node.
- Load balancing - for even work distribution.
- Memory ushering - migrate processes from a node that will have run out of main memory, to avoid swapping or thrashing.
- High I/O performance - by migrating an intensive I/O process to a file server (rather than the traditional way of bringing the data to the process).
- Parallel I/O - by migrating parallel I/O processes from a client node to file servers.

The algorithms of MOSIX are decentralized - each node is both a master for processes that were created locally, and a server for processes that were assigned to it from other nodes.

The MOSIX algorithms are geared for maximal performance, overhead-free scalability and ease-of-use.

Scalability

MOSIX can support configurations with large numbers of computers, with minimal scaling overheads to impair the performance [16].

For example, the MOPI (MOSIX massive Parallel I/O) [18] package can deliver a high I/O performance by migrating parallel processes to the respective nodes that hold their data, e.g. different segments of a file. In addition to maximal throughput via a local cache, the MOSIX scheme does not saturate the network, thus it can be scaled up indefinitely. Scalability considerations were taken into account in almost all the algorithms of MOSIX.

MOSIX in Use

Production MOSIX systems around the world range from small clusters built from a few PC (Personal Computer)s, workstations and servers that are connected by Fast Ethernet, to high-end clusters, with hundreds of processors (e.g. SMP and non-SMP servers) that are connected by Gigabit-Ethernet. [15]

3.3.3 Beowulf

From Robert G. Brown[2], the accepted definition of a true Beowulf is that it is a cluster of computers ("compute nodes") interconnected with a network with the following characteristics:

1. The nodes are dedicated to the Beowulf and serve no other purpose.
2. The network on which the nodes reside is dedicated to the beowulf and serve no other purpose.
3. The nodes are MCOTS (Modified COTS) computers (this essential to the definition).

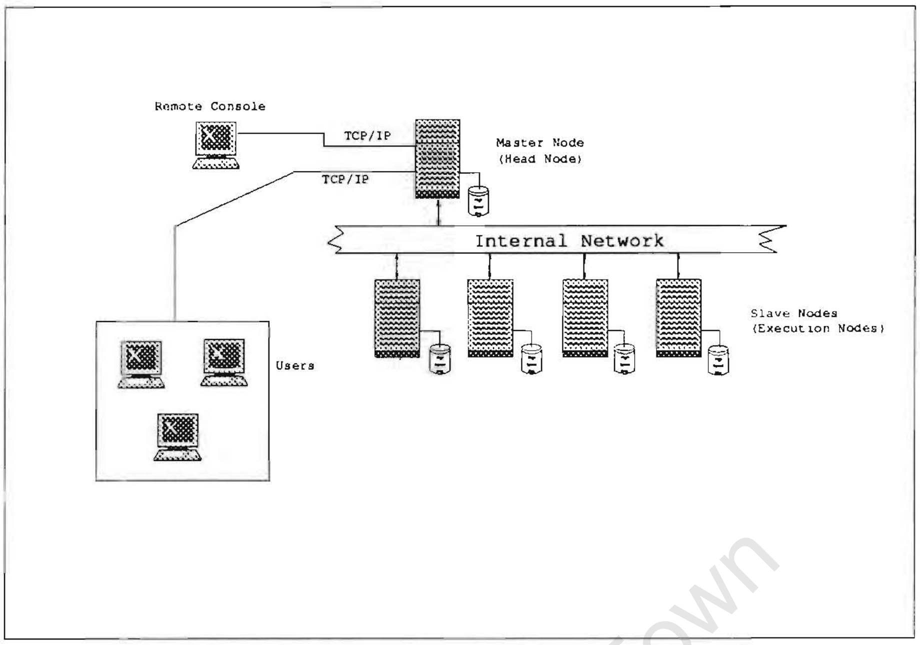


Figure 3.4: A Typical Beowulf Cluster

4. The network is also a COTS element, although it could also be a network specially made for Beowulfs.

5. The nodes all run open source software.

6. The resulting cluster is used for HPC (High Performance Computing) (also called "parallel supercomputing" and other names).

There is one "special" node generally called the "head" or "master" node. This node often has a monitor and keyboard and has network connections both on the "inside" network (connecting it to the rest of the "slave" nodes) and on the "outside" to a general purpose organizational inter-network. This node often acts as a server (e.g. for shared disk space). The design is shown in Figure 3.4

Generally the nodes are all identical - configured with the same CPU, motherboard, network, memory, disk(s) and so forth and are neatly racked up or stacked up in shelves in a single room in a spatially contiguous way. Also, the nodes are all running just one calculation at a time (or none).

Classification of a Beowulf

Quoting Phil Merkey[3]:

> In the taxonomy of parallel computers, Beowulf clusters fall somewhere between MPP (Massively Parallel Processor) (like the nCube , CM5 (Connection Machine 5), Convex SPP , Cray T3D , Cray T3E, etc.) and NOW. The Beowulf project benefits from developments in both these classes of architecture. MPPs typically are larger and generally have a tighter network architecture than a Beowulf cluster. Software production on a beowulf still deals with issues of locality, load balance, granularity, and communication overheads in order to obtain the best performance. Even on shared memory machines, many programmers develop their programs in a message passing style.

3.3.4 NOW

A NOW is a set of computers that is accessible over a network [4][5]. They differ from a Beowulf cluster in that the machines in a Beowulf cluster are dedicated to the cluster and only stand for the Beowulf computations. Whereas in a NOW the usual theme is making use of unused cycles on wider spread workstations (such as a computing lab). Thus NOWs are usually harder to measure their performance since it differs with the conditions of testing.

Another difficulty faced with NOWs is the software production, since a NOW usually contains a complex network architecture and the latency times between the different machines consisting the NOW are not quite the same, this usually constitutes timing problems during parallel programs.

Since machines in a NOW are usually visible to the outside world, this also makes them prone to security issues not dealt with by Beowulfs. The advantage of NOWs though is that they do make use of machines which are already around thus they usually present a cheaper alternative to buying dedicated compute nodes.

Programs that do not require fine-grain computation and communication can usually be ported and run effectively on Beowulf clusters. Programming a NOW is usually an attempt to harvest unused cycles on an already installed base of workstations in a lab or on a campus. Programming in this environment requires algorithms that are extremely tolerant of load balancing problems and large communication latency. Any program that runs on a NOW will run at least as well on a cluster.

3.4 LVS

According to Wensong Zhang[6][7], a virtual server is a scalable and highly available server built on a cluster of loosely coupled independent servers. The architecture of the cluster is transparent to clients outside the cluster. Client applications interact with the cluster as if it were a single high-performance and highly available server. The clients are not affected by interaction with the cluster and do not need modification.

The real servers may be interconnected by high-speed LAN (Local Area Network) or by geographically dispersed WAN (Wide Area Network). The front-end of the real servers is a load balancer, which schedules requests to the different servers and makes parallel services of the cluster appear as a virtual service on a single IP address. Scalability is achieved by transparently adding or removing a node in the cluster. High availability is provided by detecting node or daemon failures and re-configuring the system appropriately.

3.4.1 Virtual Server Architectures

From the LVS-HOWTO [13], LVS is available in three architectures . These are:

1. LVS via NAT (Network Address Translation) (LVS-NAT)
2. LVS via TUN (IP Tunelling) (LVS-TUN)
3. LVS via DR (Direct Routing) (LVS-DR)

Virtual Server via NAT

Due to the shortage of IP addresses in IPv4 (IP version 4) [9] and some security reasons, more and more networks use private IP addresses which cannot be used outside the network (or in the Internet). The need for network address translation arises when hosts in internal networks want to access the Internet or to be accessed by the Internet. Network address translation relies on the fact that the headers for Internet protocols can be adjusted appropriately so that clients believe they are contacting one IP address, but servers at the different internal IP addresses believe they are contacted directly by the clients. This feature can be used to build a virtual server, i.e. parallel services at the different IP addresses can appear as a virtual service on a single IP address via NAT.

The architecture of virtual server via NAT is illustrated in Figure 3.5. The load balancer and real servers are interconnected by a switch or a hub. The real servers usually run the same service and they have the same set of contents. The contents are either replicated on each server's local disk, shared on a network file-system, or served by a distributed file system (such as AFS (Andrew File System) or CODA FS (file system)). The load balancer dispatches requests to different real servers via NAT.

Virtual Server via IP Tunneling

From LVS-Tunneling HOWTO [10] IP tunneling (IP encapsulation) is a technique to encapsulate IP datagram within IP datagrams, which allows datagrams destined for one IP address to be wrapped and redirected to another IP address. This technique can be used to implement a virtual server where the load balancer tunnels the request packets to the different real servers. The real servers process the requests and return the results to clients directly, and the services can still appear as a virtual service on a single IP address.

In the layout of LVS-TUN shown in Figure 3.6, the real servers can have any real IP address in any network, they can be geographically distributed, but they must support IP encapsulation protocol and they must all have one of their tunnel devices configured

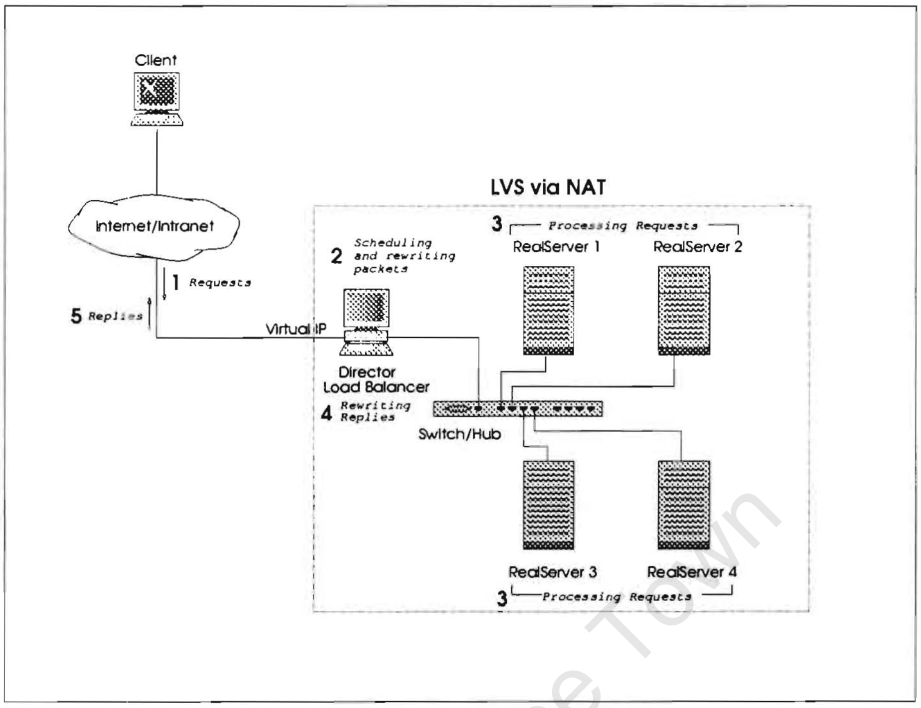


Figure 3.5: Architecture of a virtual server via NAT

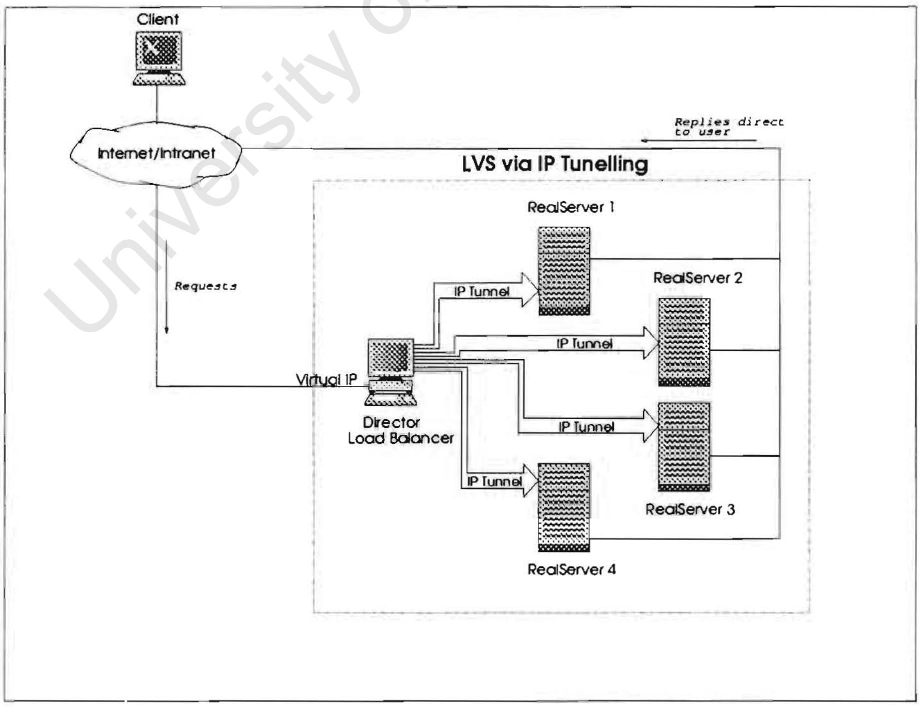


Figure 3.6: Architecture of a virtual server via IP Tunelling

with <Virtual IP Address>, like "ifconfig tunl0 <Virtual IP Address>" in Linux. When the encapsulated packet arrives, the real server decapsulates it and finds that the packet is destined for <Virtual IP Address>, so it processes the request and returns the result directly to the user in the end.

LVS-DR

LVS-DR's working principle is based on the request dispatching approach [11]. This request dispatching approach is similar to the one implemented in IBM (International Business Machines)'s NetDispatcher. The NetDispatcher served HTTP (Hyper-Text Transfer Protocol) pages for the Atlanta and the Sydney Olympic games and for the chess match between Gary Kasparov and Deep Blue. The VIP (Virtual IP) address is shared by real servers and the load balancer. The load balancer has an interface configured with the virtual IP address. This is used to accept request packets. These packets are directly routed to the chosen servers. All the real servers have their non-ARP (Address Resolution Protocol) alias interface configured with the virtual IP address or redirect packets destined for the virtual IP address to a local socket, so that the real servers can process the packets locally. The load balancer and the real servers must have one of their interfaces physically linked by an ethernet hub or a switch. The architecture of virtual server via direct routing is shown in Figure 3.7

LVS-DR setup and testing is the same as LVS-Tun except that all the machines within the LVS-DR (i.e. the director and real-servers) must be able to communicate using ARP to each other. This means that they have to be on the same network without any forwarding devices between them. Thus they use the same piece of transport layer hardware ("wire"), e.g. RJ-45 (Registered Jack 45), coaxial cable or optical fibre. There can be hub(s) or switch(es) in this mix. Communication within the LVS is by link-layer, using MAC addresses rather than IP's. All machines in the LVS have the VIP, only the VIP on the director replies to ARP requests, the VIP on the real-servers must be on a non-ARPing device (for example the loopback device lo:0 or the dummy network device dummy).

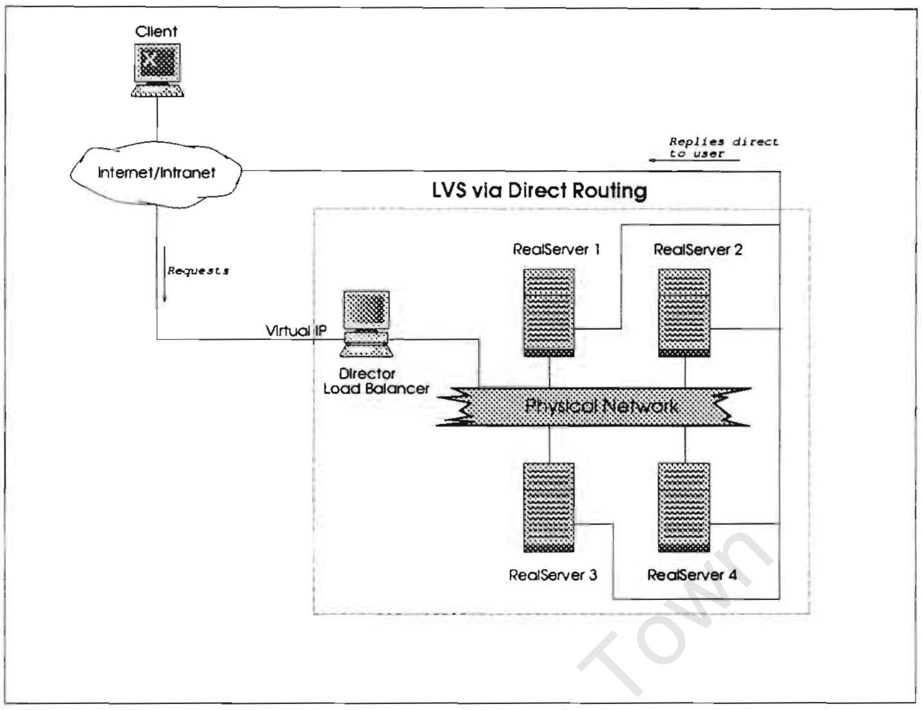


Figure 3.7: Architecture of LVS-DR

3.4.2 Advantages and Disadvantages

The following are the advantages and disadvantages of the various types of LVS available as mentioned in the LVS-HOWTO [13].

Virtual Server via NAT

The advantage of the virtual server via NAT [9] is that real servers can run any operating system that supports TCP/IP (TCP over IP) protocol, real servers can use private Internet addresses, and only one outside IP address is needed (for the load balancer).

The disadvantage is that the scalability of the virtual server via NAT is not very good. The load balancer may be a bottleneck of the whole system when the number of server nodes increases to around 25 or more. This is also dependent on the throughput of real servers, because both the request packets and the reply packets need to be rewritten by the load balancer. Supposing the average length of TCP (Transmission Control Protocol) packets is 536 bytes, then the average delay of rewriting a packet is around $60\mu s$ (this on the Pentium I processor, the use of a faster processor can counter this). If the maximum throughput of the load balancer with 10 Mb/s ethernet is 8.93 MB/s, then the load balancer can schedule 22 real servers if the average throughput of each real server is 400 KB/s (Kilobyte per second).

Virtual server via IP Tunneling

For many Internet services (such as web services) the incoming packets are short and reply packets tend to be much larger. In the virtual server via IP tunneling, the load balancer just schedules requests to the different real servers, and the real servers return replies directly to the clients [10]. Thus the load balancer can handle a huge amount of requests. It is able to schedule hundreds of real servers without becoming a bottleneck of the system. Therefore, using IP tunnelling, there can be more real servers and the maximum throughput of the virtual server may reach over 1 Gb/s, even if the load balancer just has 100 Mb/s full-duplex network adapter.

An example of an Internet service that benefits from LVS-TUN is a proxy cache server. This is because when the real proxy servers receive requests, they can access the Internet directly to fetch objects and return them directly to the users.

The drawback of virtual server via IP tunneling is that it requires that every real server must support the IP Tunneling protocol. This places a restriction on the operating system that can be used. The system has been tested with servers running Linux. It should also be available on other OSes.

LVS-DR

LVS-DR is similar to LVS-TUN in that there is a direct connection between the real servers and the client. Thus the expected handling capacity should be the same [11].

The restrictions for LVS-DR are:

- The client must be able to connect to the VIP on the director

- Real servers and the director must be on the same piece of wire (they must be able to ARP each other) as packets are sent by link-layer from the director to the real-servers.

- The route from the real-servers to the client cannot go through the director, i.e. the director cannot be the default gateway for the real-servers. (Note: the client does not need to connect to the the real-servers for the LVS to function. The real-servers could be behind a firewall, but the real-servers must be able to send packets to the client). The return packets, from the real-servers to the client, go directly from the real-servers to the client and do not go back through the director.

For high throughput, each real-server can have its own router/connection to the client/Internet and return packets need not go through the router feeding the director.

3.4.3 Scheduling Algorithms

Four scheduling algorithms are available for selecting a server from the cluster [12], these are:

1. Round-Robin,

2. Weighted Round-Robin,

3. Least-Connection and

4. Weighted Least-Connection.

The first two algorithms do not take into account any load information about the servers. The last two algorithms count active connection number for each server and estimate their load based on those connection numbers.

The following are extracts from LVS documentation on the scheduling algorithms available [12].

Round-Robin Scheduling

Round-robin scheduling algorithm, in its word meaning, directs the network connections to the different servers in a round-robin fashion. It treats all real servers as equals regardless of number of connections or response time.

Weighted Round-Robin Scheduling

The weighted round-robin scheduling takes into account the different capabilities of the real servers. Each server can be assigned a weight, an integer that indicates its processing capacity. The default weight is 1. Suppose there are three real servers, **A**, **B** and **C**, with weights, 4, 3, 2 respectively, then a good scheduling sequence will be **ABCABCABA**.

In the implementation of the weighted round-robin scheduling, a scheduling sequence will be generated according to the server weights after the rules of virtual server are modified. The network connections are directed to the different real servers based on the scheduling sequence in a round-robin manner.

The weighted round-robin scheduling algorithm does not need to count the network connections for each real server, therefore the overhead of scheduling is smaller than that for dynamic scheduling algorithms. This means that it can have more real servers.

However, a system using this algorithm may suffer from dynamic load imbalance among the real servers. This is especially true if the request load varies highly.

Least-Connection Scheduling

The least-connection scheduling algorithm directs network connections to the server with the lowest number of active connections. This is one of the dynamic scheduling algorithms, because it needs to count the active connections for each server dynamically. At a virtual server where there is a collection of servers with similar performance, the least-connection scheduling is good for smoothing distribution when the load of requests varies a lot. This is because all long requests will not have a chance to be directed to a server.

It may appear as if the least-connection scheduling can also perform well even when there are servers of various processing capacities. Unfortunately, it cannot perform very well because of the TCP's **TIME_WAIT** state. The TCP's TIME_WAIT is usually 2 minutes, between this 2 minutes a busy web site may get thousands of connections. For example, suppose there is a server **A** that is twice as powerful as another server **B**. The server **A** has processed thousands of requests and kept them in the TCP's TIME_WAIT state, but the server **B** is lagging to get its thousands of connections finished. So, the least-connection scheduling cannot get load well balanced among servers with varying processing capacities.

Weighted Least-Connection Scheduling

The weighted least-connection scheduling is a super-set of the least-connection scheduling. With this system a performance weight can be attached to each real server. The servers with a higher weight value will then receive a larger percentage of active connections at any one time. The virtual server administrator can assign a weight to each real server, and network connections are scheduled to the server where the percentage of the current number of active connections for each server is in ratio to its weight. The default weight is one.

The weighted least-connections scheduling works as follows:

> Supposing there is n real servers, each server i has weight $W_i (i = 1, .., n)$, and active connections $C_i (i = 1, .., n)$, ALL_CONNECTIONS is the sum of $C_i (i = 1, .., n)$, the network connection will be directed to the server j, in which
>
> $(C_j/\text{ALL\_CONNECTIONS})/W_j = \min \{ (C_i/\text{ALL\_CONNECTIONS})/W_i \} \; (i = 1, .., n)$
>
> Since the ALL_CONNECTIONS is a constant in this lookup, there is no need to divide C_i by ALL_CONNECTIONS, it can be optimized as
>
> $C_j/W_j = \min \{ C_i/W_i \} \; (i = 1, .., n)$

The weighted least-connection scheduling algorithm requires additional division than the least-connection. In a hope to minimize the overhead of scheduling when servers have the same processing capacity, both the least-connection scheduling and the weighted least-connection scheduling algorithms are implemented.

3.5 SSH

SSH (Secure SHell) is a secure remote terminal program that is made by SSH Security Communications Corp. It allows for the remote access of computers over TCP/IP networks. According to the makers [52], the main security requirements which SSH observes are :

- Confidentiality - The transmitted data must not be readable by unauthorized parties on the network. Confidentiality is achieved through encryption.
- Integrity - Unauthorized parties must not be able to modify the data without detection. Integrity is achieved by using check-sum values, which reveal tampering attempts at the receiving end.
- Authentication - Both parties of the communication must be able to identify each other reliably, so that no one can masquerade as the other party.

The SSH protocol supports the strongest encryption methods available to date. These algorithms are: 3DES (Triple DES), CAST-128 (CAST 128 bit Encryption), Twofish, AES (Advanced Encryption Standard) and DES (Digital Encryption Standard).

SSH also supports tunnels. These are a very powerful feature in providing secure connections since insecure applications can be accessed securely. Tunnels can be used on virtually any TCP/IP based service such as e-mail, X (X-window) applications or terminals.

3.6 IP Masquerading

IP Masquerading refers to the use of a single IP address on several machines [14]. With this, several machines can access the Internet using one real IP address.This idea is useful in providing an Internet gateway to machines on an internal network, where real IP addresses are not available for the machines behind such a gateway. IP masquerading is especially useful as a fire-walling method since it usually incorporate packet filtering. Thus packets are analysed during input and output for content. All packets which do not comply with the fire-walling rules will be discarded.

IP Masquerading generally makes use of NAT. Network address translation means that the IP section of a packet is rewritten during input and output as to change the machine it is coming from. Thus if an address like `192.168.0.1` goes through a gateway with address `156.132.128.3` then it gets the gateway address on its outgoing packets. On return the reply packets are rewritten back to `192.168.0.1` so that they can reach their rightful owner.

On Linux systems IP masquerading has been implemented with IPTables and IPChains. IPTables is the newer of the two packages and thus IPChains is now somewhat obsolete. IPTables allows the creation of fire-walling rules for use with such gateways. These rules allow the implementation of packet transfer between input and output network interfaces. It allows the use of a single interface between several networks so that such an interface can act as a gateway between the networks.

IPTables allows several fire-walling rules to be put together to customize the behaviour of a given network fire-wall.

3.7 Distributed File Systems

Peter J. Braam [50] tells us that a DFS stores files on one or more computers called servers, and makes them accessible to other computers called clients, where they appear as normal files. There are several advantages to using file servers:

- The files are more widely available since many computers can access the servers,
- Sharing the files from a single location is easier than distributing copies of files to individual clients.

- Backup and safety of the information are easier to arrange since only the servers need to be backed up.

- The servers can provide large storage space, which might be costly or impractical to supply to every client.

A DFS becomes more useful in the case of a group of people sharing a document [50]. At the same time it is also possible to share applications software. In both cases administration is more centralised and thus is simplified.

The design of a good DFS has to deal with many issues. Some of these are problems of efficiency. Since the transportation of large amounts of data over a network can easily produce sluggish performance and high latency. Servers could easily become network bottlenecks and overloads may occur. A second important consideration is security. In this case unauthorised clients can gain access to data and perhaps even corrupt it. Other issues to deal with are those of reliability and resilience of the file system during network failures.

3.7.1 CODA

The CODA distributed file system is a state of the art experimental file system developed in the group of M. Satyanarayanan at Carnegie Mellon University [50]. The CODA project incorporates many features, some of which are not found in other systems. According to the developers, some of the features available in CODA are shown in the Table 3.2.

Volumes, Servers and Server Replication

Files on CODA servers are not stored in traditional file systems [50]. Partitions on the CODA server workstations can be made available to the file server. These partitions will contain files which are grouped into volumes. Each volume has a directory structure similar to that of a file system, i.e. a root directory for the volume and a tree below it. A volume is much

Table 3.2: CODA Features

Mobile Computing	Disconnected operation for mobile clients
	Re-integration of data from disconnected clients
	Bandwidth adaptation
Failure Resilience	Read/write replication servers
	Resolution of server/server conflicts
	Handling of network failures which partition the servers
	Handles disconnection of clients
Performance and scalability	Client side persistent caching of files,
	Directories and attributes for high performance write back caching
Security	Kerberos like authentication
	ACL (Access Control Lists)'s
Others	Well defined semantics of sharing
	Freely available source code

smaller than a partition, but is much larger than a single directory and is a logical unit for files. An example of a CODA volume could be a user's home directory. A single server would normally hold some hundreds of volumes, perhaps with an average size approximately 10 MB (Megabyte). A volume is a manageable amount of file data according to the perspective of the administrator.

CODA holds volume and directory information, ACLs and file attribute information in raw partitions. These are accessed through a log based RVM (Recoverable Virtual Memory) package for speed and consistency. Only file data resides in the files in server partitions. RVM has built in support for transactions - this means that in case of a server crash the system can be restored to a consistent state without much effort. Coda has read/write replication servers, i.e a group of servers can hand out file data to clients, and generally updates are made to all servers in this group. The advantage of this is higher availability of data: if one server fails others take over without a client noticing the failure. Volumes can be stored on a group of servers called the VSG (Volume Storage Group). The VSG is a list of servers which hold a copy of the replicated volume.

3.7.2 MFS

MFS [19] is a distributed file system that comes with the MOSIX [16][17][20] package. Since MOSIX gives a transparent means of migrating processes, it can balance out load and give an even distribution of load in a cluster. This is well and true for CPU bound processes. Unfortunately for I/O bound processes, this would mean accessing the I/O on the home node on which the process is created and transferring this data to a remote node where the process has migrated to. On a system already using a remote file system (networked), this also means that I/O will almost always be remote. This has negative performance implications for the network and the computation itself.

MFS provides a feature called DFSA (Direct File System Access). This feature essentially means that MOSIX can take into account the I/O requirements when making calculations

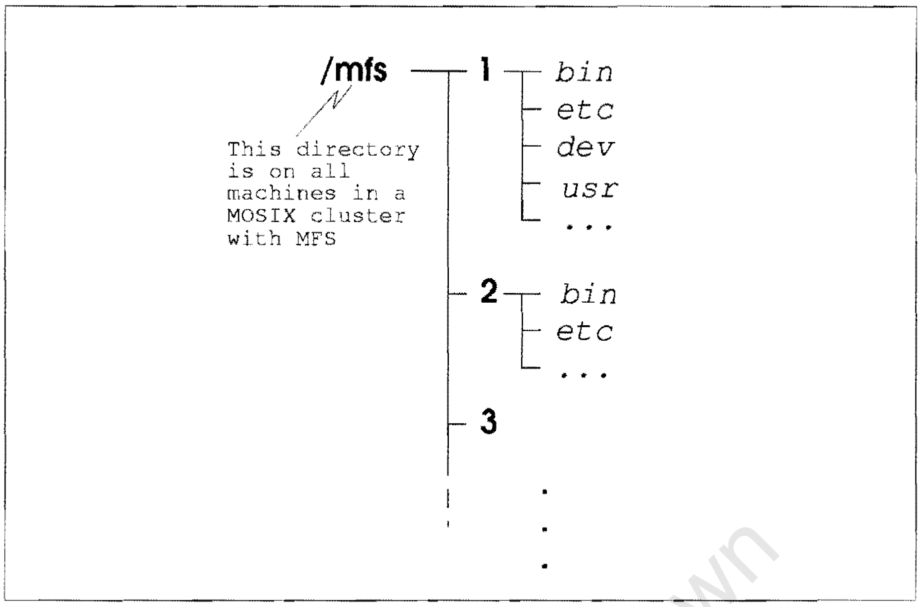

/mfs
This directory is on all machines in a MOSIX cluster with MFS
1
bin
etc
dev
usr
...
2
bin
etc
...
3

Figure 3.8: File Tree Structure of MFS

of where to move a process to. Thus it can move processes to the machine on which the data resides. Essentially, this is moving a process to the data.

Accessing MFS

The MFS subsystem provides a directory tree accessible under the mount point **`/mfs`**. All the machines in the MOSIX cluster that are configured to make use of MFS should have identical directory structures for the files and directories which are used by migrating processes. Under **`/mfs`**, a particular node's directory tree is accessible under a directory named by its MOSIX identity. As an example of this structure, the first machine in the cluster will have MOSIX identity `1`, thus its tree will be found under the **`/mfs/1/`** directory. Figure 3.8 will give a better illustration on this structure.

3.7.3 NFS

The NFS allows files and directories to be shared across a network [39][51]. Originally, it was developed by Sun Microsystems, but is now supported by virtually all operating systems.

Through NFS, clients are able to access files located on remote servers as if they are stored locally. In an ideal NFS environment, the user neither knows or cares where the files are actually stored.

Craig Hunt [51] mentions NFS as having the following several benefits:

- It reduces local disk storage requirements because a network can store a single copy of a directory, while the directory continues to be fully accessible to everyone over the network.
- NFS simplifies central support tasks, since files can be updated centrally.
- NFS allows users to use familiar Unix commands to manipulate remote files instead of learning new commands. There is no need to use `ftp` or `rcp` to copy a file between hosts on a network; `cp` works just fine.

There are two sides to NFS, i.e. a client side and a server side. The client side is the system that uses the remote directories as if they were part of the local file system. The server is the system that makes the directories available for use. Attaching a remote directory to the local file system (a client function) is called *mounting* a directory. Offering a directory for remote access (a server function) is called *sharing* a directory (or *exporting*).

3.7.4 ClusterNFS

ClusterNFS is a set of patches for the UNFSD (Universal NFS Daemon) server [43][44]. Its main purpose is to allow multiple diskless clients to NFS mount the same root filesystem by providing "interpreted" file names.

When a client requests the file:

```
/path/filename
```

the ClusterNFS server checks for the existence of files of the form:

```
/path/filename$$KEY=value$$
```

If such a file exists and the clients has a matching value for **KEY**, this file is returned. If the client does not have the matching value or no such file exists, the file request proceeds as normal. Currently supported keys include, **HOST** (hostname), **IP** (IP number), **UID** (user id), **GID** (group id), and **CLIENT** (matches any NFS client).

By naming all machine-specific files `filename$$IP=aaa.bbb.ccc.ddd` and naming files which are the same for all clients `filename$$CLIENT$$`, the server and all clients can share the same root partition. This makes it easy to set up and maintain a pool of diskless machines.

How it Works

The property of ClusterNFS which allows it to do this is *name translation*. Once the daemon is started with the `--translate-names` or `-T` option, then it will translate all tagged files. A tag is an additional string that is appended to a file or directory name that gives a certain special meaning to the NFS daemon. In the introduction above a tag is the statement `$$KEY=value$$`. Valid tags are shown in Table 3.3 [43].

If a matching file is located, but authorization is denied,this is not considered a match, and the next entry on the list will be attempted.

To access the "base" file or directory, rather than the machine specific version, simply append `$$$$` to the file-name. This also causes two look-up attempts, first using the file-name without the trailing `$$$$`, then with the trailing `$$$$`. The latter occurs in case there is a file actually named `filename$$$$`.

The ClusterNFS system supports all file and directory actions (i.e. read, create, link, softlink, getattrib, setattrib, rename, remove, copy, mkdir, rmdir). mount requests also support this scheme via a similar option on **mountd**.

Also available in ClusterNFS is the ability to control the names of created/renamed/linked files via the presence of a file with the tag `$$CREATE=<tag>$$`. This means that when an

Table 3.3: ClusterNFS Keys

Filename	**Matches if:**
`/path/filename$$UID=xxxx$$`	Matches if the user ID (identity) of the user accessing the file is `xxxx`.
`/path/filename$$GID=yyyy$$`	Matches if the current group ID of the user accessing the file is `yyyy`.
`/path/filename$$HOSTNAME=ssss$$`	Matches if the hostname of the client making the request is `ssss`.
`/path/filename$$IP=a.b.c.d$$`	Matches if the host IP number of the client making the request is `a.b.c.d`.
`/path/filename$$CLIENT$$`	Always matches on access through the NFS server.
`/path/filename`	Always matches.

attempt is made to create a file named `myfile` the NFS server will check for the presence of a file named `myfile$$CREATE=<tag>$$`. If such a file exists, `myfile$$tag=value$$` will be created. If not, the file creation will proceed as normal. Creating a file named `$$CREATE=<tag>` (no base file-name) would control the name of all created files in the directory.

3.7.5 Network Block Device

The NBD (Network Block Device) provides a means of transferring raw hard disk partitions over a network [36].

Quoting P. T. Breuer [35]:

> Basically they define a virtual set of wires to a remote device.

NBDs take advantage of the VFS layer to define a network driver for a virtual hard drive. Since the devices use the VFS layer, all that NBD defines is a driver for the network. After the driver is attached to the kernel, it operates like any other fixed disk driver. It defines a client-server type of connection to its remote counterpart. The machine with the actual storage device will have a server while the machine which uses the device will be a client.

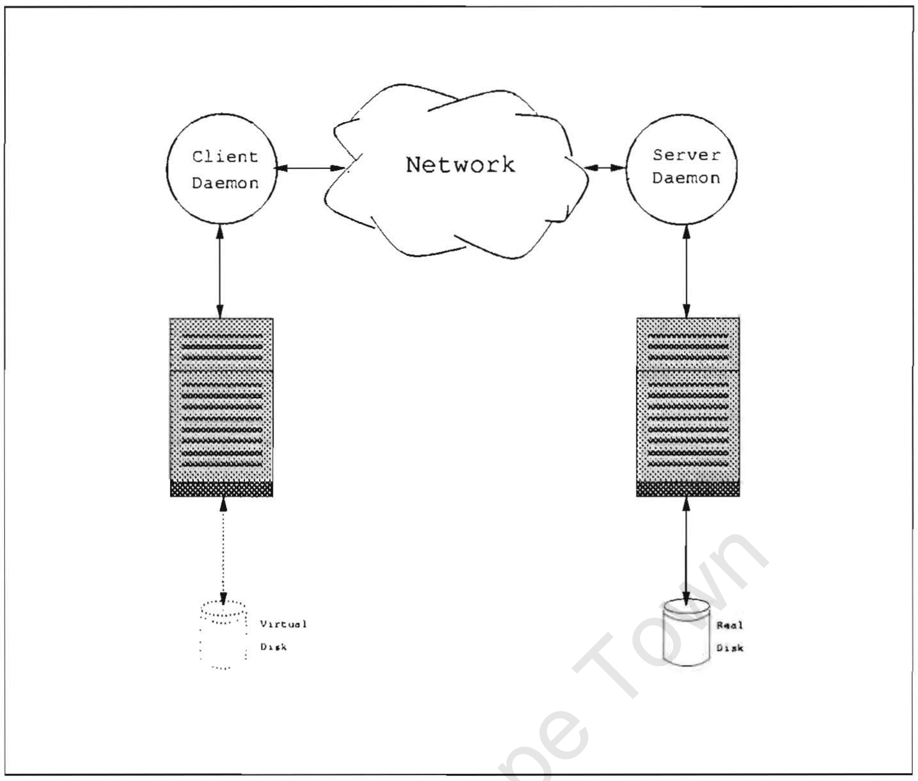


Figure 3.9: NBD Layout

The NBD system consists of three main parts. These are: a server, a client and a kernel device driver. Thus the server connects to the physical device via the VFS layer, while the driver defines the virtual block devices. Since the NBD driver connects via the VFS layer it also behaves just as if it were any other storage device. Thus it can be re-exported as another NBD device.

After a network block device is successfully exported and is made to run on a remote machine it can be formatted, or combined in a RAID array or in logical volumes via the LVM (Logical Volume Manager) [36].

The NBD Protocol

According to the Linux kernel documentation in the Linux kernel sources [31], the NBD system defines a packet format that is shown in Figure 3.10. This allows the definition of all the necessary hard drive semantics for export over the network.

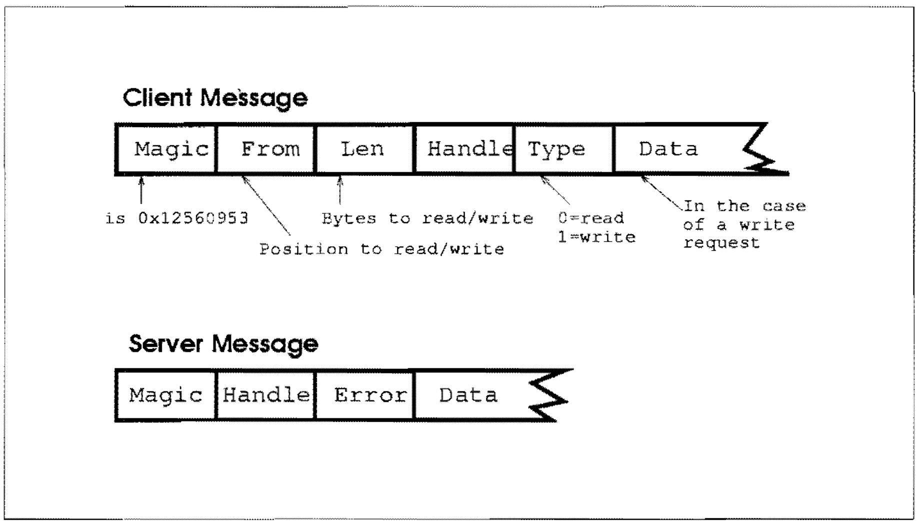


Figure 3.10: The NBD Packet Breakdown

3.7.6 Linux Software RAID

RAID is a means of combining hard disk drives so that their storage space can be accessible as one volume [47][48]. At the same time depending on the configuration, then there can also be some data redundancy built in so that there can be some increased data redundancy to go along with the data.

RAID systems are defined in what are termed as *levels* or *personalities*. The most widely used RAID personalities are summarised in Table 3.4

RAID systems are usually defined as hardware devices by means of specialised boxes or also with the use of special controllers. These offer the advantages of simplified maintenance and extremely high speeds for data access. In such devices high speeds are gained by the use of parallelism in certain combinations of the drives.

The Linux software RAID (shall be termed **sRAID** in this text) system is a realisation of such systems by use of software [49]. This system allows the use of conventional hardware. It combines the storage devices in a computer system so that they can give an image of a single hard disk drive. The sRAID system takes advantage of the VFS layer in defining the devices. Thus the sRAID (software RAID) system can be layered. With such, then it is possible to combine sRAID devices in higher level sRAID devices, and even combine them

Table 3.4: Summary of Common RAID Levels

Linear	This level is not redundant. Data is stored on the drives in turn as each one gets full. Performance is only as good as the drive being accessed.
Level 0	This level is also not redundant. With this, data is striped across drives, resulting in higher data throughput. No redundant information is stored thus performance is very good, but the failure of any disk in the array results in data loss.
Level 1	Level 1 provides redundancy by writing all data to two or more drives. The performance of a level 1 array tends to be faster on reads and slower on writes compared to a single drive, but if either drive fails, no data is lost. On the other hand, the cost per megabyte is high. This level is known as mirroring.
Level 4	This Level stripes data at a block level across several drives, with parity stored on one drive. The parity information allows recovery from the failure of any single drive. The performance of a level 4 array is very good for reads, but writes, however, are slow. Because only one drive in the array stores redundant data, the cost per megabyte of a level 4 array can be fairly low.
Level 5	This level is similar to level 4, but distributes parity among the drives. This can speed small writes in multiprocessing systems, since the parity disk does not become a bottleneck. Because parity data must be skipped on each drive during reads, however, the performance for reads tends to be considerably lower than a level 4 array. The cost per megabyte is the same as for level 4.

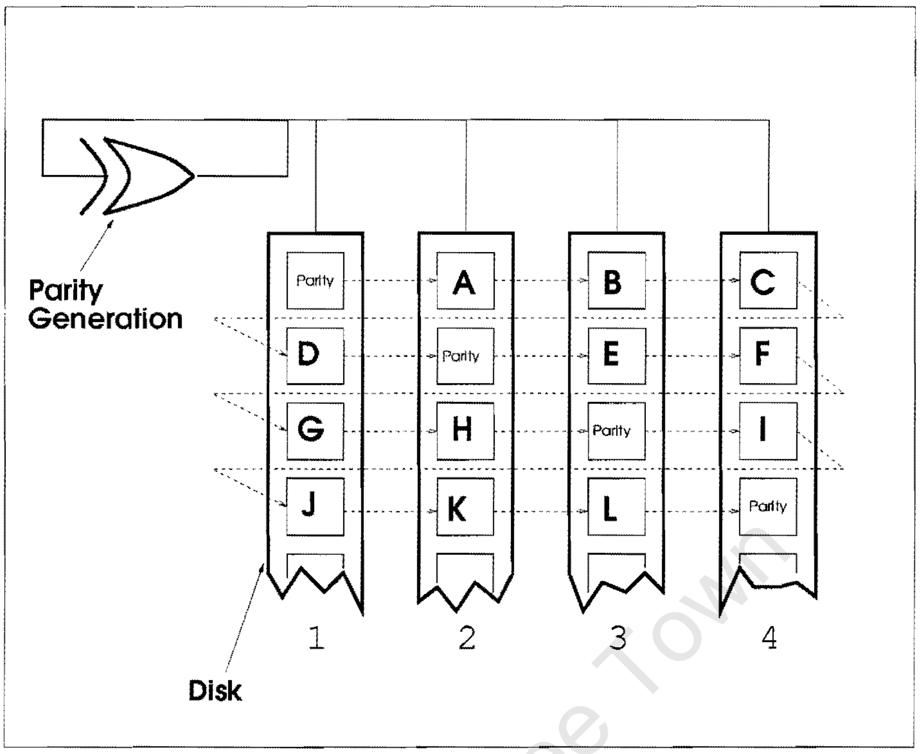


Figure 3.11: Simplified Operation of RAID-5

with other storage devices to give a myriad of possibilities.

The sRAID system requires the use of a kernel device driver and therefore requires reconfiguration of the kernel in order to get it to work. The personalities are then defined by a file called `/etc/raidtab` which allows the definition of the required settings.

Chapter 4

Implementation of the System

This chapter is a discussion on the actual building of the cluster computer. It details the components used and how they were setup for their purpose.

This chapter is divided into two main parts. The first part is an overview of the components making up the cluster. This part is known as *Conceptual Overview.* This first part merely states the components used and serves as an introduction to the assembly of the cluster computer.

The second part is a detailed explanation of the implementation of the components.

4.1 Conceptual Overview

In this cluster we aim to bring together the components making up the cluster in various levels of unity. The three major concepts requiring unity are *user* (i.e. single computer image),*process* (non-locality of processes) and *storage* (disk) implementation.

Thus we seek to combine the processing power of the machines and their storage to give a SSI [23][24]. This means that the user sees one machine and does all their work with this view. Yet many machines pitch in to lighten the load.

The cluster is split into 4 main sections that can be focussed on. These are :

1. I/O and display
2. Processor
3. Local Storage (hard disk)
4. Mass storage

4.1.1 Hardware Setup

There were a number of hard disk drives that were available for use in this project. Each one of these disks is an 18 GB (Gigabyte) 7200 rpm (Revolutions Per Minute) SCSI drive (See Appendix D). Before use there had to be assurance of the functioning of the drives. After doing some tests, 16 disks were prepared for use in the cluster. These hard drives throw off quite a bit of heat, thus their fitment came with some changes to the machines. For each computer in the cluster, two hard drive coolers were added. This means one cooler for each disk drive. Also added to this setup was a case fan per node machine. At the end of the day, there are two 18 GB drives in each machine totalling 36 GB per node.

4.1.2 Software Setup

It is all nice putting together hardware and talking about performance data. To make this all really practical meant using a small collection of utilities together with an appropriate configuration.

Operating System

The main server had Redhat operating software installed. The version used in this was RedHat 7.3 [53]. This distribution comes with kernel version 2.4.18 which also happened to be the current version for use with MOSIX.

Process Load Balancing

Process distribution is done by the MOSIX multicomputer software. The job of MOSIX is to make sure that whenever a computer in the cluster is getting heavily loaded it moves some of the processes off to another machine thus easing the load on that computer and giving a better balance of load.

Storage Distribution

Storage is combined with the help of NBD, the network block device. NBD exports a storage device over a network to give a view on a remote client of local storage. Once exported this device can be formatted or combined into RAID arrays, just like a normal disk.

Shared Operating System

To simplify administration and give a good single view, all the nodes from node 1 to node 7 boot over the network using a combination of GRUB, DHCP, TFTP and ClusterNFS. DHCP gives the node network configuration which GRUB uses to locate a server carrying a kernel. The kernel is copied over the network via TFTP. After the kernel is copied and started up, it uses DHCP yet again to setup its own network parameters and get information on the location of its network root file system.

Diskless Booting

Grub is the next important step in the system. GRUB is the program code that installs itself in the master boot record and loads up the operating system. If compiled with the right options, it will load up the OS over a network instead of using a local hard drive. Thus on start-up, it detects the network parameters via DHCP . Once it has a TFTP server location, GRUB loads in the kernel and the rest is up to the kernel to continue.

System Access and Display

IO and display is handled by LVS software, so far it is set up using LVS-NAT. LVS-NAT is the simplest setup of Virtual Server available on Linux. But it also has the greatest disadvantage and is the most unscalable.

LVS allows a collection of several real servers to appear as one machine in order to distribute network based services to clients. This is not the same as the type of distribution offered by MOSIX since MOSIX is targeted to transparent process migration of processor demanding processes.

SSH

SSH provides the main point of entry to this cluster. SSH provides X tunneling via the SSH port 22. Therefore once the SSH software is running, displays can be exported to clients.

System Layout

The general layout of machines and the main services in the Gollach cluster is summarised in Figure 4.1

4.2 Hardware Implementation Details

The only hardware we had to worry about within this cluster was the hard disk drives that were going to be inside each machine. Ethernet and switch setup were as expected, straightforward connections.

Each machine in our cluster carried two 7200 rpm hard disks. We intended to get the most out of our storage devices, thus we had to set them up in the most advantageous way. Since the SCSI controllers in our machines are rated at 40 MB/s and the disks are specified to be around 16 MB/s, then a controller should be able to handle two disks easily .

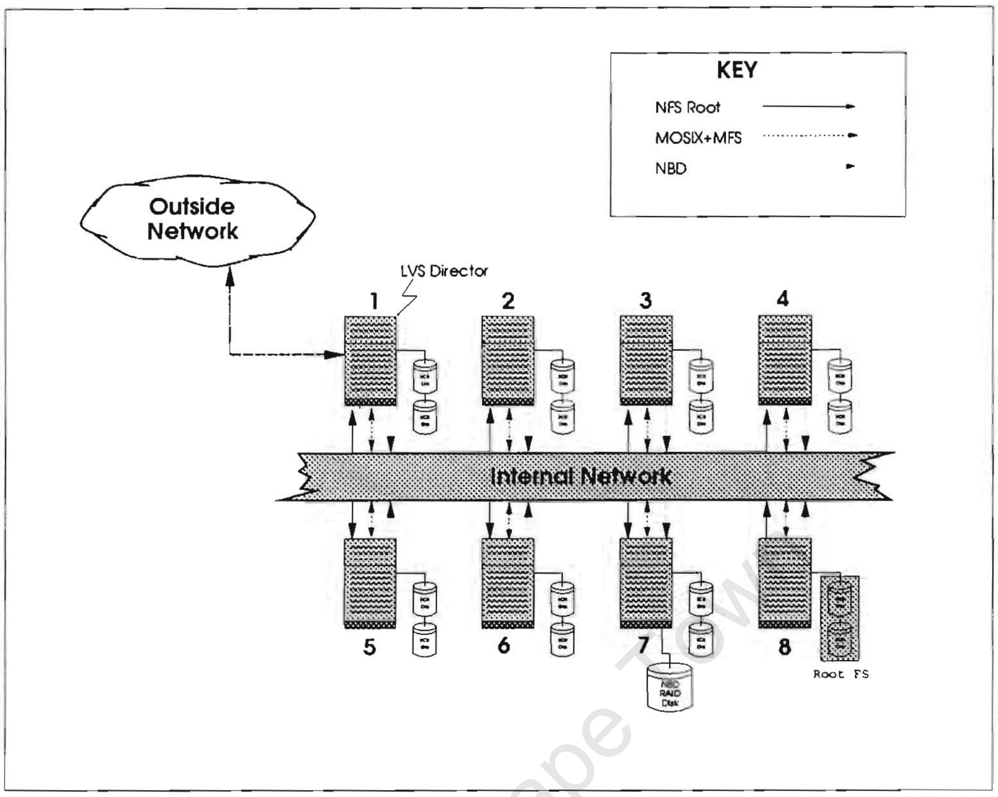


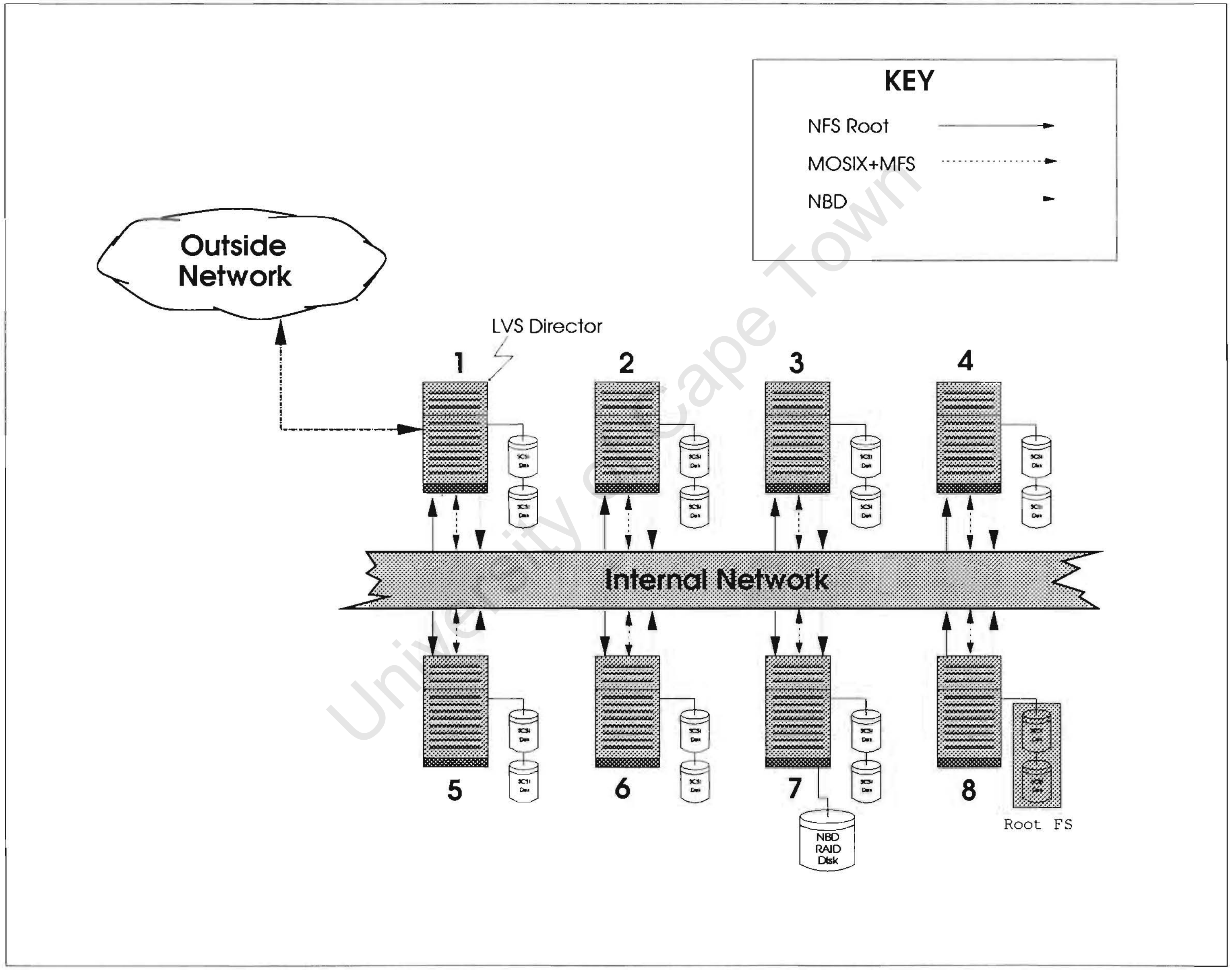
Figure 4.1: General Layout of the Gollach Cluster

Therefore our disks are connected to the one controller as `device ID 0` and `device ID 1`.

Just merely connecting them is barely enough, so we set up the drives as part of striped RAID arrays [49]. Therefore each pair of drives is set up as a RAID 0 array within that node machine. This action is motivated by software RAID (sRAID) documentation which mentions that there is some level of parallelism to be obtained from such a combination.

These RAID arrays are going to be initialised at system start-up. This is especially true for the NFS file server. This machine has to bootstrap its system from a RAID array which carries all the cluster's software. Therefore it is necessary that the kernel must initialise the array. Thus the RAID arrays are set as persistent.

The resulting configuration file (`/etc/raidtab`) for these hard disks is shown below :


KEY
NFS Root
MOSIX+MFS
NBD
Outside Network
LVS Director
1
2
3
4
Internal Network
5
6
7
8
SCSI Disk
NBD RAID Disk
Root FS

```
raiddev  /dev/md0
         raid-level              0
         nr-raid-disks           2
         persistent-superblock   1
         chunk-size              32
         device                  /dev/sda3
         raid-disk               0
         device                  /dev/sdb3
         raid-disk               1
```

4.3 Details of LVS Implementation

To give an overall unified log-in view for the users, required providing a virtual server. The creation of a virtual server requires the use of a VIP. This VIP is the IP address of the cluster server. This IP is also the IP of the machine selected as the director. The director has two network cards, one which communicates with the outside network. The other interface deals with the internal network of which the other machines in the network are attached.

The real servers are on an internal network and thus are inaccessible from the outside network. This is where LVS-NAT comes into play. Although LVS-NAT has severe limitations, it is the simplest setup to implement [9]. As mentioned before, the translations required can become computationally intensive until they overwhelm the director machine. Thus bringing in diminishing returns. In this cluster though, the number of machines is very small and thus the above is not an issue. Another consideration to note is that we are not running network intensive operations like web servers or email servers. Most of the operations are CPU or I/O intensive and thus they do not require very much of the outside network to be used continuously.

Thus our virtual server has 8 machines with one acting as a director. The primary service supplied by the virtual server is SSH. This is all that is necessary for our purposes since SSH allows the use of a user shell and also tunnels X services via the SSH port 22.

4.3.1 Kernel Configuration

The virtual server first requires the implementation of a patch on the kernel. After the patch then the following settings may be selected in the kernel configuration:

```
Code maturity level options --->
         [*] Prompt for development and/or incomplete code/drivers

     Networking options --->
         [*] Network packet filtering (replaces ipchains)
         [ ]   Network packet filtering debugging
         ...
           IP: Netfilter Configuration  --->
           IP: Virtual Server Configuration  --->
             <M> virtual server support (EXPERIMENTAL)
             [*]   IP virtual server debugging
             (12)   IPVS connection table size (the Nth power of 2)
             --- IPVS scheduler
             <M>   round-robin scheduling
             <M>   weighted round-robin scheduling
             <M>   least-connection scheduling scheduling
             <M>   weighted least-connection scheduling
             <M>   locality-based least-connection scheduling
             <M>   locality-based least-connection with replication scheduling
             <M>   destination hashing scheduling
             <M>   source hashing scheduling
             --- IPVS application helper
             <M>   FTP protocol helper
```

4.3.2 Configuration Files

With the kernel configuration in place, we continued to create two script files that would initialise the LVS. These files are called `lvs-director-start` and `lvs-real-start`. The file `lvs-director-start` intialises the director machine, while `lvs-real-start` is there to initialise the real servers.

In these files it will be seen that the primary service supplied by our LVS is SSH. The scheduling algorithm in use is *Weighted Least Connections* scheduling. All the weights on our servers are the same except for **Gollach-1** which has a weight of 2. This decision is justified by the fact that **Gollach-1** is a two processor SMP.

The complete listings of these files are found in Appendix A.

4.4 MOSIX

Although we have distribution on a client level , the granularity is still too high. At the client level we can end up with a user starting up a multitude of processes and thus bogging down one machine while the rest of the cluster might be free. To counter this situation we use MOSIX. MOSIX will then redistribute processes to provide transparent load balancing. This allows the other machines in the cluster to take care of the excessive load imposed by a user. Thus giving us load balancing on a process level, which is much finer granularity compared to LVS.

4.4.1 Kernel Configuration

The installation of MOSIX requires patching of the kernel and selecting the required parameters for the particular cluster. In our case we have a very simple and straightforward cluster on a simple network and thus all those options associated with complex networking topology are left off.

What we did require though was the MFS with DFSA. This is an option that allows MOSIX to consider the location of data during process redistribution. Our MOSIX kernel options are shown below:

```
MOSIX --->

  [*] MOSIX process migration support
  [ ] Support clusters with a complex network topology
  [ ] MOSIX Kernel Debugger
  [*] MOSIX Kernel Diagnostics
  [*] Stricter security on MOSIX ports
```

```
(1) Level of process-identity disclosure (0-3)
[*] Create the kernel with a "-mosix" extension
[*] Direct File-System Access
[*] MOSIX File-System
[ ] Poll/Select exceptions on pipes
```

4.4.2 Configuration Files

A further step in setting up MOSIX was the creation of the configuration file associated with the whole cluster. In `/etc/mtab` we have the following:

```
1       192.168.0.1    8
```

This means that `192.168.0.1` is the first machine IP in the cluster and there are 8 machines in total with consecutive IP addresses i.e `192.168.0.1` to `192.168.0.8`.

An alternative form for this file would have been:

```
1       192.168.0.1    1
2       192.168.0.2    2
3       192.168.0.2    3
4       192.168.0.2    4
5       192.168.0.2    5
6       192.168.0.2    6
7       192.168.0.2    7
8       192.168.0.2    8
```

In our case we had a very simple structure for the IP addresses in our cluster thus we had the opportunity to use the more abridged version above.

Also to go along with `/etc/mtab` is the file `mospe`. The only data contained in this file is a number corresponding to the particular node's MOSIX reference.

For example the machine `192.168.0.1` has the following contained in the file `mospe`:

```
1
```

While the machine 192.168.0.2 has the following:

```
2
```

A similar pattern continues with all the other machines in the cluster. The files mospe and mtab are found on every node in the cluster. The file mtab is replicated on every node while mospe is unique to every machine.

MOSIX redistributes all services that slow down a machine when load levels are high, but there are services which have to remain on the particular machine on which they run. These are system services like system initialisation, kernel update command and system shutdown. Thus they have to be marked to prevent their migration and unpredictable system operation.

To mark these processes off we use the command mosrun. Without any options, the command mosrun executes the process on its home node . Therefore in the files inittab we mark off the operations. An example is shown below:

```
.
.
#
id:3:initdefault:

# System initialization.
si::sysinit:/bin/mosrun -h /etc/rc.d/rc.sysinit

l0:0:wait:/bin/mosrun -h /etc/rc.d/rc 0
l1:1:wait:/bin/mosrun -h /etc/rc.d/rc 1
l2:2:wait:/bin/mosrun -h /etc/rc.d/rc 2
.
.
```

This procedure will lock in those necessary server processes.

4.5 Operating System

For the operating system we needed to keep our administration work at the bare minimum. To achieve this required having a means of maintaining a unified file system image across the entire cluster. As such there are two main ways to achieve this.

1. Replicate the file system across the cluster

2. Have all the machines in the cluster access a single image

4.5.1 File System Replication

Replicating the file system has problems where certain files are intended to be unique. The setup that can deal with such intricacies can end up being more complex than what we are prepared to deal with. For this we would have to consider a replicating file system that is enabled over networks, e.g Intermezzo or DRBD (Distributed Replicated Block Device).

4.5.2 Root NFS

We took the alternative route of putting a root file system that is accessible via NFS as our method [40][42]. This choice is intuitively inviting due to the apparent simplicity behind the idea. All we need is one machine with a root file system and the rest of the machines access that file system.

Having a single system ensures 100% data consistency and total uniformity across the whole cluster. As a further bonus, the system administrator's work is reduced to only dealing with only one machine. All setup and configuration is done on that one machine.

This setup also has useful spin-offs for the users. Since the home directories will be consistent across the cluster, then it will complement the virtual server image very well.

4.5.3 Conceptual Problem

Unfortunately, having a single file system presents a counter-variable to our cluster multiprocessor speed increase. Since all the machines constantly collect their data from one source, there is bound to be a speed descaling with the number of machines accessing the server simultaneously.

The suspicion though is that most of the work to be done will not access the disks continuously thus we would not have to worry about that factor very much.

4.5.4 Kernel

To be able to use this shared operating system in Linux required enabling the `NFS root` feature in the kernel configuration. Also necessary for this is `kernel level IP configuration`, since the kernel will have to know its IP address and the IP of the root server in order to access the file system during the boot-up process. We set the kernel to acquire the IP information via Bootp. The choices were ARP, Bootp and DHCP. DHCP should have been the automatic choice because it is the newest and therefore has the most features, but we were using Bootp as the IP configuration in GRUB (This is the only option in GRUB). Thus we selected this to maintain consistency in our setup.

4.5.5 Alternative Attempts

As a first attempt, the root over NFS system was setup with normal NFS which comes standard with RedHat 7.3. This layout had the client machines setup in a directory called `client-root`. This was done for experimental reasons. This was in contrast with the ideal of having the clients access the true root directory. The kernel image for the clients is transferred via **TFTP**, thus it is found in the `/tftpboot` directory.

With this initial setup we were quick to find out that implementing a consistent image for all machines in the cluster was a task with little hope for resolution. A particular problem that arises is how to keep shared files/directories shared while keeping unique files unique. This is what we will term the *share/unique* problem in this document.

One method attempted was to have a small start-up image which carries the necessary start-up files. This approach is similar to that used when booting with an *Initial Ramdisk*.

Separating Files/Directories

If this setup is approached without dealing with the *share/unique* problem, each machine ends up with a separate root directory. Thus all we end up with is multiple file systems on a single disk.

With the above method we can adapt it to try and deal with the *share/unique* problem by doing a switch-over after boot-up. In this case the real directories are mounted over the temporary initial ones after start-up, but before the full initialisation is complete.

Using Links

The above approach works to a certain extent, except that on a file level it becomes especially complex. Some attempts use soft/hard links to keep common files common. Unfortunately for a directory like `/etc` there are many more shared files than unique files. Thus we end up doing a lot of linking and such becomes difficult to keep track of.

On the NFS host server, we set up `/etc/exports` to allow the machines in the cluster to access their root directories and also to access the shared directories. This is shown below:

```
/client_root  192.168.0.2(rw,no_root_squash)
/client_root  192.168.0.3(rw,no_root_squash)
/client_root  192.168.0.4(rw,no_root_squash)
/client_root  192.168.0.5(rw,no_root_squash)
/client_root  192.168.0.6(rw,no_root_squash)
/client_root  192.168.0.7(rw,no_root_squash)
/client_root  192.168.0.8(rw,no_root_squash)
```

4.5.6 ClusterNFS

A working and lasting solution to our woes for a consistent single file-system image across the whole cluster were solved with the help of **ClusterNFS**. ClusterNFS is an extension to the UFS daemon which is specially suited for clustering situations. The feature which makes it so suitable is *name-translation*. This feature solves the *share/unique* problem mentioned in previous attempts to create a shared root file-system.

When files which are unique are tagged then they are visible only to the machine associated with the tag.

For our setup we then had to figure out which files were unique for the machines in our cluster. These are the files that we tag. The rest are exported as they are and thus remain shared.

To get ClusterNFS package working involved acquiring the package as an RPM (Redhat Package Manager) and installing it as a usual package. By default the name translation is not enabled when the daemon starts up during the boot-up process. To get this feature up and running needs editing of the file `/etc/rc.d/init.d/nfs`. The lines with

```
.
.
        echo
fi
echo -n $''Starting NFS mountd:  ''
daemon rpc.mountd $RPCMOUNTDOPTS
echo
echo -n $''Starting NFS daemon:  ''
daemon rpc.nfsd $RPCNFSDCOUNT
echo
touch /var/lock/subsys/nfs
.
.
```

are modified to the following :

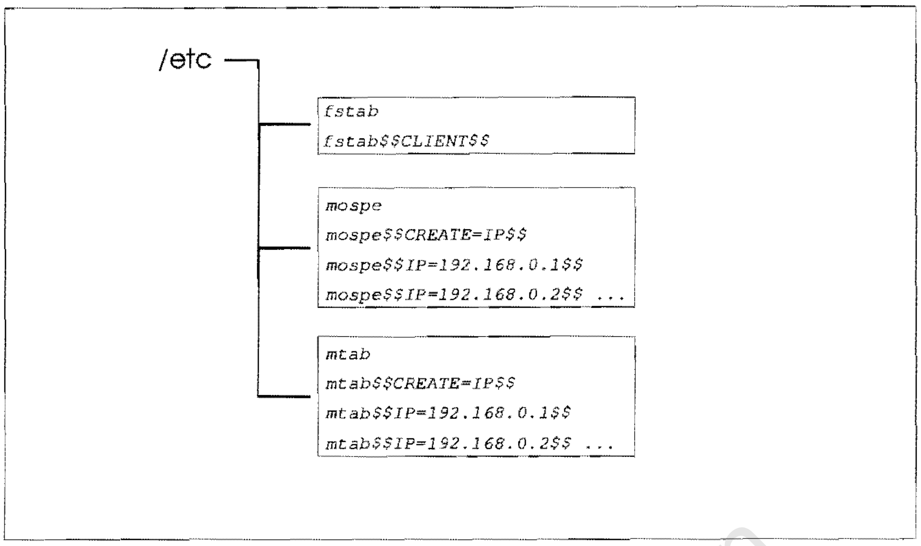


Figure 4.2: Files Modified in `/etc`

```
.
.
        echo
fi
echo -n $''Starting NFS mountd: ''
daemon rpc.mountd --translate-names $RPCMOUNTDOPTS
echo
echo -n $''Starting NFS daemon: ''
daemon rpc.nfsd --translate-names $RPCNFSDCOUNT
echo
touch /var/lock/subsys/nfs
.
.
```

Then from there tagged files will be translated by the system.

With this system all set up, we proceeded to tag the files which were intended to be unique. At the highest level we have the directories. The unique ones are : `/dev`, `/var`, `/tmp` and `/proc`.

Thus we make a copy that is unique for every machine. Next to be discussed is the `/etc` directory. This directory has files that determine the behaviour of the particular machines in the cluster. Thus we can proceed to mark off the files which are unique. These are shown in Figure 4.2.

Also in `/etc/sysconfig` directory there was some work to be done with the networking options. Since each machine has unique network settings, we therefore have to separate out the directories and files concerned with networking.

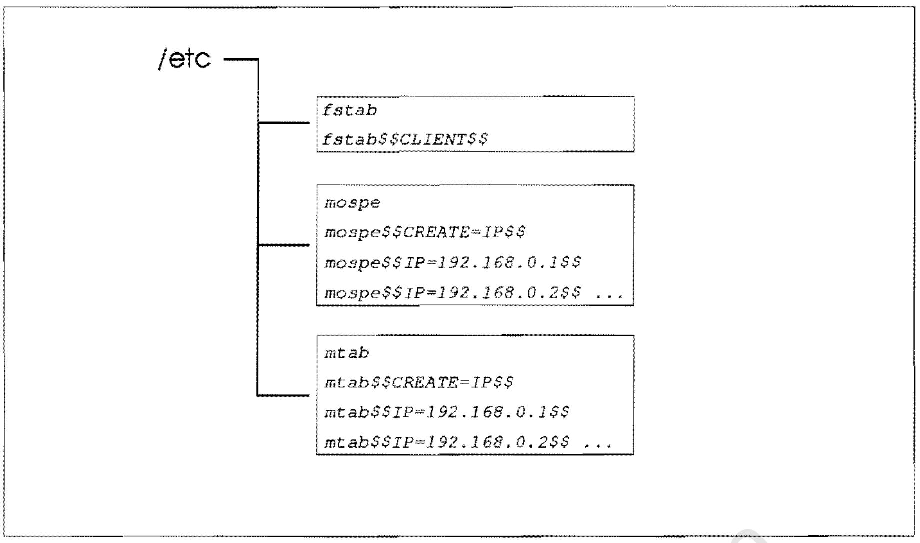


Figure 4.3: Files Modified in `/etc/sysconfig`

Once all the associated files have been marked and done, then the setup is basically complete. All of this work is done on the root file server. The last item is to allow access by the machines in the cluster to the root directory. All this requires is a few lines in `/etc/exports` like

```
/     192.168.0.1(rw, no_root_squash)
/     192.168.0.2(rw, no_root_squash)
/     192.168.0.3(rw, no_root_squash)
/     192.168.0.4(rw, no_root_squash)
/     192.168.0.5(rw, no_root_squash)
/     192.168.0.6(rw, no_root_squash)
/     192.168.0.7(rw, no_root_squash)
```

If all other things are setup then the machines should be able to boot as if they were quite diskful.

A change was also made to `/etc/fstab`, which on the server reads

```
/dev/md0     /            ext3     defaults                 1 1
none         /dev/pts     devpts   gid=5,mode=620           0 0
none         /proc        proc     defaults                 0 0
none         /dev/shm     tmpfs    defaults                 0 0
/dev/sda2    swap         swap     defaults                 0 0
/dev/sdb2    swap         swap     defaults                 0 0
/dev/fd0     /mnt/floppy  auto     noauto,owner,kudzu        0 0
none         /mfs         mfs      dfsa=1                   0 0
```

And on the clients is

```
192.168.0.8:/   /            ext3     defaults                 1 1
none            /dev/pts     devpts   gid=5,mode=620           0 0
none            /proc        proc     defaults                 0 0
none            /dev/shm     tmpfs    defaults                 0 0
/dev/sda2       swap         swap     defaults                 0 0
/dev/sdb2       swap         swap     defaults                 0 0
/dev/fd0        /mnt/floppy  auto     noauto,owner,kudzu        0 0
none            /mfs         mfs      dfsa=1                   0 0
```

In the above the significant lines are written in larger print. The top line tells the OS that the root is a RAID device. The top line in the second case above is to let the client know that the root file-system is on the NFS server `192.168.0.8`.

ClusterNFS is completely transparent during operation and thus the view on the user machines should be that of a regular file system. Our final solution to the *share/unique* problem is shown in Figure 4.4.

A lot of files have been left out of the figure for the sake of brevity.

With the help of ClusterNFS, we are able to separate out the parts which are not shared on the master disk. Thus with this system, we only have “one computer” to administer. If anything has to happen, it happens on that one machine. Any changes which need to be made are made on that one computer. Fortunately, since the computers share a file-system, the converse is also true and any changes made on other nodes is system-wide.

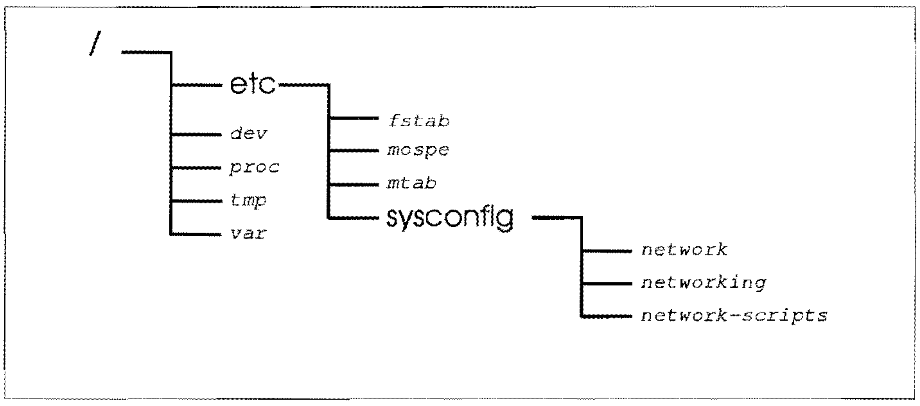


Figure 4.4: The **Share/Unique** Solution

The machine with the file-system has a lot of tags and thus ends up looking rather a mess. Thus thanks to NFS we can use a "clean" machine for most of the administration. Thus we have a Single System Image and a conventional view for most of the installation.

4.5.7 MFS

In the MOSIX scheme of things, we wanted to take as much advantage of the MFS as possible. A prominent feature in MFS is DFSA. This feature makes MOSIX migrate processes to their I/O home nodes rather than migrate the I/O to the processes.

To take advantage of DFSA, we moved the directories `/opt` and `/home` to directory names `/real_root/opt` and `/real_root/home`. We then created symbolic links `/opt` and `/home` and pointed these to their real namesakes using the MFS directory. Therefore `/home` is a symbolic link to the directory `/mfs/8/real_root/home`. There is a similar link for `/opt`.

Thus our `/home` and `/opt` directories are now directly sourced from their home node.

4.6 Diskless Booting

4.6.1 GRUB

GRUB was our system of choice for the boot-up management [28]. Conventional diskless booting normally makes use of PXEs which usually includes a boot-ROM installed on the ethernet card together with a packet driver to get the device talking on the network.

Since our machines do have disks, we discarded the inflexibility of boot-ROMs and opted to use the installed disks instead.

GRUB as supplied with the RedHat system does not support booting off the network. To get this to work meant obtaining the source code for GNU-GRUB and compiling it with `--network` option.

This together with the packet driver for our interfaces produces a GRUB system that can talk to the network.

Initially GRUB has to be used off a boot floppy diskette. This is created with the commands :

```
#   dd if=stage1 of=/dev/fd0 bs=512 count=1

#   dd if=stage2 of=/dev/fd0 bs=512 seek=1
```

On boot-up GRUB gives a command prompt. At this prompt we give the following commands to bring up a node in our virtual server.

```
grub>  bootp
grub>  root (nd)
grub>  kernel /vmlinuz-2.4.18 root=/dev/nfs ip=bootp rw
grub>  boot
```

This detects the IP settings via the Bootp protocol, lets the system know the root device is the network, gets the kernel over the network via TFTP and the last command tells us to boot up the system. If the DHCP system, the kernel and the root NFS system are all setup correctly the machine comes up as any ordinary Linux computer.

With our computer up and running we took the opportunity to install the GRUB system on a local hard disk drive in an effort to automate the boot-up process.

For the GRUB system we used the small 10 MB partition for the GRUB directory and copied all the necessary GRUB software onto it. With this done, running the command grub starts up an interactive command system for GRUB maintenances. On this prompt we issued the commands

```
grub>  root (sd0,0)
grub>  find /boot/grub/stage1
grub>  setup (sd0)
```

and the GRUB software is then installed on the MBR (Master Boot Record) of the hard drive thus the system can now boot-up unaided.

The last item was to throw our boot-up commands into the file grub.conf as a menu item. Thus grub.conf for the machine **Gollach-1** has the following text

```
default  = 1
timeout  = 5
title    Red Hat Linux (Network)
         bootp
         root (nd)
         kernel /vmlinuz-2.4.18 root=/dev/nfs \
         ip=::::eth0:bootp rw
```

The final line in the file grub.conf above differs depending on the role it plays on the particular machine. In the case above, the term ip=::::eth0:bootp tells the kernel that it must configure the IP settings for the interface eth0 via bootp. This is a requirement since the machine has two ethernet interfaces and the computer may fail to boot-up because the kernel attempts to configure the wrong interface.

On the machines **Gollach-2** to **Gollach-7** the line is somewhat different since each one of these machines only has one interface. Thus the line reads as below:

```
.
.
    kernel /vmlinuz-2.4.18 root=/dev/nfs ip=bootp rw
```

In this case we can see that with the term ip=... we only needed to specify bootp as input.

For the NFS root server, i.e. **Gollach-8** the case is completely different. Since This machine boots up from a hard disk drive, then it specifies its root as such. Also since the computer is the one which issues IP setting to the other machines in the cluster, then it will not make use of any kernel IP configuration. The grub.conf file for this machine is :

```
default  = 0
timeout  = 10

#image    0
title    Mosix 1.6.0 ( 2.4.18 )
         root (sd0,0)
         kernel /vmlinuz-2.4.18 root=/dev/md0 \
         md=0,/dev/sda3,/dev/sdb3 ro

#image    1
title    Red Hat Linux (2.4.18-3smp)
         root (sd0,0)
         kernel /vmlinuz-2.4.18-3smp root=/dev/md0 \
         md=0,/dev/sda3,/dev/sdb3 ro
         initrd /initrd-2.4.18-3smp.img
```

4.6.2 DHCP

DHCP takes care of most of our boot-strapping requirements from the server side of things in our diskless configuration.

The DHCPD (DHCP daemon) is supplied with the RedHat system. We enable DHCP in the system configuration. Once this is done then we setup the file /etc/dhcpd.conf file adding in the entries for our machines as shown below.

With GRUB and the kernel working in Bootp mode, we enable Bootp compatibility using

```
.
.
enable bootp;
.
```

Then we add in an entry for the machines in the cluster using the following lines.

```
.
.
host g1 {
hardware ethernet 00:a0:c9:ec:04:90;
fixed-address 192.168.0.1;
option root-path "/";
}
.
```

In this case we have shown only the configuration for one server. The rest take up a similar setting, with appropriate changes to the `host` setting and also to `hardware ethernet` and `fixed-address`. This gives each machine a static IP address and the name server settings for the location of the kernel. Thus enabling GRUB to bootstrap the machine, the kernel to set IP settings and also the location of the root NFS system.

4.7 NBD

NBD is a client server system that will allow us to network raw storage devices. We implemented the NBD system to show that storage devices in a network can be unified to give an image of a single large conventional storage device. To achieve this, we also took advantage of Linux Software RAID.

4.7.1 Storage Sources

Storage devices were exported from every machine except the RAID host and the NFS server (i.e. **Gollach-7** and **Gollach-8** respectively). With the former, the reasoning was that the hard disk drives were already available locally thus there was no need to network them. With the latter, all the drives were being used as the root file system on the clusterNFS server.

Once exported and then made available on the RAID host, the network drives were configured in two separate scenarios to test two personalities. The RAID levels used were **RAID-0** and **RAID-5**.

4.7.2 Kernel Configuration

The NBD system is enabled by selecting its appropriate section under the `Block Devices` heading. This is shown next:

```
Block Devices --->

    <*> Normal PC floppy disk support
    < > XT hard disk support
    < > Compaq SMART2 support
    < > Compaq Smart Array 5xxx support
    < > Mylex DAC960/DAC1100 PCI RAID Controller support
    <*> Loopback device support
    <M> Network block device support
    < > RAM disk support
```

This option will allow the NBD device driver to be loaded into the kernel as a module. This device driver is only required on the machine which will hold the NBD clients.

Further to the kernel option, it is also necessary to create the necessary devices in the `/dev` directory. This was done by calling the command

```
  #   mknod /dev/nda b 43 0
```

Devices in `/dev` have three characteristics, which are:

- type
- major number
- minor number

`mknod` has to be run for all the clients that are required, thus it requires device names from `nd0` to `nd5` and minor numbers from `0` to `5`. The major for these devices is `43`. The `b` in the command call signifies a block special device (buffered).

4.7.3 RAID Configuration

To get a combined RAID drive working required setting up the system in `/etc/raidtab`. The layout was such that each machine exports a single drive. Therefore we setup `/dev/md0` as a stripe of two drives on each of the machines. The configuration used has been shown above in the hardware implementation section.

This would utilise some parallelism and allow us to maximise on the controller's bandwidth. `/etc/raidtab` is read by the system scripts during initialisation, this means it would start up automatically just after kernel boot.

Having the networked RAID drive start up automatically as one with local devices has problems. This is due to the fact that several other things are now involved i.e. :

- nbd.o the kernel module for clients.
- nbd-server
- nbd-client

Thus we decided to conduct this initialisation in a separate script file. To separate it from system initialisation, we put the setup in a file `/etc/raidtab2`. The contents of this were as follows :

```
raiddev  /dev/md1
         raid-level             0
         nr-raid-disks          7
         persistent-superblock  0
         chunk-size             32

         device                 /dev/nd0
         raid-disk              0
         device                 /dev/nd1
         raid-disk              1
         device                 /dev/nd2
         raid-disk              2
         device                 /dev/nd3
         raid-disk              3
         device                 /dev/nd4
         raid-disk              4
         device                 /dev/nd5
         raid-disk              5
         device                 /dev/md0
         raid-disk              0
.
.
```

In the same file we set-up `/dev/md2` as follows

```
.
.
raiddev  /dev/md1
         raid-level              5
         nr-raid-disks           6
         nr-spare-disks          1
         parity-algorithm        left-asymmetric
         persistent-superblock   0
         chunk-size              32

         device                  /dev/nd0
         raid-disk               0
         device                  /dev/nd1
         raid-disk               1
         device                  /dev/nd2
         raid-disk               2
         device                  /dev/nd3
         raid-disk               3
         device                  /dev/nd4
         raid-disk               4
         device                  /dev/nd5
         raid-disk               5
         device                  /dev/md0
         spare-disk              0
```

The two sets of RAID devices /dev/md1 and /dev/md2 were setup for the two personalities i.e RAID-0 and RAID-5 which are to be run in different contexts.

4.7.4 Running the NBD System

To run an NBD server we issue the command :

```
#   nbd-server 1700 /dev/md0
```

This starts a server on the machine and exports the device /dev/md0. Then on our client machine we issue :

```
#   modprobe nbd
#   nbd-client g1 1700 /dev/nd0
```

This set of commands inserts the NBD kernel module, starts a client and attaches it to /dev/nd0. We run the nbd-client command consecutively for devices up to /dev/nd5, which are our components for the network RAID.

The RAID device is then started up with the command :

```
#   mkraid -c /etc/raidtab2 /dev/md1
```

At this point the RAID drive is ready for a file format and is to be used as any other storage device.

The above collection of commands was captured and put in a script file to automate the process during system start-up. The listings for the server and the client configurations is shown in Appendix A.

The line in the listing with /dev/md1 is changed to /dev/md2 when testing the RAID-5 configuration.

We initialised our network distributed drive with EXT3 (Third Extended File-system). Afterwards it was mounted with the mount command like any normal storage device.

4.8 System Running

The trace of the Gollach cluster during initialisation is shown in Figure 4.5. This shows how the main components building up this server machine get initialised.

4.9 Summary of Implementation

In summary, this chapter has dealt with the building of a cluster server. We have discussed how GRUB, DHCP and TFTP combinations were used in order to build up a bootstrapping subsystem for the cluster. We mentioned how ClusterNFS was our system of choice for combining the root file system. We have described the implementation of LVS as a means

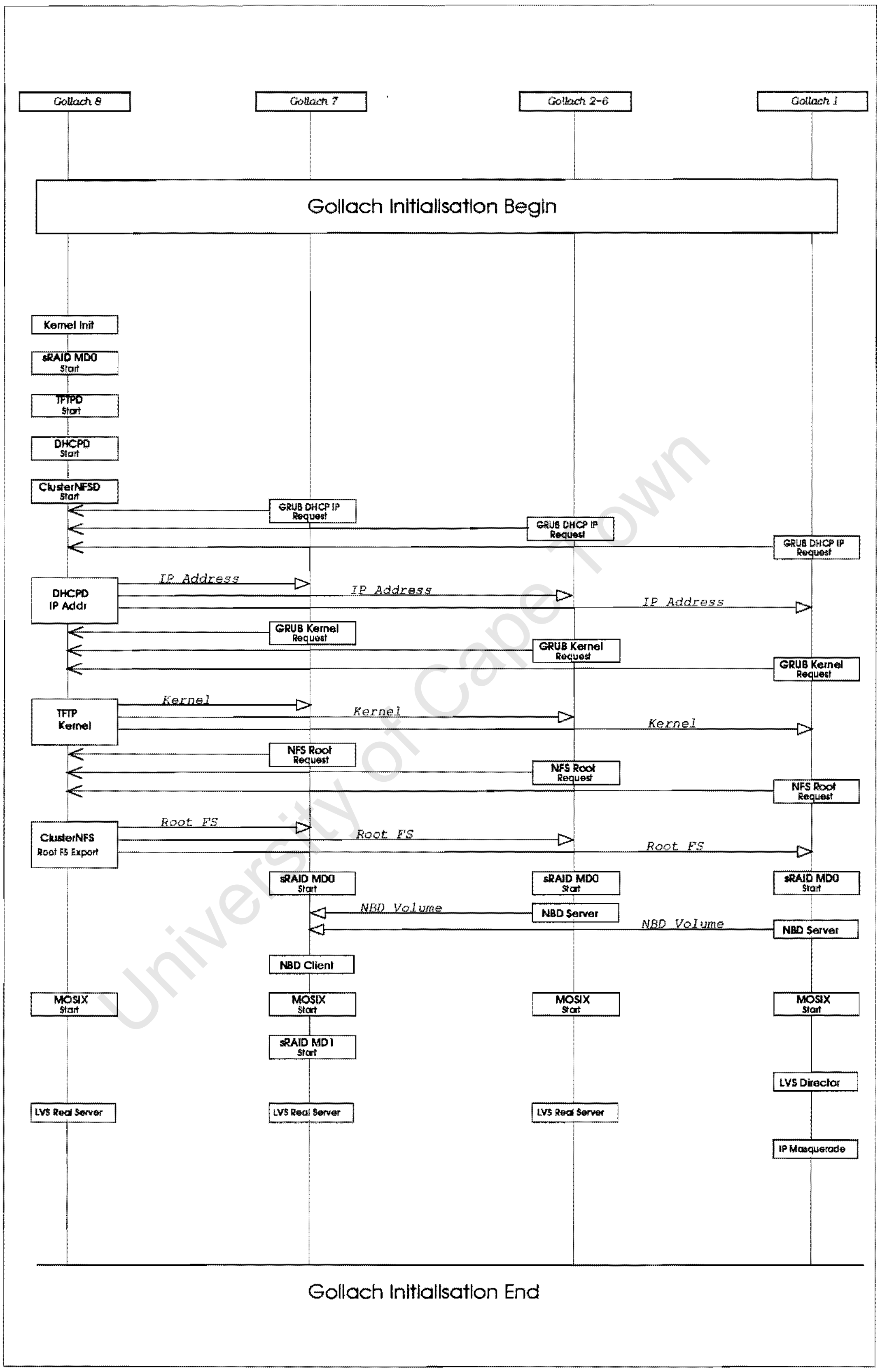


Figure 4.5: Gollach System Startup Trace

to access the cluster server and give the user a single image of the system through SSH. We saw how NBD was the major building block for combining the mass storage in the server. To fully take advantage of what NBD had to offer it was mated with sRAID. We mentioned how MOSIX was implemented in order to provide a finer grain of parallelism to our server use.

Chapter 5

Tests and results

This chapter details the testing process of the cluster after being built up. It states the steps taken, the results and the findings that were made during the testing phase.

5.1 Overview of Tests

On building the system, we have put together the abstracted parts . Thus we have realised our vision in terms of theoretical implementation. What is left is to actually test these things and determine their work-ability.

We have tried to test system properties from a high level, down to the ground level. On knowing how the ground level components behave, then we can deduce explanations for the behaviour of higher level components of the system.

The tests conducted on this cluster are in several classes depending on their realisation of the physical world. We ran tests on the hardware and also on the software involved in this cluster. The main hardware tested is the storage system in various configurations. Hardware testing also included testing of the network and issues surrounding it. Software tests included running the programs expected to run on this cluster and seeing whether they operate correctly. The high level tests conducted looked at how well the cluster holds up in a work environment. This means long term stability and performance under load.

For the testing, we did three kinds of tests. These were :

1. *High level tests* - these were conducted to find out whether the cluster does what we intended it to do in a qualitative sense.

2. *Reliability tests* - these were meant to give us some idea of how our cluster fares as an everyday utility. Therefore we see how it deals with undesirable operation.

3. *Low level tests* - involve the hard drive storage, NFS speed, network efficiency, NBD performance and MOSIX efficiency. These gave us raw figures for performance which we used in formulating some of our explanations.

5.2 Cluster Software High Level Tests

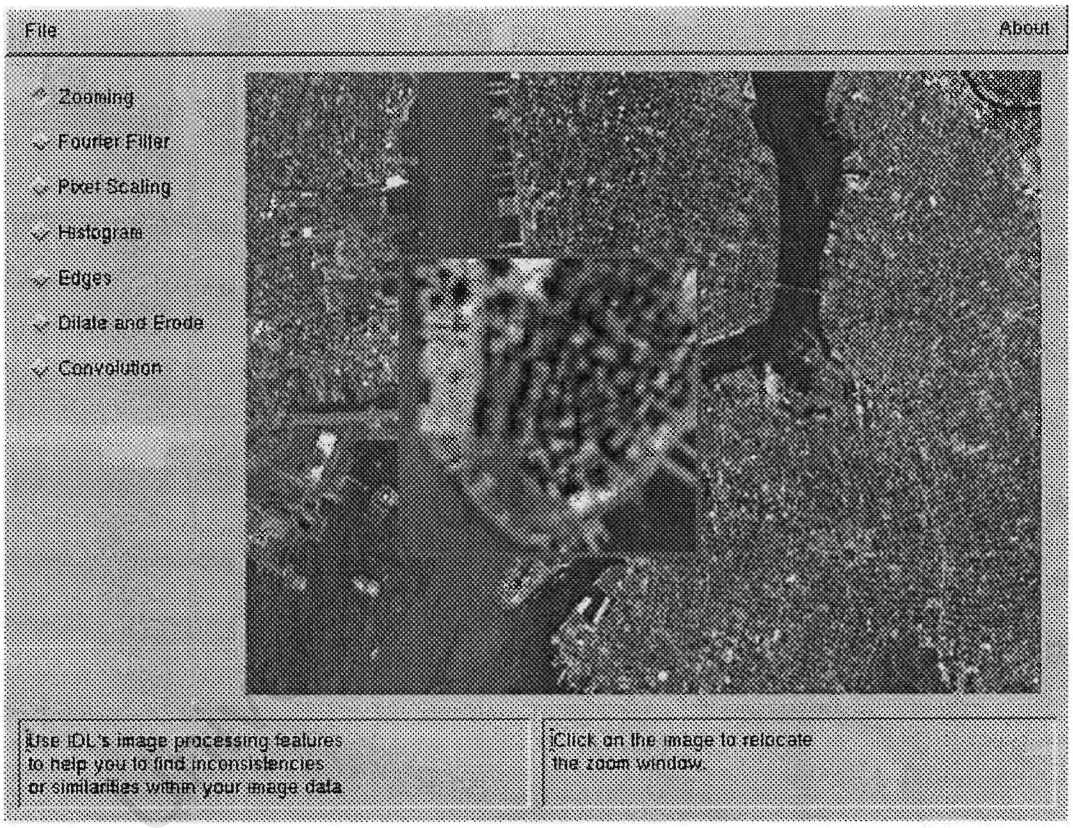
With these high level tests, we intend to show that the cluster does what is expected of it. In short this cluster is intended to behave as a conventional single Linux machine in terms of usage.

5.2.1 Software Running

We tested the running of some conventional software on this cluster to ensure that it works as required. The programs tested include Matlab, IDL, Galeon, PVM. The running of these programs was found to be identical to that of a conventional machine. A screenshot of IDL running an image processing demonstration is shown in Figure 5.1

5.2.2 ClusterNFS

The only items which have to be looked at are the ClusterNFS specific features. This means dealing with the files which are specific to the different machines in the cluster. Fortunately



Use IDL's image processing features to help you to find inconsistencies or similarities within your image data.

Click on the image to relocate the zoom window.

Figure 5.1: IDL Demo Screenshot

most of these are created during initial installation of the cluster and the administrator virtually never has to fiddle with them once the system is up. ClusterNFS does have a quirk in its semantics for file creation and this has to be an administration watch-point.

If a file exists for both the home machine and a certain remote machine, for example :

```
fstab        fstab$$IP=192.168.0.1$$
```

On 192.168.0.1, editing `fstab` and saving is only meant to modify the file :

```
fstab$$IP=192.168.0.1$$
```

Numerous attempts have shown that the more realistic situation experienced has been to remove `fstab` and recreate `fstab$$192.168.0.1$$`. Thus the main server unwittingly loses a file. This can be a grave disadvantage for the administrator who wants to take a shortcut by editing host specific files on the particular host.

5.2.3 MOSIX

Since we know that MOSIX runs on a fork-and-forget principle in order to take advantage of its parallelism, the testing of MOSIX involves showing that a CPU bound process will migrate to other computers in the cluster.

To show that our cluster was migrating processes, we used **mon** the MOSIX process monitoring program to see our load distribution. To get a decent load for the cluster we compiled a Linux kernel with the `-j` switch. The `-j` switch takes a parameter for the number of make processes that should be spawned. In our case we used 8. Figure 5.2 and 5.3 respectively show the load distribution on the cluster soon after calling the command and then after letting the processes redistribute.

In the Linux kernel source directory we ran the following commands to achieve this :

```
# make clean

# make -j 8 bzImage
```

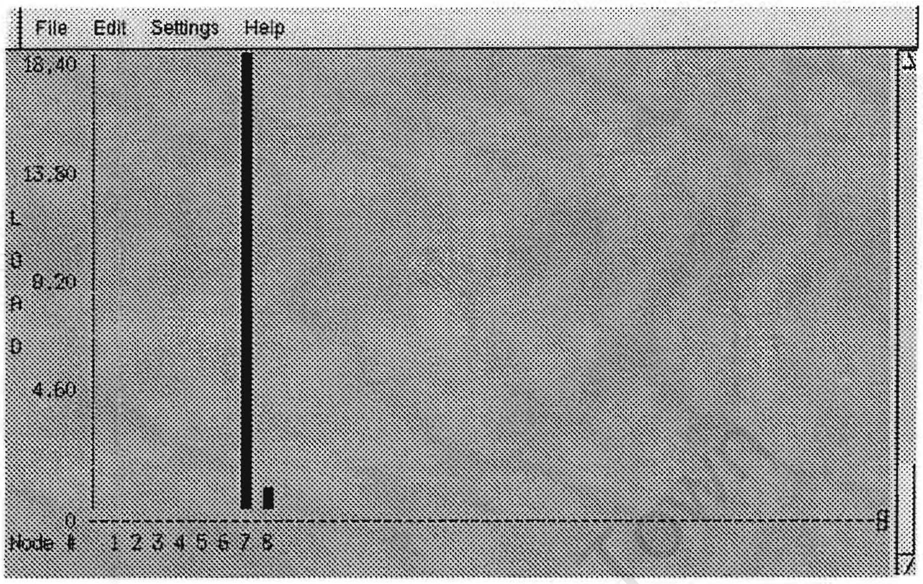


Figure 5.2: Initial Screen of Kernel Compilation

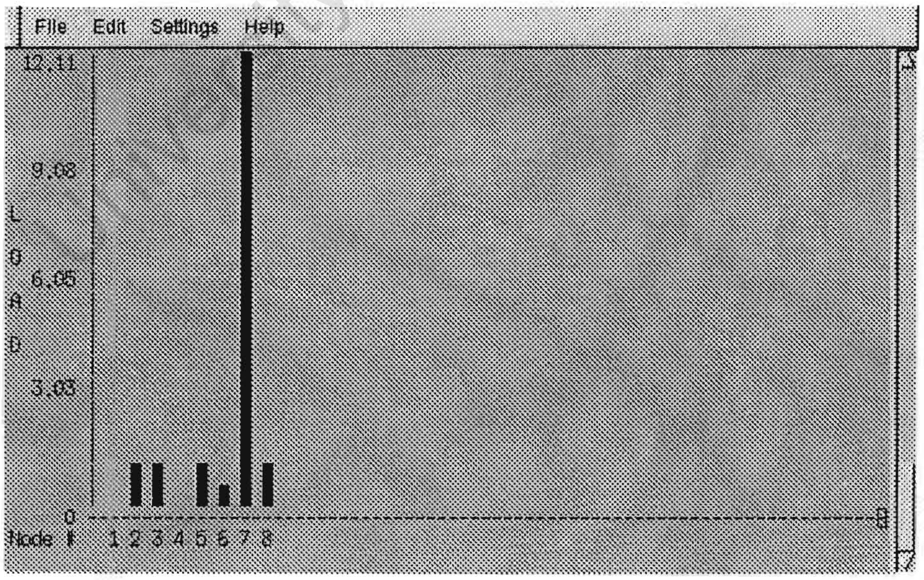


Figure 5.3: Kernel Compilation With Some Migration

In these plots we can observe that the bar on **Gollach-7** has remained tall even after the process migration. This is due to the heavy I/O requirements of this operation. In the plot for the initialisation of the compilation there is a small bar on **Gollach-8**. This is due to the NFS server requests from **Gollach-7** for the files. The result of the entire operation gave us the following time as a result :

16.46 seconds

As a comparison to a stand-alone server, we disabled MOSIX migration by issuing the command:

```
# echo "1" > /proc/mosix/admin/stay
```

This operation locks the processes onto this node. The we re-ran the entire operation and got a result of:

23.28 seconds

Therefore there has been a gain of 6.82 seconds from using MOSIX.

5.2.4 LVS

With the LVS, we wanted to find out how well log-in clients were being distributed on our cluster. To do this required that we log in multiple times and check the user distribution with the LVS command `ipvsadm`. A sample of the output from `ipvsadm` is shown below :

```
[root@gollach root]# ipvsadm
IP Virtual Server version 1.0.4 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  gollach.ee.uct.ac.za:ssh wlc
  -> g8:ssh                       Masq    1      1          0
```

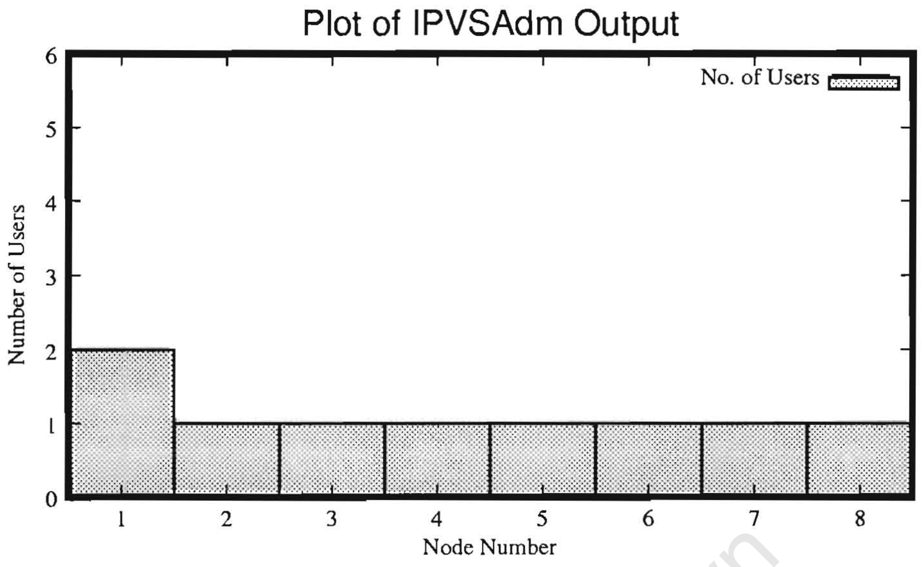


Figure 5.4: A Sample Graphing of LVS Output

```
-> g7:ssh                      Masq    1      1        0
-> g6:ssh                      Masq    1      1        0
-> g5:ssh                      Masq    1      1        0
-> g4:ssh                      Masq    1      1        0
-> g3:ssh                      Masq    1      1        0
-> g2:ssh                      Masq    1      1        0
-> gollach:ssh                 Local   2      2        0
```

The above can be graphed as in Figure 5.4 by taking figures from the **ActiveConn** column.

We logged into the server numerous times and then acquired figures for the changing scenarios. The results are all in the collection of graphs in Figure 5.5. These are plotted for the cases with 15 , 25 , 35 and 45 users respectively.

We can see from most of the graphs that **Gollach-1** generally gets more users of which this is expected from its greater weight setting.

5.3 Reliability

We needed to establish how reliable our server was in the long term. This required testing the server resilience at different levels. Therefore we gave it several tests to determine if it

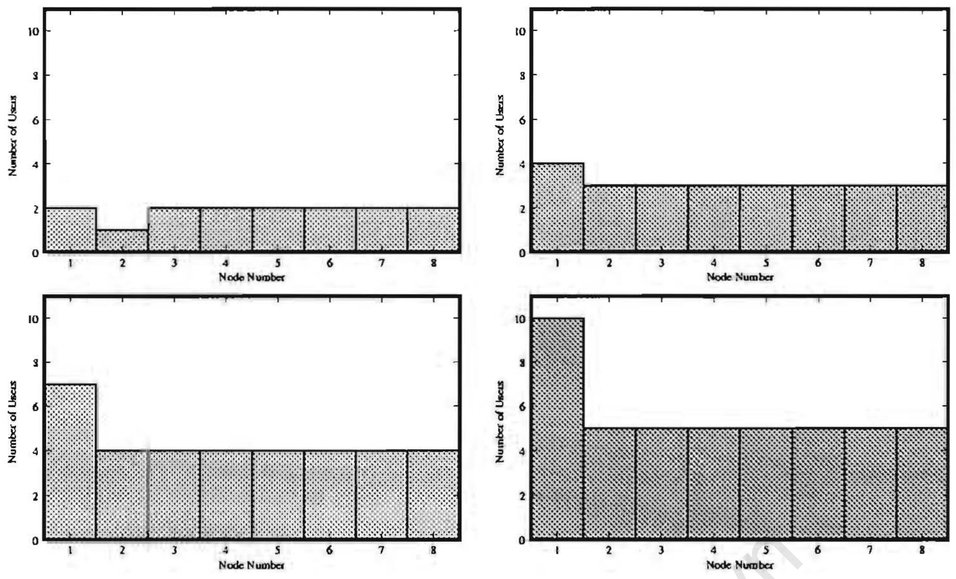


Figure 5.5: LVS Result for Various User Counts

would maintain uniform behaviour.

Firstly we ran start-up test so we could ensure that our server could start up unaided.

After the start-up tests, we ran tests on the critical points of the system to see what the reaction of the server would be to these kinds of failures. The main points we were watching out for were: the LVS, the NFS and the hard drives in the system.

After the above tests, we did a long term test.

5.3.1 Start-up Test

We had two start-up tests to conduct on this cluster server. The first is a "clean" start-up. In this test we shutdown the server correctly so as to find out if it comes up as expected.

In the second case we simply turn off the power while the system is running. Such a situation is typical of a power-cut. This test will show us how our server responds to interrupted operation.

The server comes up as expected in both cases. The second case took a while longer to complete the operation though. The resulting time difference is due to the file system check produced by the "unclean shutdown". Thus in both cases the server comes up correctly, unaided.

5.3.2 Critical Points

In this section we tested the removal of three critical items to the system while it was running. These were

- LVS Director
- NFS server
- Client in the NBD storage system

LVS Director

On removing the LVS Director, the cluster was immediately rendered inaccessible. Since this particular machine is the only one with an interface to the outside network, removing it immediately cuts-off all the other servers. The other machines in the cluster are otherwise not affected and on returning their director machine, all that is needed is to re-login and continue.

NFS server

The NFS server has almost a similar effect to the LVS Director on removal, except milder. On removing this machine all operations halt. The difference in this case is that all the other machines go into a busy-waiting state. Linux gives a message

```
nfs: task 5547 can't get a request slot
```

and on returning the server machine, then it gives a message

```
nfs: server 192.168.0.8 OK
```

From this point everything is back to normal and the NFS client machines continue as though nothing had happened.

NBD clients RAID 0

With the RAID 0 system, removing an NBD client during run-time immediately gives system write errors. The client was removed by issuing a `kill` command directed at the particular `nbd-client` process.

Data corruption has been observed at certain times with this test. This was caused by a write being initiated before the system had realised that something was amiss.

Otherwise restarting the missing device, restarting the RAID array and performing a file system check has been able to cure the system in some cases.

Thus in RAID 0 the entire membership of the RAID array has to be present all the time while the system runs.

RAID 5

It must be noted that the current version of NBD did not fully support RAID 5 (becomes inaccessible after file system creation), therefore we could not perform a test with a file system on it. Thus this test was conducted on a raw device.

An NBD client was removed in a similar way to the RAID 0 case above.

In the case of RAID-5, the result of removing a device was not as critical as in the case of RAID-0. On removal of a device, the system immediately goes into what is termed *resync*. This is where the system rebuilds the missing information onto the spare disk using parity information contained on the remaining devices.

With the RAID-5 system, returning a device did not require an entire restart of the software RAID system. All that was needed was to remove the failed device from the array and then return it. A device was removed and returned with the commands :

```
#   raidhotremove /dev/md0 /dev/nda

#   raidhotadd /dev/md0 /dev/nda
```

Once this was done, then the device was registered as a spare disk and the RAID device continued running.

5.3.3 Long Term Test

In our long term test, we had the server run for several weeks uninterrupted. In that time it was serving out connection requests and running normal Linux applications as required by users. In this case it proved to be just as reliable as a stand-alone Linux server.

In this case all the machines in the cluster have run without major errors (like system lockups or kernel dumps).

5.4 Low Level Tests

At the lowest level we have the hard drive, the SCSI controller, the ethernet and the operating system as our lowest level components.

The first item we need for our tests is the raw I/O speed for devices in our Linux machines. This we can do by timing the dd operation as it reads or writes to the particular device. Before doing these I/O tests, we first had to know the performance of our timing tools (i.e. dd), thus we ran :

```
#   time dd if=/dev/zero of =/dev/null bs=1M count=100000
```

This operation was repeated for data sizes ranging from 10 MB to 1000 MB so that we had a good idea of the total response of this operation. The result of this is shown in Figure 5.6.

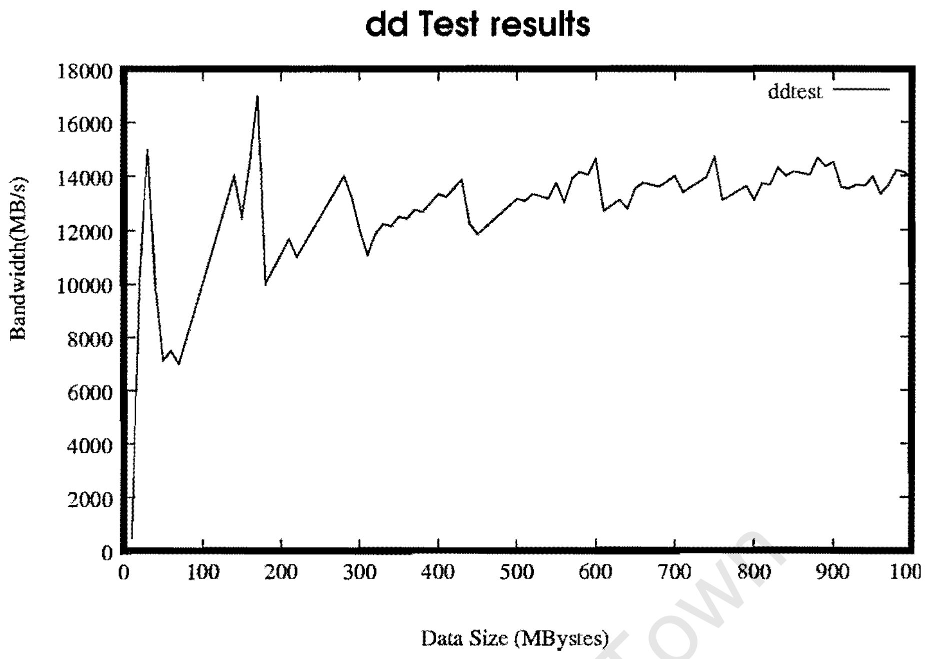


Figure 5.6: Results for the dd Test

This was our equivalent of shorting out the terminals of a voltmeter in order to get its zero reading. From this graph we can see extremely low bandwidths for tests below 150 MB. This is because at such small data sizes our user overhead time is very large compared to the dd reading. Typically the total time for such an operation at 10 MB is 0.010 s and user time is also 0.010 s. Yet for a much larger count like 500 M total time is 0.074 s and user time remains at 0.010 s.

What should be noticed though is that beyond 200 MB our bandwidth does not fall below 10 GB/s. Thus, this is the time for either `/dev/zero` or `/dev/null`. Since trying to get a figure for each one separately is like trying to hear the sound of one hand clapping (i.e. we have no infinite bandwidth sources to test these two separately), we just have to use what we have. Therefore these will have to be our assumed infinite sources for the purposes of our tests.

With the result of how well **dd** works and knowledge of how good our two data pipelines are, we proceeded to test the other hardware devices in various configurations.

At the lowest level we tested the raw ethernet speed and the raw storage speed. The storage was tested in the various configurations mentioned in the *Implementation* chapter.

5.4.1 Network

For our network we wanted to find out the response speeds of our interfaces and our switch.

Test Procedure for Network Tests

To conduct tests using TTCP requires the creation of client and server processes between the two machines under test. The server is created on some remote machine (assume **Gollach-1** for this example, therefore IP number is 192.168.0.1) with the following command :

```
# nttcp -i
```

Then from another machine (assume **Gollach-2**) we can run a network receive with the following :

```
# nttcp -T -r 192.168.0.1
```

To conduct a transmission we change the `-r` option to `-t`. This will default to reading/writing 2048 buffers of 4 KB length (which is 8 MB). The kind of result that we get from this is shown below :

```
     Bytes  Real s   CPU s Real-MBit/s  CPU-MBit/s   Calls  Real-C/s   CPU-C/s
l  8388608    0.71    0.07     94.9101    958.6981    2048   2896.43   29257.1
1  8388608    0.71    0.28     94.0627    239.6745    5794   8121.12   20692.9
```

The figure we were most interested in was the top one in the column `Real-MBit/s`. This figure is the one which corresponds to the local host. The other row is for the remote host. We ran each read and write test that we required ten times and took average figures.

Loop-back Test

First we tested our ethernet response on the loop-back device. This was tested with TTCP. In a similar way to the `dd` test, this tells us if our benchmarking software is not limited to the real results we obtain. With this we ran a test on the 127.0.0.1 network. This gave us :

613.76Mb/s

Interface Test

Next we tested the network. Our cards are rated at 100Mbits/sec full duplex. The switch is rated at 3.2 Gbit/sec back-plane thus it should be very capable of handling all 12 ports talking simultaneously.

Running a test with TTCP on the default data set of 8192 KB for input and output, we get the following as the average :

94.7Mb/s

Then our NIC (Network Interface Card)s are running very close to their manufacturer's ratings. We also tested the connection to the outside network. This gave us a speed on a 10 Mb/s link of :

7Mb/s

5.4.2 Storage

The tests conducted covered the following areas:

- Single disk
- Two striped disks
- Striped NBD devices
- NBD devices on RAID-5

These tests were covered on the raw devices and then on the devices after being formatted to EXT3 file-system.

For our low level tests we want to test the speed of our hard drive. The manufacturer has specified the transfer rate of the given hard drives to be 16 MB/s. The controller is claimed to be capable of transfer speeds up to 40 MB/s. We need to know how these figures correspond to our drive when mated to the controllers in our machines.

Test Procedure for Storage Systems

The basic procedure used in doing all the file system tests was the same. We used `dd` to transfer data either *to* or *from* the device as required. When transferring *to* a device then we sourced our data from `/dev/zero`. When transferring *from* a device then we sent the data to `/dev/null`.

There were four types of test which were conducted on most of the storage devices. These were :

1. Raw device write,
2. Device with file-system write,
3. Raw device read, and
4. Device with file-system read.

The basic statement for writing to a raw device is the following:

```
#   time dd if=/dev/zero of=/dev/md0 bs=1M count=1000
```

In this case our device is `/dev/md0` which we are writing 1000 MB (or 1 GB). The statement for writing to a device with a file system is :

```
#   time dd if=/dev/zero of=/mnt/data/testfile bs=1M count=1000
```

This writes to the file `testfile` on some device mounted under the mount point `/mnt/data`. The statement for reading from a raw device is:

```
#   time dd if=/dev/md0 of=/dev/null bs=1M count=1000
```

Here we read 1000 MB from the device /dev/md0. To read from a file on a device with a file system required the statement:

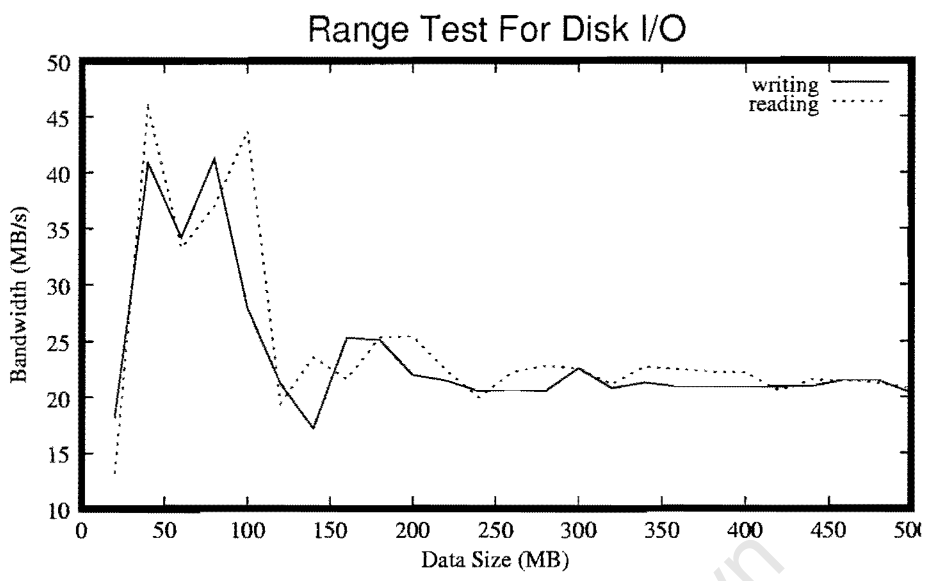


Figure 5.7: Results for the Range Test

```
#   time dd if=/mnt/data/testfile of=/dev/null bs=1M count=1000
```

This reads from the file `testfile` on the device mounted on mount point /mnt/data. In the case of reading a file, the file must exist. To ensure that the file was there before doing a read test, we always ran the file system write test prior to doing a read test. This ensured that the file `testfile` was created.

With the tests which had to cover some range, we would run the tests with a gradual change of the data size as required. For all of these tests, we did 10 runs of each test and then took an average. For those tests that had a range of data sizes, the averages were then used as data for plotting graphs.

Range Test

In our tests, we wanted to keep clear of anomalies which are caused by file-system buffers when doing disk bandwidth tests. We suspect the file-system caches are approximately 128 MB. Thus we should fall into a non-linearity with our file-system response. Using dd as above, we did a range test for sizes from 10 MB to 500 MB in steps of 20 MB and we got the results shown in Figure 5.7.

Table 5.1: Storage Test Results

Conditions		**Reading** MB/s	**Writing** MB/s
Single Disk	Raw	16.38	14.89
	FS	12.53	18.32
2 Disk Stripe	Raw	28.62	22.31
	FS	23.01	19.46
NBD Single Disk	Raw	9.35	7.64
	FS	9.66	8.21
NBD RAID-0	Raw	12.95	11.34
	FS	12.99	11.36
NBD RAID-5	Raw	11.00	7.02
	FS	*	*

Thus we can see that results below around 150 MB are quite anomalous, dubious and reach some over-optimistic heights. After 150 MB figures do not vary quite as wildly. Therefore smaller figures cannot be trusted for conducting these tests, although they can shorten the test period dramatically.

With this in mind, we ran our file-system tests with sizes of 1 GB.

Storage Test Results

The storage tests we conducted are summarised in the Table 5.1.

In this table we can see that raw disk speed is at best around 18 MB/s during writing. This figure improves to almost 22 MB/s case of striping. Thus there is a visible increase of speed after disk striping. Reading from a bare device peaked at around 16 MB/s for a single disk, while for the striped device was over 28 MB/s. This should mean that there are performance gains to be had from RAID combinations.

With the NBD we find that the resulting bandwidth is at best around 13 MB/s for the case of the striped configuration and about 11 MB/s for RAID-5. Thus for the case of striping we can rest our suspicions on the network as being the bottleneck. This fact can be augmented by the observation that the single NBD device also gives a similar result.

Taking another look we observe that the RAID-0 with file-system has exceeded the network limit of 12.5 MB/s. This should be possible due to the existence of local devices in the RAID array.

Storage Server

With these results in mind we proceeded to test the remote storage device's speed . As we calculated, a 100 Mb/s will translate to 12.5 MB/s. Thus since our striped drives returned a rate of about 23 Mb/s we expect the performance to only be limited by the network speed.

Again with our NFS server we got a similar result of :

$$11.2 \text{ MB/s}$$

This is the speed capacity of a conventional 5400 rpm IDE (Integrated Drive Electronics) device (usually used in conventional PCs) (See Appendix D). The speed diminish was attributed to the performance of the network. Thus we are being limited by the network and our possible speed has been diminished by more than 10 MB/s.

We also tested the home directories for their bandwidth since they were configured to take advantage of DFSA on MFS. Running a file read and write gave us results of

$$15.87 \text{ MB/s}$$

Which us way beyond what our network would allow. Therefore we have to believe that I/O is being conducted on the storage server itself. We then ran several simultaneous disk write processes so that we could see how the speed scales with demand. The result is shown in Figure 5.8.

In this figure we can see that the resultant bandwidth for a single process b is inversely proportional to the number of processes n conducting I/O i.e.

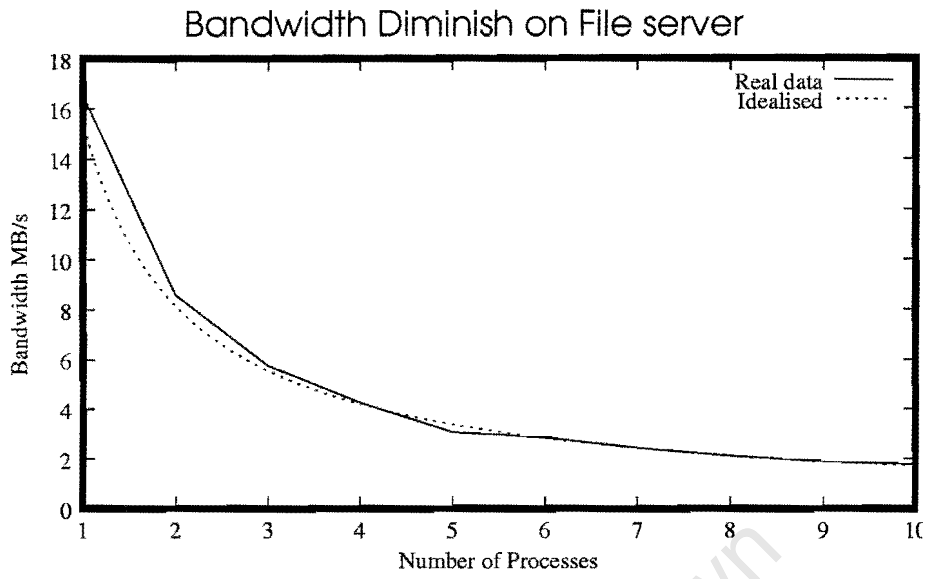


Figure 5.8: Plot of DFSA test

$$b = \frac{B}{n}$$

Where B is the total bandwidth of the device. That way total bandwidth remains constant. This total bandwidth is shown in Figure 5.9. This plot shows that bandwidth remains constant (around 16 MB/s).

5.4.3 MOSIX Low Level Test

To test MOSIX on a low level, we created a program that forked and created multiple processes. Each one of these processes contains a loop to occupy processor time. Therefore, with a large number of such processes, they should overload their home node. To counter this, they will migrate and run on all the machines in the cluster since they are not I/O bound.

We wrote the program to take the option -n. This allows us to change the number of processes that are spawned. In order to test the results of the cluster's response without MOSIX, we used the command :

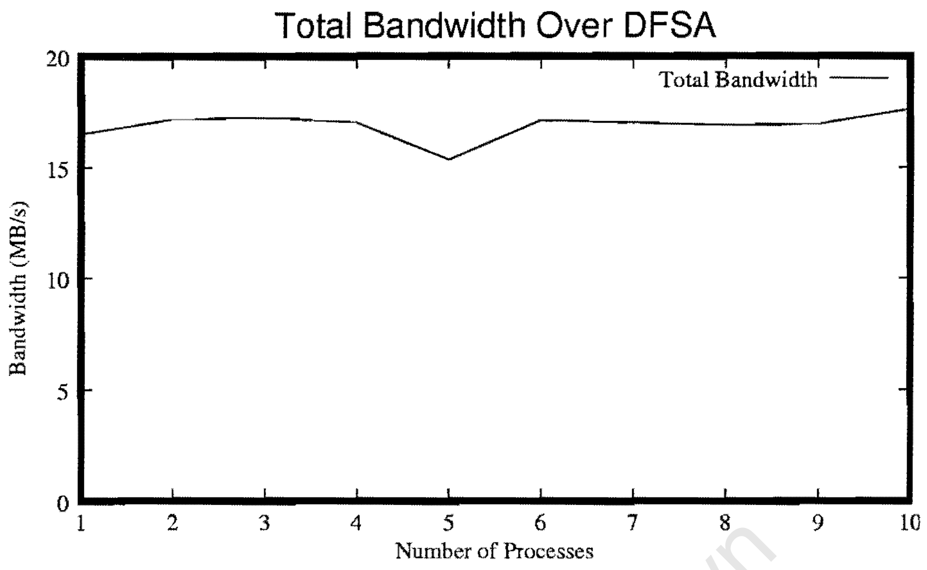


Figure 5.9: Plot of DFSA Total Bandwidth

```
# echo "1" > /proc/mosix/admin/stay
```

This forces all processes that are spawned on this node to stay and not migrate when required to by MOSIX algorithms.

The program code for this test is shown in Appendix B.

Test Procedure for MOSIX

The basic procedure with the MOSIX test was to time how long it took to complete the given processes. To get this time we used the following :

```
#   time -f \%E -a -o results.dat ./mos-test -n  1
```

The options on the time command will append the resulting time taken by the command to the file `result.dat`. As we can see on the `mos-test` command, we have given it a parameter of 1 process.

The above command was run with process counts up to 50 processes in steps of 1. This procedure was repeated ten times. The corresponding figures for each data set were averaged and then plotted on the graphs shown in the next section.

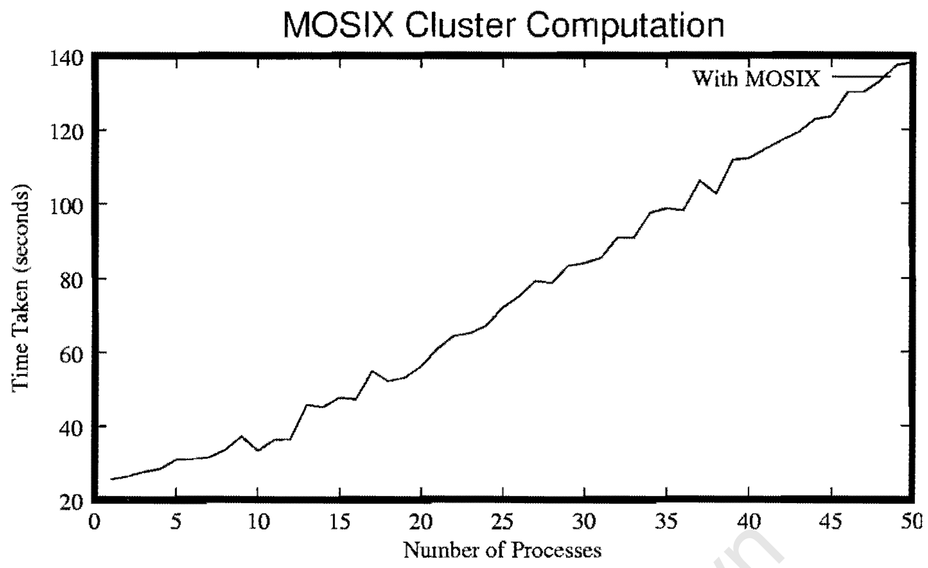


Figure 5.10: A Multiprocess Test with MOSIX

Results for MOSIX Test

Figures 5.10 and 5.11 shows the result of running the program for various numbers of processes in the cases with and without MOSIX.

In comparing these two cases we can see that the time to complete the given program code on a MOSIX cluster becomes approximately 9 times smaller than that on a single machine as the number of processes grows.

To be more exact, in our results for 50 Processes we had times of 1234.99 seconds and 138.11 seconds for the cases without MOSIX and with MOSIX respectively. Thus at this stage our factor is

$$\text{Factor} = 1234.99 \div 138.11 = 8.94 \text{ to 2 decimal places}$$

In order to better show how this factor grows we have plotted a graph showing the scale factor between our two cases as the number of processes grows. This is shown in Figure 5.12. In this figure we can see how our scale factor is scaling off at close to 9.

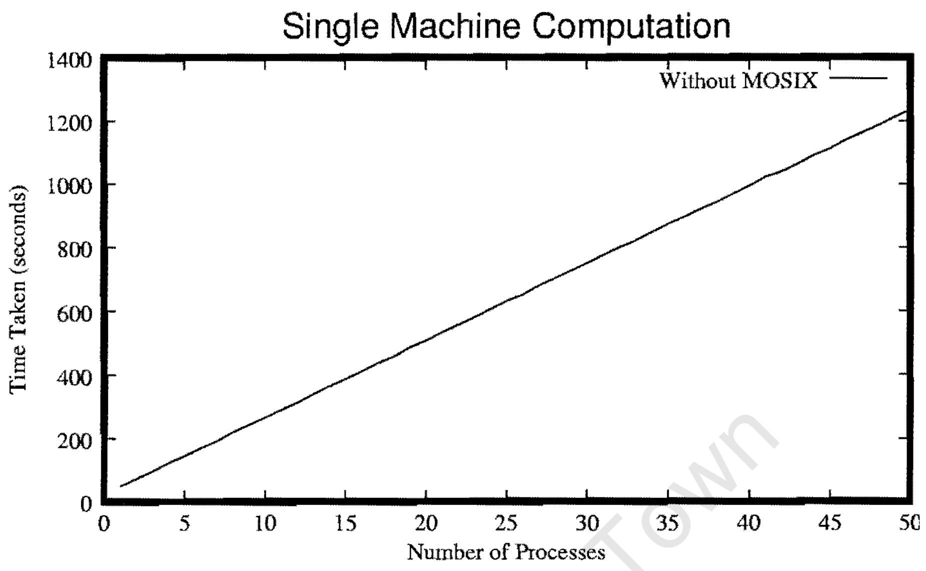


Figure 5.11: A Multiprocess Test With MOSIX Disabled

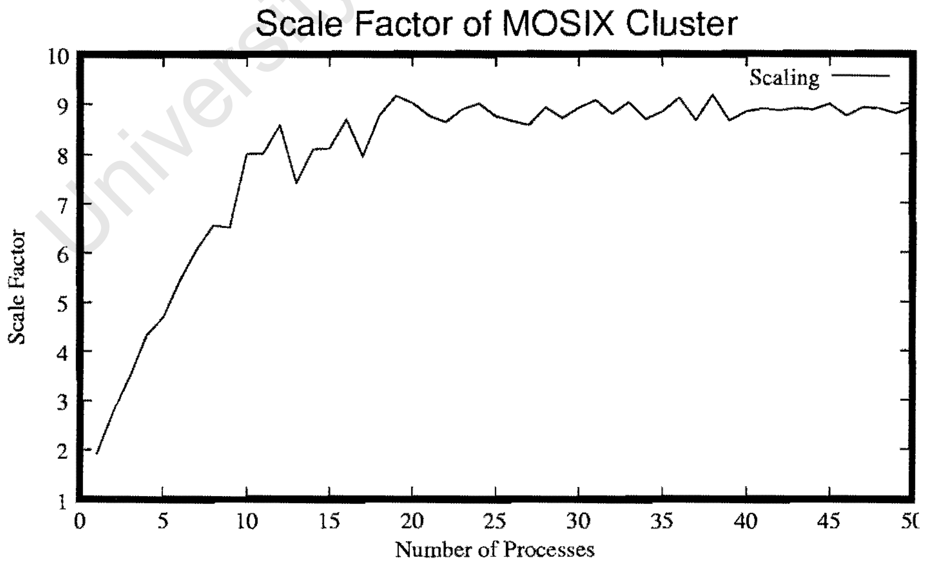


Figure 5.12: Scaling Factor Between MOSIX Cluster and Single Computer

5.5 Summary

This has been a chapter discussing testing of the cluster. We have mentioned how the cluster behaves when looked at from a high level. Here we find that the system meets most of our original requirements. We also mentioned how we conducted some lower level tests in order to acquire performance figures on a component level. Within this chapter we saw how our MOSIX processing speed scales with number of processors. We also saw how storage systems speed scales with demand.

Therefore covered both areas of tests, dealing with the *If*, (if the cluster does it) and *How*, (how well the cluster does it).

Chapter 6

Conclusions and Scope for Future Work

This is our final chapter and thus we sum up our findings. These range from the implementation and the testing phase. Therefore we can be able to setup a comparison with our initial requirements.

This project set out to create a cluster computer whose resources have been unified. The purpose of this is to have a scalable server built of conventional hardware. Thus, we would be able to access more computing power and harness more storage space without the need of a specially built server. On the other hand we did not want to burden the administrator and users of the server with a system that required special internal knowledge.

6.1 Cluster Components

We put together the fundamental parts of our computers, i.e.

1. I/O and display

2. Processor

3. Disk

4. Mass storage

In order to do this we took advantage of several software packages, these are: NBD, Grub, ClusterNFS, LVS, IPTables, DHCPD, software RAID, TFTP, SSH and MOSIX.

With the help of ClusterNFS, we were able to separate out the parts which are not shared on the master disk. Because of this, we basically have one computer to administer. If anything is to happen, it happens on that one machine. Any changes which need to be made are made on that one computer. Fortunately, since the computers share a file-system, the converse is also true and any changes made on other nodes is system-wide.

With LVS, we built something like an imitation SMP. This means users have access to a host of CPU's and do not suffer as much from the I/O bottlenecks which prevents MOSIX processes from migrating on the cluster.

With a single entry point, users could end up being trapped on the gateway machine. Thus for most I/O destined operations i.e Matlab, IDL which are some of the most useful programs in our requirements, we end up with all the users sitting on one machine and all we have is a normal single processor server to handle the load.

Using NBD we have been able to create a single mass storage device which saves users from storing data on multiple devices. This saves problems associated with tracking important information, or the case of data having to be spanned onto several devices.

6.2 Cluster in Operation

6.2.1 Administration

Once built, the administration of the cluster is nothing far different from a single Linux server. ClusterNFS is found to be virtually invisible during operation as a root file-system.

The only watch-point that was noticed was that of modifying machine specific files on the machine that the files are meant for. This is where a quirk was found in the way in which ClusterNFS over-wrote existing files. In the tests chapter this was mentioned as a case where the NFS server lost its own copy a file which was being edited for another machine.

6.2.2 Users

The resulting virtual server that we built worked as expected in most areas. Thus users could log-in onto the server using the `ssh` command. This gave a conventional log-in which was familiar.

On log-in, usage was just like a normal Linux server. Thus there were no hidden secrets in the operation requirements. ClusterNFS did not leave any traces of its existence in the users' home directories and thus was completely transparent in its operation.

6.3 Performance

Several items were measured in order to get figures for the performance of this cluster computer. The main things to be noted were

- Network bandwidth,
- NFS server drive speed,
- NFS export speed,
- NBD RAID speed,
- MOSIX parallelism and
- LVS distribution

6.3.1 Network Speed

The internal network used in this cluster consisted of 100MB/s network cards in the node machines connected with a 3.2 GB/s switch. Benchmarking our network with TTCP gave a figure of :

94.70Mb/s, which becomes 11.84MB/s

6.3.2 NFS Performance

Our NFS server had its drive built out of two 18 GB SCSI devices that were striped. Within the machine this device reported a read speed of :

23.01MB/s

and a write speed of :

19.46MB/s

On benchmarking our NFS root device, the reported bandwidth was:

11.20MB/s

Which is similar to the speed of a hard disk drive used in a conventional PC.

6.3.3 NBD Performance

A single NBD device exported, reported read and write speeds respectively of :

9.66MB/s and 8.21MB/s

the NBD performance. It has also been mentioned to be the limiting factor in our LVS configuration that employs Network Address Translation (LVS-NAT).

Therefore one thing clear from this whole project is the role that the network plays. It is the glue of the entire system and thus the performance of the network lubricates the systems components.

The role that the network plays can be seen in the ClusterNFS root, diskless boot, LVS, MOSIX process migration and NBD RAID storage.

Therefore the network is central to all this and knowing its performance is a real boon to making good judgement.

Otherwise our cluster works to our expectation on a conceptual level and gives us some performance benefits as we had hoped.

6.5 Future Work

Future work on this cluster server should be aimed at improving speed and building up resilience. As tests have shown, the cluster server suffers from vulnerable critical points which can halt the entire server workings. Therefore in order to improve resilience the following is probably a worthwhile route :

- Creating a redundant fail-over server handling the LVS director-ship. This sever should be built with a system to detect the main server's failures and should kick in as soon as they occur.

- Placing a redundant mirrored server to deal with the NFS services. This would be a move to spread the bandwidth requirements more evenly across the cluster. In such a case we will not have one machine funnelling down all the file I/O.

- Using a more improved file system for the root file server to allow for a more even spread of data on the cluster. In this case files could be more decentralized. This could

On combination in a RAID-0 array, the speeds become :

12.99MB/s and 11.36MB/s

These speeds were not far off from that of our NFS exported disk, thus we placed the same reasoning of network bottlenecks as being our sources of loss of performance.

6.3.4 MOSIX Operation

MOSIX was tested to run as expected with the help of a multiprocess application. Illustrations in the previous chapter show how processes were redistributed on the cluster during run-time.

Running a multiprocess application to spawn processes on this cluster showed that we could achieve factors of around 9 times the CPU power of a single machine from our cluster of 8 machines.

6.3.5 LVS

The LVS system ran according to our needs. Bar graphs showing user distribution have been presented. These have consistently reported an evened out user load on our cluster.

6.4 Conclusion

In terms of concept, we can safely claim to have the machine which was envisioned at the start. This cluster server can be administered and used like a conventional server. Also being a cluster there are some gains in performance due to the inherent parallelism.

As a performance server there are some shortcomings. Most of these have been traced down to the network. The limits imposed by our network performance have been the cause of a lot of bottlenecks. Network bandwidth has limited the NFS clients access speed and even out load since requests would be more distributed and improve resilience, since there is no single point of failure.

- Improve the networking hardware in the cluster with the motive to improve network response. Faster ethernet cards and switches are now available and they could be a first step in improving cluster speed.

Bibliography

[1] Daniel Ridge, Donald Becker, Phillip Merkey, Thomas Sterling,
Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCSs,
Proceedings, IEEE Aerospace, 1997.
http://www.beowulf.org/papers/papers.html

[2] Robert G. Brown,
Engineering a Beowulf-style Compute Cluster,
January 2002 Duke University Physics Department
http://www.phy.duke.edu/brahma/beowulf_online_book/beowulf_book.html

[3] Phil Merkey,
Beowulf: Introduction & Overview,
http://www.beowulf.org/intro.html

[4] Thomas E. Anderson, David E. Culler, David A. Patterson, and the NOW Team,
A case for NOW (Networks of Workstations),
IEEE Micro, Feb, 1995.
http://now.cs.berkeley.edu/Case/case.html

[5] David E. Culler, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Brent Chun, Steven Lumetta, Alan Mainwaring, Richard Martin, Chad Yoshikawa, Frederick Wong,
Parallel Computing on The Berkely NOW,
JSPP'97 (9th Joint Symposium on Parallel Processing) , Kobe, Japan
http://berkeley.intel-research.net/bnc/papers/jppf.pdf

[6] Wensong Zhang,
Linux Virtual Server for Scalable Network Services,
Ottawa Linux Symposium, 2000
http://www.linuxvirtualserver.org/Documents.html

[7] Wensong Zhang, Shiyao Jin, Quanyuan Wu, Joseph Mack,
Creating Linux Virtual Servers,

LinuxExpo 1999
http://www.linuxvirtualserver.org/Documents.html

[8] Michael Sparks,
Load Balancing The UK National JANET Web Cache Service Using Linux Virtual Servers,
UK National Janet Web Cache Service, November, 1999.
http://www.linuxvirtualserver.org/Documents.html

[9] *The LVS/NAT working principle and configuration instructions*
http://www.linuxvirtualserver.org/Documents.html

[10] *The LVS/TUN working principle and configuration instructions*
http://www.linuxvirtualserver.org/Documents.html

[11] *The LVS/DR working principle and configuration instructions*
http://www.linuxvirtualserver.org/Documents.html

[12] *Scheduling algorithms in LVS*
http://www.linuxvirtualserver.org/Documents.html

[13] *The LVS HOWTO*
http://www.linuxvirtualserver.org/Documents.html

[14] Rusty Russell,
Linux 2.4 NAT HOWTO,
http://www.netfilter.org/documentation/HOWTO//NAT-HOWTO.html

[15] McClure S. and Wheeler R.,
MOSIX: How Linux Clusters Solve Real World Problems,
Proc. 2000 USENIX Annual Tech. Conf., pp. 49-56, San Diego, CA., June 2000.

[16] Barak A., La'adan O. and Shiloh A.,
Scalable Cluster Computing with MOSIX for LINUX,
Proc. Linux Expo '99, pp. 95-100, Raleigh, N.C., May 1999.

[17] Barak A. and La'adan O.,
The MOSIX Multicomputer Operating System for High Performance Cluster Computing,
Journal of Future Generation Computer Systems, Vol. 13, No. 4-5, pp. 361-372, March 1998.

[18] Amar L., Barak A. and Shiloh A.,
The MOSIX Parallel I/O System for Scalable I/O Performance,
Proc. 14-th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2002), pp. 495-500, Cambridge, MA, Nov. 2002.

[19] Amar L., Barak A., Eizenberg A. and Shiloh A.,
The MOSIX Scalable Cluster File Systems for LINUX, July 2000.
http://www.mosix.org/

[20] Amnon Barak,
MOSIX
http://www.mosix.org

[21] Moshe Bar,
openMosix Internals: How openMosix Works,
http://openmosix.sourceforge.net/

[22] Kris Buytaert,
The openMosix HOWTO,
http://howto.ipng.be/openMosix-HOWTO/

[23] Rajkumar Buyya, Toni Cortes, Hai Jin,
Single System Image (SSI),
The International Journal of High Performance Computing Applications,
Volume 15, No. 2, Summer 2001, pp. 124-135

[24] Rajkumar Operating Systems Group, Centre for Development of Advanced Computing, India,
Single System Image: Need, Approaches and Supporting HPC Systems,
International Conference on Parallel and Distributed Processing Techniques and Applications, USA, 1997

[25] *EtherBoot*
http://etherboot.sourceforge.net/

[26] *Dynamic Host Configuration Protocol RFC 2131*
http://www.freesoft.org/CIE/RFC/2131/

[27] Erich Boleyn,
GRUB Technical Info,
http://www.uruk.org/orig-grub/technical.html

[28] Yoshinori K. Okuji,
GNU GRUB,
http://www.gnu.org

[29] Daniel P. Bovet, Marco Cesati,
Understanding the Linux Kernel (Book),
O'Reilly Publishing October 2000

[30] David A Rusling,
The Linux Kernel,
http://www.linuxhq.com/guides/TLK/tlk.html

[31] *The Linux Kernel Archives*
http://www.kernel.org/

[32] Rémy Card, Theodore Ts'o, Stephen Tweedie,
Design and Implementation of the Second Extended Filesystem,
Proceedings of the First Dutch International Symposium on Linux,

[33] Ellen Siever, Stephen Spainhour, Jessica P. Hekman, Stephen Figgins,
Linux in a Nutshell, 3rd Edition (Book),
O'Reilly Publishing 3rd Edition August 2000

[34] *Linux Online !*,
http://www.linux.org

[35] P. T. Breuer, A. Marn Lopez and Arturo Gar,
The Network Block Device,
Linux Journal Issue 73: May 2000

[36] Pavel Machek,
Network Block Device (TCP version),
http://atrey.karlin.mff.cuni.cz/ pavel/nbd/nbd.html

[37] Antoine Ginies,
PXE Documentation Version 1.0: HOWTO setup a PXE 2.x server under Linux,
http://people.mandrakesoft.com/ erwan/pxe/index.html

[38] Marc Vuilleumier Stckelberg, David Clerc,
Linux Remote-Boot mini-HOWTO: Configuring Remote-Boot Workstations with Linux, DOS, Windows 95/98 and Windows NT, February 2000
http://cui.unige.ch/info/pc/remote-boot/howto.html

[39] Tavis Barr, Nicolai Langfeldt, Seth Vidal, Tom McNeal,
Linux NFS-HOWTO, August 2002
http://www.linuxvoodoo.com/howto/HOWTO/NFS-HOWTO/

[40] Hans de Goede,
Root Over NFS Clients & Server Howto. , March 1999
http://www.tldp.org/HOWTO/Diskless-root-NFS-HOWTO.html

[41] Andreas Kostyrka,
NFS-Root Mini-Howto, August 1997
http://www.opennet.ru/docs/HOWTO/mini/NFS-Root.html

[42] Ofer Maor,
NFS-Root-Client Mini-HOWTO, Feb, 1999
http://www.ibiblio.org/pub/Linux/docs/HOWTO/mini/other-formats/html_single/NFS-Root-Client-mini-HOWTO.html

[43] Gregory R. Warnes,
ClusterNFS ,
http://clusternfs.sourceforge.net/

[44] Gregory R. Warnes, Ph.C.,
Cluster-NFS: Simplifying Linux Clusters,
Fred Hutchinson Cancer Research Center,
http://clusternfs.sourceforge.net/Presentation.html

[45] Gregory R. Warnes ,
A Recipe for a diskless MOSIX cluster using Cluster-NFS,
Fred Hutchinson Cancer Research Center, May 2000
http://ClusterNFS.sourceforge.net/Recipe.html

[46] Alan Robertson,
High-Availability Linux Project, October 2002
http://linux-ha.org/

[47] *IntelliStation RAID Technology: Information Brief*,
http://www.pc.ibm.com/us/infobrf/raidfin.html

[48] Mike Neuffer,
Linux High Performance SCSI & RAID,
http://www.uni-mainz.de/ neuffer/scsi/what_is_raid.html

[49] Jakob stergaard,
The Software-RAID HOWTO,
http://en.tldp.org/HOWTO/Software-RAID-HOWTO.html

[50] Peter J. Braam,
The Coda Distributed File System,
School of Computer Science,
Carnegie Mellon University
http://www.coda.cs.cmu.edu/ljpaper/lj.html

[51] Craig Hunt,
TCP/IP Network Administration (Book),
O'Reilly Publishing, pg 226 to 239

[52] *Securing Remote Connections with Secure Shell*,
White Paper, October 2002,
http://www.ssh.com/support/documentation/white_papers/

[53] RedHat Linux,
http://www.redhat.com/

Appendix A

Listings of Script Files

Director Machine Configuration

This first file is a script file to configure the LVS director machine named `lvs-director-start`. The two LVS scripts are run by a line in `/etc/rc.d/rc.local`

```
#!/bin/sh

#set ip_forward ON for vs-nat director (1 on, 0 off).
cat /proc/sys/net/ipv4/ip_forward
echo "1" >/proc/sys/net/ipv4/ip_forward

#director is gw for realservers
#turn OFF icmp redirects (1 on, 0 off)
echo "0" >/proc/sys/net/ipv4/conf/all/send_redirects
cat       /proc/sys/net/ipv4/conf/all/send_redirects
echo "0" >/proc/sys/net/ipv4/conf/default/send_redirects
cat       /proc/sys/net/ipv4/conf/default/send_redirects
echo "0" >/proc/sys/net/ipv4/conf/eth0/send_redirects
cat       /proc/sys/net/ipv4/conf/eth0/send_redirects

#setup VIP
#/sbin/ifconfig eth0:110 192.168.2.110 broadcast 192.168.2.255 netmask 255.255.255.0

#set default gateway
#/sbin/route add default gw 192.168.2.254 netmask 0.0.0.0 metric 1
/sbin/route add default gw 137.158.131.3 netmask 0.0.0.0 metric 1

#clear ipvsadm tables
/sbin/ipvsadm -C

#install LVS services with ipvsadm
#add telnet to VIP with rr sheduling
```

```
/sbin/ipvsadm -A -t 137.158.131.3:ssh -s wlc


#set up  realservers
#forward ssh to realserver 192.168.0.x using LVS-NAT (-m), with weight=1


#set up the realservers
/sbin/ipvsadm -a -t 137.158.131.3:ssh -r 192.168.0.1:ssh -m -w 2
ping -c 1 192.168.0.1
/sbin/ipvsadm -a -t 137.158.131.3:ssh -r 192.168.0.2:ssh -m -w 1
ping -c 1 192.168.0.2
/sbin/ipvsadm -a -t 137.158.131.3:ssh -r 192.168.0.3:ssh -m -w 1
ping -c 1 192.168.0.3
/sbin/ipvsadm -a -t 137.158.131.3:ssh -r 192.168.0.4:ssh -m -w 1
ping -c 1 192.168.0.4
/sbin/ipvsadm -a -t 137.158.131.3:ssh -r 192.168.0.5:ssh -m -w 1
ping -c 1 192.168.0.5
/sbin/ipvsadm -a -t 137.158.131.3:ssh -r 192.168.0.6:ssh -m -w 1
ping -c 1 192.168.0.6
/sbin/ipvsadm -a -t 137.158.131.3:ssh -r 192.168.0.7:ssh -m -w 1
ping -c 1 192.168.0.7
/sbin/ipvsadm -a -t 137.158.131.3:ssh -r 192.168.0.8:ssh -m -w 1
ping -c 1 192.168.0.8

# set up the IP masquerading
/sbin/iptables -t nat -A POSTROUTING  -o eth1 -j MASQUERADE
/sbin/iptables -A FORWARD  -i eth0 -j ACCEPT

#list ipvsadm table
/sbin/ipvsadm
```

Real Server Configuration

The following is the code for the real servers configuration file named `lvs-real-start`:

```
#!/bin/sh
#installing default gw 192.168.0.1 for vs-nat'
#/sbin/route add default gw 192.168.0.1
/sbin/route add default gw 192.168.0.1

#show routing table
/bin/netstat -rn

#checking if DEFAULT_GW is reachable
ping -c 1 192.168.0.1

#looking for VIP on director from realserver
ping -c 1 137.158.131.3

#set_realserver_ip_forwarding to OFF (1 on, 0 off).
echo "0" >/proc/sys/net/ipv4/ip_forward
cat      /proc/sys/net/ipv4/ip_forward
```

NBD Configuration

This is a script file to initialise the NBD clients in the machine **Gollach-7**. The servers are started by the following single line inserted into /etc/rc.d/rc.local :

```
   /usr/local/sbin/nbd-server 1700 /dev/md0
```

The above line exports the device /dev/md0 on port 1700. Then the following is the clients script :

```
#!/bin/sh

# first we start up the clients

/usr/local/sbin/nbd-client g1 1700 /dev/nd0
/usr/local/sbin/nbd-client g2 1700 /dev/nd1
/usr/local/sbin/nbd-client g3 1700 /dev/nd2
/usr/local/sbin/nbd-client g4 1700 /dev/nd3
/usr/local/sbin/nbd-client g5 1700 /dev/nd4
/usr/local/sbin/nbd-client g6 1700 /dev/nd5

# then we create a raid device

/usr/sbin/mkraid  /dev/md1 -c /etc/raidtab2

# finally we mount our drive

/usr/sbin/mount /dev/md1 /real_root/data
```

Appendix B

Listing of MOSIX Test Code

This is the listing of the program code that we use in testing the efficiency of our MOSIX cluster. In essence it is a program that forks looping processes. Each process occupies the CPUs resources and then signals the original forking parent on completion.

This line of testing is motivated by MOSIX's fork and forget mode of operation. Therefore we fork and wait for MOSIX to do the rest for us.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>

#define round(x) ((x)>=0?(int)((x)+.9):(int)((x)-.1))
#define trunc(x) ((int)(x))

typedef void (*sighandler_t)(int);

static sighandler_t sig;

int sig_counter = 0;
int num_signals = 1;
int thirty_percent = 1;
int PID = 0;

static void signal_handler(int signal_number)
{
  sigset_t mask_set;
  sigset_t old_set;

  // we must mask our other signals to avoid race conditions
  sigfillset(&mask_set);
  sigprocmask(SIG_SETMASK, &mask_set, &old_set);
```

```
      char buffer[128];
      sprintf(buffer,
      "%s: signal(%d, signal_handler) failed",
      argv[0], SIGHUP);
      perror(buffer);
      exit(1);
    }

  PID = getpid();

  // This is within the parent process
    printf("\nHandler  PID = %d, slaves = %d\n", PID, num_signals);

  // now to fork the parallel slaves
    for(i = 0; i < num_signals; i++)
    {
      PID2 = fork();

      // The   parent process must continue forking slaves
      // while the child must break off and begin computation.
      if(PID2 == 0)
break;
     }

  if(PID2 == 0)
    {
      // This is now within the slave process

      // Our CPU occupying loop representing a computation.
      for(j = 0; j <= 10; j++)
for(k = 0; k <= 100000000; k++);

      // send a signal to the signal handling slave on completion
      kill(PID, SIGHUP);

    }

  else
    // a busy waiting loop for the signal handler
    while(1);


return 0;
}
```

```
  sig_counter++;

  //  printf("\nsignal %d\n", sig_counter);

  signal(signal_number, signal_handler);


  // respond if all processes have done interrupting
  // the program terminates if thirty percent of the processes forked respond
  // this is due to race conditions during process end signalling when
  // some signals are not caught. Which also shows that processes are ending at
  // nearly the same time.

  if(sig_counter >= thirty_percent)
    {
      printf("\nProgram Complete .\n");
      exit(1);
    }
}

// The main function itself. This initialises the singal handler,
// the PID is made available to the computing slaves. These then interrupt
// it in order signal an end of computation.

int main(int argc, char *argv[])
{
  int PID2 = 0;
  int i = 0,j,k;

  if(argc > 1)
    {

      if(!strcmp(argv[1], "-n"))
{
  num_signals = atoi(argv[2]);

}
    }

    thirty_percent = (int) round(0.30 * (double) num_signals);

  // starting up the signal handler

  sig = signal(SIGHUP, signal_handler);

  if (sig == SIG_ERR)
    {
```

Appendix C

Gollach Kernel Configuration

The following is the complete kernel configuration for the Gollach cluster. This is basically a listing of the file `.config`, i.e. the file written to by the configuration program. To make this more concise, options which were commented have been left out. It has been formatted as three columns where each consecutive column is a continuation from the endpoint of the previous column. Columns continue across pages.

```
CONFIG_X86=y
CONFIG_ISA=y
CONFIG_UID16=y


# MOSIX
CONFIG_MOSIX=y
CONFIG_MOSIX_DIAG=y
CONFIG_MOSIX_SECUREPORTS=y
CONFIG_MOSIX_DISCLOSURE=1
CONFIG_MOSIX_EXTMOSIX=y
CONFIG_MOSIX_DFSA=y
CONFIG_MOSIX_FS=y

# Code maturity level options
CONFIG_EXPERIMENTAL=y


# Loadable module support
CONFIG_MODULES=y

CONFIG_MODVERSIONS=y
CONFIG_KMOD=y

# Processor type and features

CONFIG_M686=y
CONFIG_X86_WP_WORKS_OK=y
CONFIG_X86_INVLPG=y
CONFIG_X86_CMPXCHG=y
CONFIG_X86_XADD=y
CONFIG_X86_BSWAP=y
CONFIG_X86_POPAD_OK=y

CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_X86_L1_CACHE_SHIFT=5
CONFIG_X86_TSC=y
CONFIG_X86_GOOD_APIC=y
CONFIG_X86_PGE=y
CONFIG_X86_USE_PPRO_CHECKSUM=y
```

```
CONFIG_IP_NF_COMPAT_IPCHAINS=m
CONFIG_IP_NF_NAT_NEEDED=y

# IP: Virtual Server
Configuration
CONFIG_IP_VS=m
CONFIG_IP_VS_DEBUG=y
CONFIG_IP_VS_TAB_BITS=12

# IPVS scheduler
CONFIG_IP_VS_RR=m
CONFIG_IP_VS_WRR=m
CONFIG_IP_VS_LC=m
CONFIG_IP_VS_WLC=m
CONFIG_IP_VS_LBLC=m
CONFIG_IP_VS_LBLCR=m
CONFIG_IP_VS_DH=m

CONFIG_IP_VS_SH=m


# IPVS application helper
CONFIG_IP_VS_FTP=m

# QoS and/or fair queueing


# Telephony Support

# ATA/IDE/MFM/RLL support
CONFIG_IDE=y

# IDE, ATA and ATAPI Block
devices
CONFIG_BLK_DEV_IDE=y

CONFIG_BLK_DEV_IDEDISK=y
CONFIG_IDEDISK_MULTI_MODE=y
CONFIG_BLK_DEV_IDECD=y
```

```
# Character devices
CONFIG_VT=y
CONFIG_VT_CONSOLE=y
CONFIG_SERIAL=y

CONFIG_UNIX98_PTYS=y
CONFIG_UNIX98_PTY_COUNT=256

# I2C support

# Mice
CONFIG_MOUSE=y
CONFIG_PSMOUSE=y

# Joysticks

# Input core support is
needed for gameports

# Input core support is
needed for joysticks

# Watchdog Cards

# Ftape, the floppy tape
device driver
CONFIG_AGP=y
CONFIG_AGP_INTEL=y
CONFIG_AGP_I810=y
CONFIG_AGP_VIA=y
CONFIG_AGP_AMD=y
CONFIG_AGP_SIS=y
CONFIG_AGP_ALI=y

CONFIG_DRM=y

# DRM 4.1 drivers
#
CONFIG_DRM_NEW=y
CONFIG_DRM_TDFX=y
```

```
CONFIG_X86_PPRO_FENCE=y

CONFIG_NOHIGHMEM=y
CONFIG_SMP=y
CONFIG_HAVE_DEC_LOCK=y

# General setup
CONFIG_NET=y
CONFIG_X86_IO_APIC=y
CONFIG_X86_LOCAL_APIC=y
CONFIG_PCI=y
CONFIG_PCI_GOANY=y
CONFIG_PCI_BIOS=y
CONFIG_PCI_DIRECT=y
CONFIG_PCI_NAMES=y
CONFIG_HOTPLUG=y

# PCMCIA/CardBus support
CONFIG_PCMCIA=y

CONFIG_CARDBUS=y

# PCI Hotplug Support
CONFIG_SYSVIPC=y


CONFIG_SYSCTL=y
CONFIG_KCORE_ELF=y
CONFIG_BINFMT_AOUT=y
CONFIG_BINFMT_ELF=y
CONFIG_BINFMT_MISC=y
CONFIG_PM=y


# Memory Technology Devices
(MTD)

# Parallel port support

# Plug and Play configuration
CONFIG_PNP=y
CONFIG_ISAPNP=y

# IDE chipset
support/bugfixes
CONFIG_BLK_DEV_CMD640=y
CONFIG_BLK_DEV_RZ1000=y
CONFIG_BLK_DEV_IDEPCI=y
CONFIG_IDEPCI_SHARE_IRQ=y
CONFIG_BLK_DEV_IDEDMA_PCI=y
CONFIG_BLK_DEV_ADMA=y
CONFIG_IDEDMA_PCI_AUTO=y
CONFIG_BLK_DEV_IDEDMA=y
CONFIG_BLK_DEV_PIIX=y
CONFIG_PIIX_TUNING=y
CONFIG_IDEDMA_AUTO=y
CONFIG_BLK_DEV_IDE_MODES=y

# SCSI support
CONFIG_SCSI=y

# SCSI support type (disk,
tape, CD-ROM)
CONFIG_BLK_DEV_SD=y
CONFIG_SD_EXTRA_DEVS=40

# Some SCSI devices (e.g.  CD
jukebox) support multiple
LUNs
CONFIG_SCSI_DEBUG_QUEUES=y
CONFIG_SCSI_MULTI_LUN=y
CONFIG_SCSI_CONSTANTS=y

# SCSI low-level drivers
CONFIG_SCSI_SYM53C8XX_2=y
CONFIG_SCSI_SYM53C8XX_DMA_ADDRE
SSING_MODE=1
CONFIG_SCSI_SYM53C8XX_DEFAULT_TA
GS=16
CONFIG_SCSI_SYM53C8XX_MAX_TAGS=64

# PCMCIA SCSI adapter support

# Fusion MPT device support

CONFIG_DRM_RADEON=y


# PCMCIA character devices

# Multimedia devices

# File systems
CONFIG_AUTOFS4_FS=y
CONFIG_REISERFS_FS=y
CONFIG_EXT3_FS=y
CONFIG_JBD=y
CONFIG_FAT_FS=y
CONFIG_MSDOS_FS=y
CONFIG_VFAT_FS=y
CONFIG_TMPFS=y
CONFIG_ISO9660_FS=y
CONFIG_PROC_FS=y
CONFIG_DEVPTS_FS=y

CONFIG_EXT2_FS=y

# Network File Systems
CONFIG_NFS_FS=y


CONFIG_ROOT_NFS=y
CONFIG_NFSD=y
CONFIG_SUNRPC=y
CONFIG_LOCKD=y

# Partition Types
CONFIG_MSDOS_PARTITION=y

CONFIG_NLS=y


# Native Language Support
CONFIG_NLS_DEFAULT="iso8859-1"

# Console drivers
CONFIG_VGA_CONSOLE=y
```

```
# Block devices
CONFIG_BLK_DEV_FD=y
CONFIG_BLK_DEV_LOOP=y
CONFIG_BLK_DEV_NBD=m

# Multi-device support (RAID
and LVM)
CONFIG_MD=y
CONFIG_BLK_DEV_MD=y
CONFIG_MD_LINEAR=y
CONFIG_MD_RAID0=y
CONFIG_MD_RAID1=y
CONFIG_MD_RAID5=y

# Networking options
CONFIG_PACKET=y
CONFIG_PACKET_MMAP=y
CONFIG_NETFILTER=y
CONFIG_UNIX=y

CONFIG_INET=y
CONFIG_IP_MULTICAST=y

CONFIG_IP_PNP=y
CONFIG_IP_PNP_BOOTP=y
CONFIG_SYN_COOKIES=y

# IP: Netfilter Configuration

CONFIG_IP_NF_CONNTRACK=m


CONFIG_IP_NF_FTP=m
CONFIG_IP_NF_IRC=m
CONFIG_IP_NF_QUEUE=m
CONFIG_IP_NF_IPTABLES=m
CONFIG_IP_NF_MATCH_LIMIT=m
CONFIG_IP_NF_MATCH_MAC=m

# IEEE 1394 (FireWire)
support (EXPERIMENTAL)

# I2O device support

# Network device support
CONFIG_NETDEVICES=y


# ARCnet devices
CONFIG_DUMMY=m

# Ethernet (10 or 100Mbit)
CONFIG_NET_ETHERNET=y
CONFIG_NET_PCI=y
CONFIG_EEPRO100=y
CONFIG_NE2K_PCI=y
CONFIG_8139TOO=y

# Ethernet (1000 Mbit)

# Wireless LAN (non-hamradio)

# Token Ring devices

# Wan interfaces

# PCMCIA network device
support
CONFIG_NET_PCMCIA=y


CONFIG_PCMCIA_PCNET=y
CONFIG_NET_PCMCIA_RADIO=y
CONFIG_PCMCIA_RAYCS=y

# Amateur Radio support

# Frame-buffer support

# Sound
CONFIG_SOUND=y
CONFIG_SOUND_ES1371=y


# USB support
CONFIG_USB=y

# Miscellaneous USB options

# USB Controllers
CONFIG_USB_UHCI_ALT=y

# USB Device Class drivers
CONFIG_USB_STORAGE=y

# USB Human Interface Devices
(HID)

# Input core support is
needed for USB HID

# USB Imaging devices

# USB Multimedia devices


# Video4Linux support is
needed for USB Multimedia
device support

# USB Network adaptors

# USB port drivers

# USB Serial Converter
support
```

```
CONFIG_IP_NF_MATCH_MULTIPORT=m
CONFIG_IP_NF_MATCH_STATE=m
CONFIG_IP_NF_MATCH_UNCLEAN=m
CONFIG_IP_NF_FILTER=m
CONFIG_IP_NF_TARGET_REJECT=m

CONFIG_IP_NF_NAT=m
CONFIG_IP_NF_NAT_NEEDED=y
CONFIG_IP_NF_TARGET_MASQUERADE=m
CONFIG_IP_NF_NAT_IRC=m
CONFIG_IP_NF_NAT_FTP=m
CONFIG_IP_NF_MANGLE=m
CONFIG_IP_NF_TARGET_MARK=m
CONFIG_IP_NF_TARGET_LOG=m
CONFIG_IP_NF_TARGET_TCPMSS=m
```

```
# IrDA (infrared) support

# ISDN subsystem

# Old CD-ROM drivers (not
SCSI, not IDE)

# Input core support
CONFIG_INPUT_MOUSEDEV_SCREEN_X=1024
CONFIG_INPUT_MOUSEDEV_SCREEN_Y=768
```

```
# USB Miscellaneous drivers

# Bluetooth support

# Kernel hacking
```

Appendix D

Some Hardware Specifications

The following shows some specifications of hardware that was used in the gollach cluster. This shows the hard disk drives, ethernet controllers and SCSI controllers. The only member of these specifications which was not actually used here is the Western Digital IDE hard disk drive. This was included here for comparison purposes. All the data shown in this appendix was acquired from the respective manufacter's data sheets.

IBM Ultrastar 9LP, 18XP 3.5-inch 9.1 GB and 18.2 GB drives

Configuration	**DGHS Ultra SCSI**
Interface	Fast, Fast/Wide, SCA-2
Formatted capacity(512 bytes)	9.1/ 18.2 GB *
Number of disks	5/10
Number of heads	10/20
Areal density (maximum)	1253 Mbits/sq. in.
Recording density (maximum)	150 KBPI
Track density	8356 TPI Sector size 512 to 732 Bytes
Channel	PRML
Encoding method	RLL (16/17)
Head type	Magnetoresistive Extended (MRX)
Dedicated landing zone	Yes
Performance	
Media data rate (banded)	92.2 to 179.2 Mbits/sec
Interface transfer rate (max)	40 MB/sec **
Access times (pop. avg)	
- Average read	6.5/7.5 ms
- Track-to-track read	0.7 ms
Rotational speed	7200 RPM
Latency (average)	4.17 ms
Buffer size	1 MB +

* MB = 1,000,000 Bytes; 1 GB = 1,000,000,000 Bytes ** 40 MB/sec represents Ultra Fast/Wide + Up to 337 KB used for firmware 6 Idle power dissipation/formatted capacity

Specifications for the Western Digital Caviar AC34300

Recommended Setup Parameters

Cylinders	8896
Heads	15
Sectors/Track	63
Landing Zone	8896
WPC	13328
Jumper Setting Information	Ten Pin Drive

Physical Specifications

Formatted Capacity*	4,304 MB
Interface	40-pin EIDE
Actuator Type	Rotary Voice Coil
Number of Platters	3
Data Surfaces	6
Number of Heads	6
Bytes Per Sector	512
User Sectors Per Drive	8,406,720
Servo Type	Embedded
Recording Method	GCR 8,9 - PRML
ECC	Reed Solomon
Head Park**	Automatic

* Western Digital defines a megabyte (MB) as 1,000,000 bytes and a gigabyte (GB) as 1,000,000,000 bytes

** Turning the system power off causes the WD Caviar to perform an automatic head park operation.

Performance Specifications

Average Seek (Read)	11 ms
Track to Track Seek	3.0 ms
Full Stroke Seek	21 ms
Average Latency	5.5 ms
Rotational Speed	5400 RPM
Transfer Rate (Buffer to Host)	33.3 MB/s (Mode 2 Ultra ATA)
	16.6 MB/s (Mode 4 PIO)
	16.6 MB/s (Mode 2 multi-word DMA)
Transfer Rate (Buffer to Disk)	68 Mbits/s minimum
	135.5 Mbits/s maximum
Interleave	1:1
Buffer Size	256 K DRAM
Error Rate (Non-Recoverable)	¡1 in 1013 bits read
Spindle Start Time	
- From Power-on to Drive Ready*	11s typical, 18s maximum
- From Power-on to Rotational Speed	7s typical, 15s maximum
Contact Start/Stop Cycles (CSS)	40,000 minimum

* Defined as the time from power-on to the setting of the Drive Ready and Seek Complete including calibration.

Intel PRO/100 S Server Ethernet Adapters

Adapter Specifications

Feature	Value
PCI bus compatibility	PCI 2.2 specification
Connector	RJ-45
Power requirements	1.06 watts @ +5VDC
Interrupt	INTA
LEDs	Two: Link/Activity, 100 Mbps
Duplex mode	Half or Full duplex
Data Rate	10 or 100 Mbps
Transmit/Receive Buffer	6 Kbytes

Symbios 53C875 SCSI Controller

SCSI Controller Performance Features

To improve SCSI performance, the LSI53C875:

- Includes 4 Kbyte internal RAM for SCRIPTS instruction storage.
- Performs wide, Ultra SCSI synchronous transfers as fast as 40 Mbytes/s.
- Increases SCSI synchronous offset from 8 to 16 levels.
- Supports variable block size and scatter/gather data transfers.
- Performs sustained memory-to-memory DMA transfers faster than 47 Mbytes/s (@ 33 MHz).
- Minimizes SCSI I/O start latency.
- Performs complex bus sequences without interrupts, including restore data pointers.
- Reduces interrupt service routine overhead through a unique interrupt status reporting method.
- Performs fast and wide SCSI bus transfers in SE and differential mode. 10 Mbytes/s asynchronous (20 Mbytes/s with Ultra SCSI). 20 Mbytes/s synchronous (40 Mbytes/s with Ultra SCSI).
- Supports Load and Store SCRIPTS instructions to increase the performance of data transfers to and from chip registers.
- Supports target disconnect and later reconnect with no interrupt to the system processor.
- Supports multithreaded I/O algorithms in SCSI SCRIPTS with fast I/O context switching.
- Supports expanded Register Move instructions to support additional arithmetic capability.
- Complies with PCI Bus Power Management Specification (LSI53C875E) Revision 1.0.