An extensible administration and configuration tool for Linux clusters

John D. Fogarty B.Sc

A dissertation submitted to the University of Dublin,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

1999

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____
John D. Fogarty
15th September, 1999

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____
John D. Fogarty
15th September, 1999

Summary

This project addresses the lack of system administration tools for Linux clusters. The goals of the project were to design and implement an extensible system that would facilitate the administration and configuration of a Linux cluster. Cluster systems are inherently scalable and therefore the cluster administration tool should also scale well to facilitate the addition of new nodes to the cluster.

The tool allows the administration and configuration of the entire cluster from a single node. Administration of the cluster is simplified by way of command replication across one, some or all nodes. Configuration of the cluster is made possible through the use of a flexible, variables substitution scheme, which allows common configuration files to reflect differences between nodes. The system uses a GUI interface and is intuitively simple to use.

Extensibility is incorporated into the system, by allowing the dynamic addition of new commands and output display types to the system. Through the use of a menus configuration file the system is easily extended to include additional commands. This can be accomplished without reprogramming for the majority of commands. An API has been provided to allow different types of component to be displayed on the main GUI panel. The system thus exhibits extensibility in supporting different types of display components such as text or graphics. This extensible scheme can then be used to support, for example, graphical output.

Through the use of a configuration file for the nodes in the system, the system can scale well. New nodes are easily added through a single entry in this configuration file. Scalability is also incorporated into the design of the system, whereby as much work as possible is distributed to the server nodes.

Acknowledgements

I would like to thank my supervisor Dr. Paddy Nixon for his advice and assistance throughout the project. I would also like to express my appreciation to Dr. Simon Dobson who also assisted me during this work. Stefan Weber was unstintingly generous with his time and his expertise so “danke schon”. Finally, cheers to my classmates.

TABLE OF CONTENTS

Table of Figures

Table of Tables

Chapter 1 Introduction

Introduction

Clusters of computers, where a cluster is a group of independent systems that act and appear as a single system, have become popular in both research establishments in need of high performance computing power and in business organisations seeking high availability from their computing resources. The popularity of clusters has arisen due to a number of factors. Not least amongst these, particularly with regards to research computing, has been the dramatic increase in price-performance of PC chips which has shown much greater improvement vis a vis traditional supercomputer components. This has led people to experiment with using commodity off the shelf components in building their supercomputers out of clusters of PCs. Other factors that have reinforced the interest in clusters include the fact that clusters scale well. They can grow very large, accommodating a large number of nodes and also nodes can be added in a piecemeal fashion.

Linux is the dominant operating system with regard to cluster systems. It is a popular choice mainly because it is free, stable and full source code is available. Additionally Linux implements a feature of operating system customisability whereby modules can be loaded and unloaded on-demand. This is particularly important in relation to clusters, where often, small kernels are desired in order to maximise performance. It also means that different nodes in the cluster can run modified versions of the kernel if needed.

Motivations

Concomitant with the deployment of clusters, is the need for tools to facilitate their administration. Administration in this context relates to the normal administrative duties needed to effectively manage a single machine. The bulk of research effort related to cluster software has up to now been devoted to the development of management tools to aid parallel programming, job scheduling and load balancing. A lack of administration tools continues to render the management, particularly of large scale clusters, more difficult. This has been the main motivation underpinning this project.

Goals

The goals of the project were to design and implement an extensible system that would facilitate the administration of a Linux cluster. Extensibility would allow new commands to be easily added to the system – the system should easily accommodate commands that return different types of output.

One of the features and main attractions of cluster systems is their scalability, which allows nodes to be added incrementally and supports a large number of nodes. The cluster administration tool should also scale well. In addition to these general design goals, a number of specific requirements are demanded of the system.

Roadmap

A conventional thesis pattern is used. This chapter has given an introduction to the project and the problem that is being addressed. The motivations & goals of the project were described.

Chapter 2 reviews the field of cluster systems and cluster administration tools. It explains the essential characteristics of clusters and describes the different classes of clusters. It also gives examples of the major types of clusters in use today. The place of cluster machines in the hierarchy of parallel computing machines is examined. The chapter then turns its attention to the software that is used on these machines. Specifically a description of available administration tools is also given.

Chapter 3 gives a detailed problem specification. The chapter details the goals of the project and outlines the requirements of the system.

Chapter 4 details the design of the cluster management system. It explains how the various goals and requirements of the system will be accomplished through the proposed design.

Chapter 5 describes the implementation of the system. It details how particular design features were incorporated into the system. It also outlines any problems encountered during the implementation.

Chapter 6 gives a detailed analysis of the cluster tool developed in this project. It reviews the goals and requirements of the system and assesses the extent to which these have been achieved. The chapter also suggests what future work might be carried out to extend and improve the system.

Summary

A broad overview of the project has been given in this chapter. The problem under consideration has been outlined along with the motivations and goals of the project. The next chapter examines the field of cluster computing and examines the software tools available to administer a Linux cluster.

Chapter 2 Background Research

Introduction

The problem being investigated by this project is the design and implementation of a Linux cluster administration tool. This chapter explains the essential characteristics of clusters and examines the different types of clusters that are in use today. Contrasts are made with other types of parallel machines such as SMPs (Symmetric Multi-Processors) and NOWs (Network of Workstations). Examples of the different types of clusters are used to illustrate the flexibility and power of this type of system.

The chapter concludes with an examination of the software that is used on these machines. Specifically, software relating to system administration is detailed.

Cluster definition

Achieving a precise definition for a cluster is difficult, as evidenced by the inconclusive and sometimes rancorous electronic submissions of the IEEE Task Force on Cluster Computing [HREF1]. However a consensus of sorts was formed around the following definition:

"A cluster is a type of parallel or distributed system that consists of a collection of interconnected whole computers used as a single, unified computing resource" [FIS97].

[Whole computer in this context is taken to mean a normal computer that can be used on its own: i.e. contains processor(s), memory, I/O, OS, software subsystems and applications.]

An alternative but similar definition given by Microsoft [HREF2]:

"A cluster is a group of independent systems working together as a single system. A client interacts with the cluster as if it were a single server. A node is a server in the cluster."

Cluster characteristics

Taking the loosest interpretation of this definition would allow that the PCs on a LAN, can together be considered a form of cluster. In general though, clusters can be distinguished from distributed systems which tend to be loosely connected and use slow

interconnects (not always). Furthermore nodes in a cluster have a strong sense of membership which generally nodes in a distributed system do not.

Other characteristics of a cluster include:

- it can be managed as a single system, reflecting the tight coupling
- services are cluster wide
- it can tolerate component failure
- components can be added transparently to users

Types of Clusters

In general there are two broad categories of clusters in use today:

i) research machines comprising many compute nodes which are used for massively complex computations such as weather forecasting or N-body particle simulations.

ii) smaller >=2 clusters which are used in business as web servers, mail servers, database servers etc. These might be classified as commercial clusters and are popular as a preferred alternative to other forms of high availability technology such as data mirroring, server mirroring and fault tolerant systems.

The former type of machine is generally found in academic institutions or government installations and generally uses Linux or some other free Unix operating system distribution to run the machine. On the other hand the latter is found in business and frequently uses Windows NT as its operating system of choice. Both these types of system are described in further detail below.

Within these two broad categories, a further classification of clusters can be made on the type of use to which they are put and the type of service they are expected to provide:

Server farms

These are the oldest and simplest type of cluster whereby a cluster consists of a group of compute nodes which demand work from a central server. This is well suited to applications that require large amounts of processing and that need limited inter-node communication. In this cluster, if a node fails, the server merely assigns its work to an alternative compute node.

Failover clusters

The focus here is on availability rather than scalability. Typically there are two nodes, one that serves as a primary the other as a backup and together they aim to provide continuous service. This is the type of cluster often used by business for e-mail or web servers.

Coupled clusters

These are essentially a combination of the features of the above two. They work closely together and there is a considerable amount of inter-nodal communication –consequently high-speed inter-nodal interconnect is extremely important. Availability is a group responsibility so that failures can be handled gracefully by the cluster.

There are two types of coupled cluster, a shared nothing cluster and a shared memory cluster.

shared nothing cluster

> Each node has its own memory space. So the programming model must be smart enough to know what nodes are in the cluster and what processes are running on what nodes, so that recovery from failure can be accomplished.

shared memory cluster

> All nodes share the same address space and an application running on this type of cluster sees it as a single entity. Nodes communicate using the IEEE's Scalable Coherent Interface [HREF15].

Coupled clusters probably represent the future of this technology. They scale seamlessly, handle nearly any kind of application, use commodity hardware and offer high availability.

Cluster configurations

Various types of cluster configurations can be constructed [SHA97]. The ability to cluster nodes together can be implemented in hardware, in software or in a combination of both. In hardware, specialized interconnects facilitate cluster creation. Examples of this include various storage area networks.

Both hardware, as described above and the Operating System can combine to form the cluster. The IBM NetFinity or Dell Enterprise Server provide examples of this approach.

Components of the Operating System can communicate to form a cluster though there is a blurring of distinction between this and distributed Operating Systems. Systems that adopt this method include MSCS (Microsoft Scalable Cluster Server) and Solaris Enterprise Cluster, AIX clusters & HP_UX clusters.

Another type of cluster allows middleware running over the Operating System to implement the cluster functionality. The Beowulf class of clusters (see below) use MPI and PVM to achieve this.

Finally some applications have internal code that allows them to create clusters with other nodes running the same application, e.g. Oracle Parallel Server.

Cluster Forces

Supercomputing may be a small part of the computer industry, but yet is a segment worth $2.2billion [POL99]. Supercomputing applications involve extremely complex mathematical computations and thus require large amounts of computing power. Until recently this computing power was sourced from specially constructed machines that were characterized not alone by their massive speeds (now measured in TeraFlops) but

also by their relative cost. They were and remain very expensive and beyond the means of most organizations.

Undoubtedly the rapid accelerated growth in computational performance has supported the developing interest in clusters. In the last decade, a 3-order of magnitude increase in the number of devices per chip has been achieved [RID97]. A similar increase in chip speed has also been made. The much bigger economies of scale attainable in the PC market has resulted in a major realignment of cost/performance ratios between PCs and workstations. For example in the past four years PC microprocessors have had a performance rate increase of approximately 200% per annum as against 50% per annum for workstation microprocessors. This trend is even more pronounced when comparing PC performance against mainframes and supercomputers. Due to the smaller sales volumes of mainframe and supercomputer products, advances in these systems are amortized over a smaller number and cost/performance ratio vis a vis that of PCs have slipped considerably. It is this changing price performance ratio that is the major driving force behind the cluster revolution [HAL99].

Furthermore clusters scale well, both absolutely and incrementally. This allows organizations to achieve incremental growth and to match the growth of their supercomputing facility to the extent of their budgets.

Despite the enormous advances achieved, the requirement for ever increasing computational power remains. For example, the US Department of Energy's ASCI (Advanced Strategic Computing Initiative) has targeted a benchmark of 100 TeraFlops per second by the year 2005 [GRE97]. Indeed many of the national agencies in the US, DOE, NASA, DARPA, NSA have targeted Petaflop machines by 2010. These incredibly fast machines would be a thousand times faster than today's fastest machines.

Cluster Taxonomy

A taxononomy as given by Thomas Sterling of the NASA JPL [STE97], illustrates the different classes of cluster computing and provides a useful depiction of the place of clusters in the hierarchy of Parallel Computing.

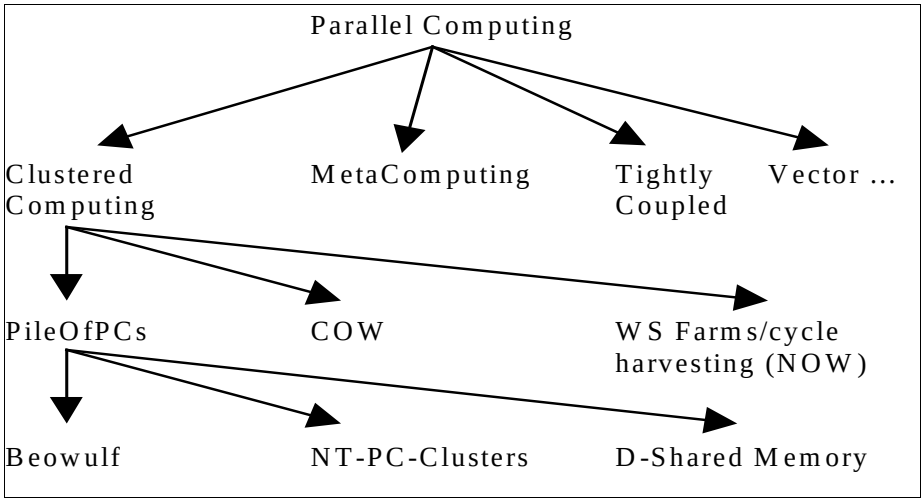

Figure 1: A Taxonomy of Parallel Computing

This taxonomical breakdown of cluster computing is discussed in detail and the differences between the different elements of the Clustered Computing branch are explained.

Pile Of PCs

The term, Pile Of PCs, is used to describe a loose ensemble of PCs working together on a single problem. It is similar in concept to both Cluster of Workstations (COW) and Network of Workstations (NOW) but emphasizes the following:

- Pile of PCs uses mass market components
- dedicated processors
- private system area network

Cluster of Workstations (COW)

A cluster of workstations is a group of workstations that can be formed from existing ‘‘idle” workstations and PCs overnight and on weekends. In principle, no cost is involved

and hence is very attractive. Experience has shown that they perform many parallel tasks effectively.

Network of Workstations (NOW)

The Network of Workstations [AND95] concept extends the COW concept by also using idle workstation cycles even during normal working hours. This was done in response to the finding that even during daylight hours in an electrical engineering lab, over 60% of workstations are available 100% of the time.

The idea of using idle resources over the network is not new but a number of recent technological advances has made it more feasible:

- high speed LAN interconnects such as ATM and Myrinet
- continued dramatic increases in workstation performance which means that the harvesting of idle workstation cycles is more worthwhile than ever.
- while processor speeds have experienced dramatic increases, disk technology advances have been made mainly in terms of capacity, which inevitably will lead to disk I/O being a retardant to overall performance increases. NOWs provide a way around this by offering a huge pool of memory as a disk cache.

Beowulf

Perhaps one of the more important recent developments in the field of cluster computing, Beowulf is a series of modifications to the Linux kernel and a message passing system, that allows scaling of computers (not SMP), connected via a standard networking medium [STE98]. Beowulf further refines the Pile of PC concept as defined above, by stressing the following:

- no custom components
- easy replication from multiple vendors
- scalable I/O
- freely available computing tools. To this end Linux is used as the operating system of choice on Beowulf systems.

Beowulf History

In 1994, two NASA researchers began investigating the use of commodity components to build a functional cluster supercomputer to address problems associated with the large data sets that are often encountered in Space science applications. They built a cluster computer consisting of 16 DX4 processors connected by 10Mbit/s Ethernet (The processors were too fast for a single Ethernet and Ethernet switches were still too expensive. So the Ethernet drivers for Linux were re-written to build a "channel bonded" Ethernet where the network traffic was striped across two or more Ethernets. As 100Mbit/s Ethernet and 100Mbit/s Ethernet switches have become cost effective, the need for channel bonding has diminished.). They called their machine Beowulf. The machine was an instant success and the idea of using COTS (Commodity Off The Shelf) base systems to satisfy specific computational requirements quickly spread through NASA and into the academic and research communities.

Advantages of a Beowulf

- no single vendor controls the product.
- since Beowulf systems are constructed from commodity components, this facilitates technology tracking – i.e. as new components become available they can be included in the system. Consequently, Beowulfs are very flexible.
- the use of free software with available source code allows system software to be customized for particular applications.

D-Shared memory Clusters and SMPs

Shared memory clusters have been mentioned previously as a type of coupled cluster. This type of cluster presents four advantages over single larger machines such as SMPs: Note that these advantages are shared by other classes of clusters [BRE97].

- absolute scalability, cluster can grow much larger than SMPs
- incremental scalability, clusters enable pay-as-you-go growth
- high availability, the failure of a node in the cluster does not cause the system to fail
- superior price/performance

Several companies sell hardware optimized for shared-memory clusters (SMCs). These machines simplify the porting of applications as they do not require re-coding. However there are two disadvantages [HREF3]:
Firstly, partial failures on an SMC can be a problem, whereby the failure of even one request typically causes the whole program to fail. It is possible to re-code the application to circumvent this problem but this defeats the purpose of using SMCs. Secondly, application performance varies by orders of magnitude depending on whether the data requested resides in local memory or cache or if it must be called from another node.

NT-PC
Clustering applications for clusters that use Windows NT as their operating system are still in their infancy as are the clusters themselves. Up to this point the clusters have been really just failover devices used in business to provide a degree of fault tolerance. To illustrate this point, Microsoft is preparing an update of its Windows clustering software with a goal of demonstrating clusters of up to four servers by spring 2000 [GAR99]. These systems have not been used to any great extent in the research community where Linux clusters have been predominant. The clustering tools for Windows NT are dealt with in the software section.

Cluster examples
Some examples of existing clusters are given below. Architectural features (size and cost) are used to illustrate the relative power and cost of these machines. The systems are being used in a wide range of applications.

Beowulf class
Firstly due to the impact of the Beowulf class of machines, some examples of this type of system are given.

Loki/Hygalc
As an example of a Beowulf class computer, these two machines are pretty representative of the genre. They are considered together as they were constructed by the same research group and differ primarily in their network topology. The cost of the machines in Autumn

1996 was $50,000-$60,000 but by Autumn 1997 because of continued dramatic reductions in PC component costs, this had fallen to about $30,000 - and continues to fall.

Loki is comprised of 16 Pentium Pro microprocessors connected by fast ethernet. The whole machine contains 2 Gbytes of memory and 50 Gbytes of disk. This configuration has attained sustained performance in excess of one Gigaflop [WAR97a].

The configuration of Hyglac is practically identical to that of Loki, the major difference being the choice of network cards and network switches. Sustained performance is approximately equivalent.

These two machines were joined at the Supercomputing ’96 conference for the first time and recorded sustained performance of 2.19 Gigaflops [WAR97b]. The cost of the combined system was $103k which gave a price/performance result of $47/Mflop.

Avalon

Avalon [WAR98] is an Alpha/Linux cluster that was the co-winner of the 1998 Gordon Bell Price/Performance prize. Built by the same team responsible for the Loki system and constructed entirely from commodity components and freely available software, Avalon used 70 DEC Alpha processors (the same processors used in the Cray T3E series) to achieve a 10 Gigaflop performance for a cost of $150k.

Stone SouperComputer

The Stone SouperComputer [HOF1] named after a children’s fictional story, is a development of the Oak Ridge National Laboratory and was their first attempt at building a Beowulf class computer. It is mentioned here as it exhibits the absolute flexibility of Beowulfs and probably represents the low-end extreme of what is possible with regard to building these machines.

Having grown tired of seeking funding to build a Beowulf machine, the proponents of the SouperComputer used surplus personal computers donated by individuals and other departments to build a parallel machine. They used public domain compilers and message passing libraries to have a functional system, built at virtually no cost. The system is already being used to do large scale computations on landscape analysis.

ASCI Red

This machine as winner of the 1997 Gordon Bell Performance prize and co-winner of the 1997 Gordon Bell Price/Performance prize, represents the cutting edge of clustered supercomputers. The machine is built entirely out of commodity components leading Intel to claim that " The system will be the first large-scale supercomputer to be built entirely of commodity, commercial off-the-shelf (C-COTS) components – the same processors, memory, disks and other modules found in millions of desktop computers and servers". The complete machine consists of 4,360 compute nodes, each containing two 200 MHz Pentium Pro processors. At the time of evaluation, only 3,400 nodes were available but these had a theoretical peak speed of 1.36 Teraflops [WAR97b].

Other Examples

There are many other examples of Beowulf class machines which has proved extremely popular as a means of achieving significant performance at affordable prices. A good starting point is the Beowulf Project home page [HREF4].

CAG Cluster

The Computer Architecture Group at Trinity College Dubin maintains a cluster system. Physically, it is divided into three sub-clusters: a cluster of 16 compute nodes each running Linux, a storage cluster running Windows NT and a cluster of workstation nodes. The sixteen compute nodes, CAGnode0 - CAGnode15, are ordinary Pentium II/450 PCs. Further details are available on the CAG Cluster web page at [HREF12].

The CAG cluster has been constructed primarily as a platform to conduct systems software experiments and systems performance analysis i.e. it is not used as a mini-supercomputer. Though the objectives of the CAG cluster may be different to those of a

normal cluster implementation, the fundamental problems of administering a cluster remain.

Windows NT example

The National Center for Supercomputing Applications (NCSA) at the University of Illinois created one of the world's largest Windows NT based clusters [HREF5]. The cluster consists of 128 dual processor Pentium II workstations and it is planned to upgrade this to 512 processors within the next year.

In terms of performance this cluster compared with a Linux based cluster system from the University of New Mexico. The tests were conducted on a 3D weather prediction model. It should be noted though that as the number of processors being used increased, the NT performance did not scale as well as the Linux system.

Cluster software

Two broad areas with regard to cluster software are examined now. A brief overview of the issues important in operating system software for clusters is followed by a more detailed examination of cluster administration tools.

Operating system software

Modern operating system (OS) designs have favoured modular microkernel approaches instead of the traditional monolithic kernel design. Microkernel architectures can be easier to tune for performance, reliability and customized for different uses. Improved performance is achieved by allowing unnecessary functionality to be eliminated. This has particular resonance for those with an interest in high-performance parallel computing

Linux

Linux is a popular choice of operating system in the field of cluster computing for a number of reasons:

- it is free
- it has shown itself to be a stable product
- full source code is available

- good near real-time scheduling
- Linux uses Intel I/O port protection
- can run DOS tools

Even though the trend of operating system design in recent years, as described above, has been toward microkernels, LINUX has a monolithic kernel. As of version 2.0 of LINUX for the Intel architecture, the kernel consisted of around 470,000 lines of C code and 8000 lines of Assembler [BEC98]. The assembler coding is principally used in emulating the maths co-processor, booting the system and controlling the hardware. Of the C code, approximately only 5% is concerned with what might be considered the core activities of the operating system in a microkernel context i.e. process and memory management. The remainder implements other aspects of the system such as file systems, device drivers and network drivers. However within this monolithic architecture LINUX does present many of the desirable features of microkernel design in its flexibility and customisability. The feature of the kernel design which facilitates this is loadable modules, which are object code that can be dynamically loaded and integrated fully with the kernel at run time.

Linux Modules

Modules are object code that can be dynamically loaded/unloaded and integrated fully with the kernel at run time. This means that the modules do not run as separate processes and do not affect the monolithic nature of the kernel. Mostly they are device drivers, pseudo-device drivers such as network drivers or file systems. The idea of loadable modules is not unique to LINUX and also exists in other UNIX implementations such as Solaris albeit in a different form [CAR98]. The functions modules export to the kernel are added to its symbol table and it is through this that they are invoked. When a module is loaded into the kernel, it runs with the same rights as the kernel and runs in system mode.

Advantages

- Small kernels can be built with other functions only being added as and when required and as mentioned this is of particular importance when trying to eke maximum performance from a cluster of machines.
- Easier to develop, test and debug modules
- If kernels that only differ slightly have to be built, it is much easier to build just one kernel and accommodate differences through loadable modules.

There is a slight memory and performance penalty associated with kernel modules. Loadable modules are slightly longer than if they were coded directly into the kernel and this and the extra data structures take a little more memory. There is also a level of indirection introduced that makes accesses of kernel resources slightly less efficient for modules. However this not withstanding, the advantages outlined above point to continued use of loadable modules. A detailed description of loadable modules is given in Appendix 1.

Extreme Linux

The Extreme Linux Society is a collaboration of industry, academic and government concerns whose goal is to make Linux and software that runs on Linux more effective for High Performance Computing. Extreme Linux [CHO98] is a Red Hat product that merges the clustering technologies of NASA's Beowulf project with Linux.

Administration software

A recurring theme in the literature is the need for software tools to more effectively and efficiently manage clusters. Brewer [BRE97] notes the problem that the available tools are relatively immature but that essentially they should be able to do two things. First, they should ensure that all machines are functionally equivalent. Second, the tools should be able to remotely monitor and configure all nodes – otherwise management entails logging in to each node, which doesn't scale well.

The bulk of the research effort, particularly in the Unix community has been towards developing tools that facilitate the execution of parallel programs on clusters. These tools are reviewed in [BAK95].

Development of tools to facilitate the normal system administration of a large number of machines has not been pursued to any great extent. The notable exception to this is the SMILE Cluster Management System, which was developed specifically to address this problem.

SMILE

SMILE is a 16 node Beowulf cluster based at the Department of Computer Engineering at Kasetsart University in Thailand [UTH98a]. The team responsible for building SMILE also developed a cluster system administration tool. By its own claims it facilitates the execution of a variety of tasks across a cluster. Repeated attempts to run and test this system were unsuccessful.

Motivation

The motivation behind the development of this package was, as has been noted, the difficulty in managing a cluster system due to the lack of a powerful resource monitoring and management tool.

Key features

The following is a list of features that the system is purported to support [UTH98b]:

- Statistics reporting, such as CPU load or memory usage, from any node or the whole cluster
- Remote login and remote command execution from any node
- Supports cluster wide parallel command execution
- Location transparent
- Shutdown or reboot any nodes
- Browse system configuration of any node from a single point
- Platform independent user interface using WWW

- APIs available for C, TCL/TK & Java to develop management applications

System Architecture

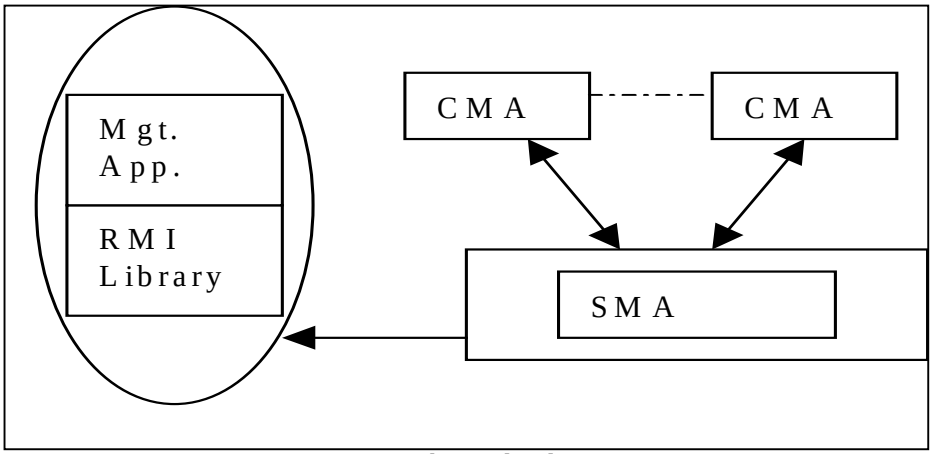

Figure 2: The Smile Cluster Management system

Each compute node has a CMA (Control and Monitoring Agent) that collects statistics on that node continuously and reports them to the central SMA (system management agent). Management information is presented to the user by a set of specific management applications. A set of APIs called the Resource Management Interface (RMI) is provided, to facilitate developers of management applications. These interfaces are available as a C library, Java class library and TCL/TK. This interface is then used to develop management applications and utilities.

System architecture justification

Using a centralised server increased the simplicity and efficiency of the system and simplified the design of the SMA. It also reduces concern about an information inconsistency and provides a uniform view of resources through an API.

However it does mean that there is a single point of failure. The intended solution to this particular problem is to have a backup SMA A centralised server does not scale well - the focus of the system is on a small to medium sized cluster (4 to 64 nodes).

Windows NT class cluster software

By way of comparison, a description of administration software available for Windows NT based clusters is given.

Microsoft

Microsoft provides clustering software called MSCS (Microsoft Scalable Cluster server) formerly called WolfPack in Windows NT [HREF6]. It currently provides for a simple two-node failover system. The benefits are higher availability, better manageability and scalability. MSCS can be remotely managed from any Windows NT workstation or NT Server. It provides the following services:

- a single system image
- audit status of all servers
- set up new applications, file shares, print queues etc.
- administer the recovery policies for applications and resources
- take applications offline, online and move from one server to another

Though it is probably easier to administer, it tends not to perform as well as the free Unix versions run on clusters, such as Linux, Free BSD & NetBSD [HREF7]. Further information on using Windows NT for parallel computation can be obtained at the Concurrent Systems Architecture Group at the University of Illinois who maintain the Resources for High Performance Computing on Windows NT web page [HREF8].

Compaq

The Compaq Intelligent Cluster Administrator is the first industry standard cluster utility providing web enabled cluster administration [HREF9]. It also provides a single point of control for Microsoft Windows NT MSCS clusters. The design decisions underlying the product were ease of use through an intuitive and consistent user interface and extensibility. The benefits claimed are for reduced system management costs and improved administrator efficiency.

Others

Many other companies including IBM, HP and Tandem provide clustering products. Information on these are available at these companies web sites [RUL97].

Summary

Clusters are being used extensively in both research and commercial installations. They provide an improved price performance and viable alternative to supercomputers and forms of high-availability such as data mirroring, server mirroring and fault tolerant machines. This illustrates the flexibility and adaptability of cluster systems and ensures ongoing interest in their development and application.

Linux is the dominant operating system with regard to cluster systems particularly in the realms of research and supercomputing. Even on commercial clusters, Linux is a popular choice because of its cost, flexibility and performance. A major research area, not just in reference to cluster computing but also with respect to OS design, is microkernels. These architectures can be made both small and flexible leading to improved performance for targeted applications. Linux implements a form of microkernel design through the use of modules, which can be loaded and unloaded on-demand and incorporated into the kernel.

A lack of administration tools for larger scale clusters renders their management more difficult. This is the problem being addressed in this project. In the next chapter, a detailed problem specification describes the goals and requirements of the system. It also details the constraints under which the system was designed and developed.

Chapter 3 Problem Specification

Introduction

Clearly the growing interest in cluster systems seems likely to persist given continued improvements in price-performance ratios of cluster components and the rising affordability of these systems. Concomitant with this interest is a large body of research devoted to these systems. However to date the majority of research in software tools for Unix based systems has been focussed on tools designed to assist programmers in parallel programming paradigm. There remains a need for tools to facilitate the daily administration of these systems.

Goals

Arising from this perceived deficit, the goals of this project are to design and implement an extensible system to ease the administration of a Linux cluster. The essence of such a system would allow command replication and configuration changes across multiple nodes. Extensibility would allow new commands to be easily added to the system. In general command output is textual however extensibility should allow the system to easily accommodate commands that return different types of output.

One of the features and main attractions of cluster systems is their scalability, which allows nodes to be added incrementally and accommodates a large number of nodes. The cluster administration tool should also scale well.

Requirements of the system

The requirements of the system were developed from a review of what other similar cluster administration tools offered. Also the requirements as specified by the cluster system manager were included. From this the general requirements of the system can be stated as:

Command replication and configuration changes across multiple nodes.

– the system should be capable of executing commands on one some or all nodes of the system. Accompanying this requirement is the need for users of the system to

be able to easily select which nodes are to participate in the execution of the command.

Variable substitution

- for maximum flexibility the system should support a scheme of variable substitution whereby variables are replaced by their actual values (similar to the Unix scheme of variable substitution).

A number of additional features were added as requirements to ensure a flexible system implementation. These are listed below:

Module management

- the importance of microkernel architectures with respect to high performance computing has been identified. Linux offers both manual and automatic loading as a means of supplying operating system customisability. The system should incorporate this functionality.

Formatted output from some menu commands

- output from commands should be formatted. This formatted output may be used for display or used to dynamically create a list of selectable items from which the user can select an items.

Shutdown & reboot of nodes and unloading of server software

- for changes in configuration to take effect it is often necessary to reboot the system on which the change has occurred. The cluster manager should enable any or all nodes in the cluster to be rebooted.

Output comparison between nodes

- this requirement is to handle scaling effectively. As the number of nodes increases it becomes increasingly infeasible for the output of all nodes to be displayed on screen. (and is of particular relevance to the situation in which nodal output is merely an acknowledgement of successful execution of the command). It is anticipated that a more acceptable solution is to contrast output from the nodes. If the output is the

same from more than one node, then only one copy is displayed with an indication of which nodes it is from.

System constraints

There were some constraints with regards to the design and implementation of the system. Both CORBA and Java were to be used in the design of the system.

CORBA

The Common Object Request Broker Architecture (CORBA) [HREF10] is an open distributed object computing infrastructure standardized by the Object Management Group (OMG). CORBA automates many common network programming tasks and eases the burden on the network programmer. It was pre-selected as the middleware to be used for this project. Furthermore a free (for non-commercial use), publicly available CORBA compliant ORB implementation, called ORBacus [HREF11], was used.

Java

Java was also a given of the system. It is a popular well-supported language and again there are many sound technical reasons to support this choice. The Blackdown [HREF14] implementation of the Java virtual machine was used.

Advantages of Java

- provides platform independence i.e. portable
- reduces development time - object oriented language which promotes reusable code
- brings new levels of interactivity to web pages.
- has in-built facilities to promote the use of distributed systems.

Summary

This chapter outlines the problem that this project is addressing namely the design and implementation of an extensible configuration and administration tool for Linux clusters. It details the goals and the requirements of the system and the constraints under which the system is to be developed. The next chapter describes the design of the system and highlights how the goals and requirements of the system are met.

Chapter 4 Cluster Manager Design

Introduction

The last chapter defined the goals of the project as designing and implementing an extensible configuration and administration tool for Linux clusters. The requirements of such a system were also detailed. This chapter presents the high level and detailed design of the essential features of the cluster management system. Specifically it highlights those design features of the system that will enable extensibility, scalability and flexibility.

Overview

Clusters

The basic architecture of a cluster system is outlined in Figure 3 below:

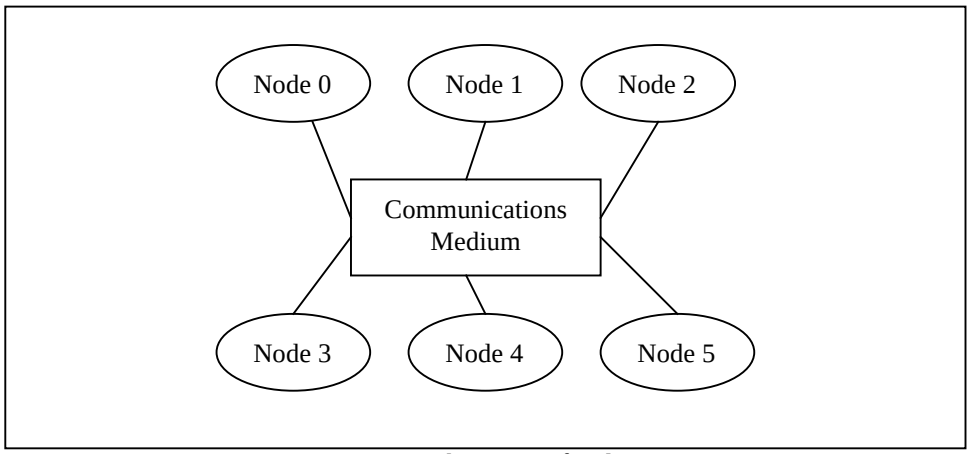

Figure 3: Basic architecture of a cluster system

Nodes communicate with one another through a shared communications medium with software to facilitate parallel programming. The Cluster Management System is designed to facilitate the administration of a cluster of Linux machines. The normal administrative duties associated with the administration of a single Linux machine must be replicated for each machine in the cluster. The Cluster Management System should reduce the amount of replication involved by automatically and transparently, replicating changes or command invocations on one machine to some or all other machines in the cluster. Generally changes will be reflected to all nodes so that a consistent image across the cluster is maintained.

Cluster System Manager Overview

An overview of the general architecture of the cluster system manager is illustrated in Figure 4 below. A client-server architecture forms the basis of the system. In this context, the client side of the system represents that part of the system which presents the user interface and through which the user issues command invocations. The server side is represented by the nodes of the system i.e. each node in the system acts as a server on which commands are executed.

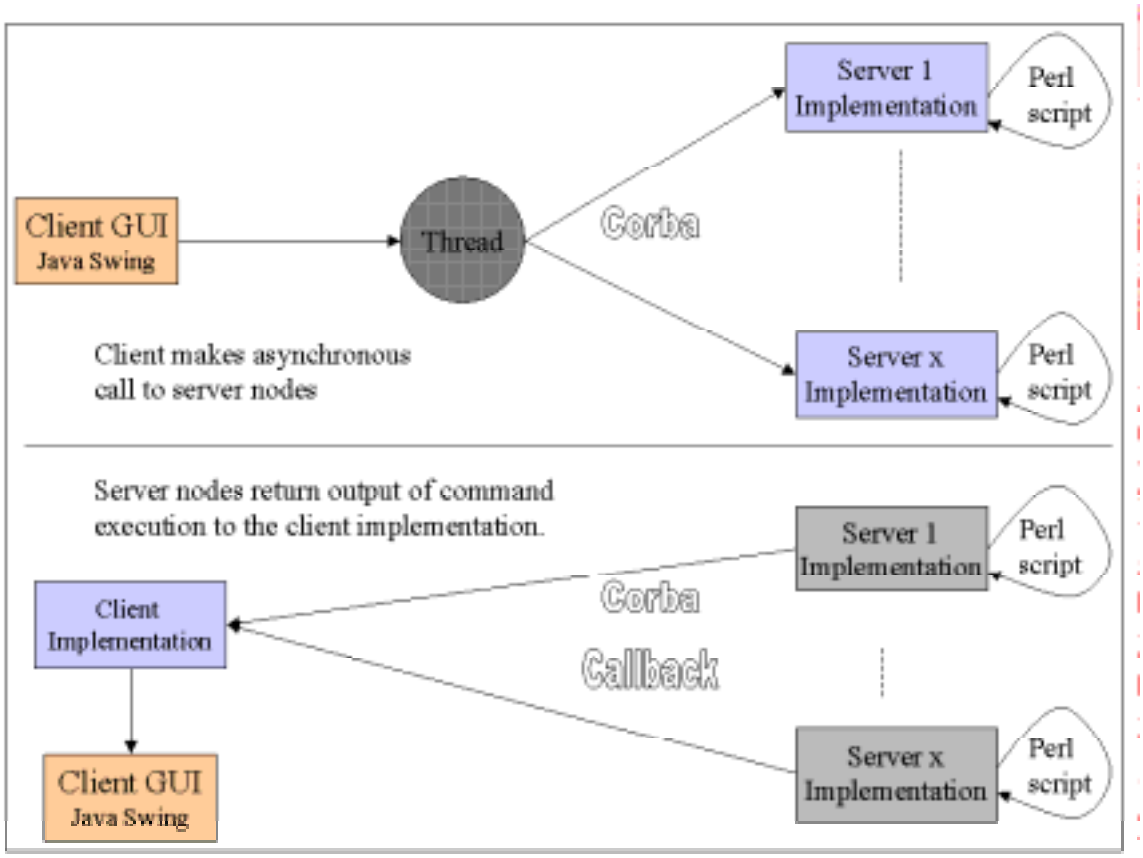

Figure 4 : Cluster System Manager Architecture Overview

Client Overview

In a cluster management system, the cluster manager (that is the client side presenting the user interface) should be capable of being loaded on any of the nodes in the cluster. The node on which it is loaded still remains as a functional part of the cluster and therefore commands which are invoked on all nodes will also be invoked on the cluster manager node.

The cluster manager essentially accepts two inputs from the user of the system. The first is the command that the user wants executed on the server node (plus any parameters to the command) and the second is the list of nodes on which the command should be executed. The system calls each node on the list of nodes and passes the specified command to it. The cluster manager operates asynchronously, so it does not wait for the command to execute before calling the next node on the list. As the nodes complete execution of the command, they initiate a callback to the cluster manager to return the response. The cluster manager takes the response and displays it on screen.

Communications Overview

The system uses Corba as its communication middleware. The user of the system chooses which nodes a particular command is to execute on and the cluster manager connects to these nodes and issues the command.

As mentioned, the cluster manager will operate asynchronously, i.e. it doesn't wait for an individual node to finish before invoking the command on another node. This improves the overall functionality of the system at the cost of dealing with replies being slightly more difficult. In single node command operation, or where the server node will reply immediately, synchronous operation of the cluster manager might be more appropriate. Certainly replies can be dealt with more efficiently.

On completion of their task, the nodes make a call back to the cluster manager, returning the output of the command execution. The client implementation accepts the response data from the node and displays it on the screen. In effect, this acts as a queuing mechanism whereby the output of only a single node can be displayed at a time, which is surely how output from a number of different sources should be displayed anyway.

Server Node Operation Overview

Each node in the cluster will run a cluster system management daemon, which can be launched on node start-up. This daemon merely listens for connections from the cluster manager through the CORBA interface. When a call from the cluster manager arrives, the specified command is executed and a callback to the cluster manager is initiated.

Cluster Manager Detailed Design

As mentioned, the cluster manager design follows a client-server architecture. There are three elements in this relationship:

- the client
- the server
- the client-server interface

A description of each of these elements is given below.

Client

The client side of the cluster manager system is that part of the system with which the user interacts. This is done through a graphical user interface. Through this, the user controls the system and issues command invocations to be run on the server nodes. Extensibility and customisability are enshrined in the system through the use of a user adaptable menus configuration file. Flexibility in command invocations is achieved using a user defined variables configuration file. Easy scalability is incorporated in the system by using a nodes configuration file in which new nodes can easily be added.

Client GUI

The requirement that the system be easy to use suggests the use of a graphical user interface to interact with the user of the system.

GUI Design

The GUI consists of a frame upon which other components may be drawn. It offers a menu driven interface through which the user can choose commands to be executed. The menu is constructed through parsing the menus configuration file as described later.

The GUI proper will contain two distinct areas:

1. a panel, that contains a list of selectable buttons that designate server nodes. From this the user can choose the nodes that are to participate in the execution of a particular command.

2. a panel that is used to display the output from the execution of commands on server nodes. In the most common case, two text areas will be visible on this panel, one on which normal output is presented, the other being used to display error messages.

The most common case caters for the situation of a standard Unix command being executed on the server nodes. Output from this command is plain text and can be displayed on a text area component. However, the design of the GUI allows for easy extensibility, so that other types of output can be displayed. For example if a statistics program generated graphical output then the system should be flexible enough to deal with this. To this end an API is provided to allow for the retrieval of the various display components. If a different form of output is required, the visible components can be hidden and the alternative component displayed.

Menus Generation

To give the system manager as much control as possible over the type of command selection facility offered by the system, a flexible implementation of menu generation has been implemented. A menus configuration file, is used to generate the menus that appear on the GUI screen. The client reads this configuration file on start-up. From this a system of menus and submenus can be generated. Each entry in the configuration file appears on a separate line and the meaning of each entry is shown in Table 1 below. Effectively this makes the entire system very flexible by allowing new utilities or commands to be added to the system without any reprogramming or recompiling.

Menu entry/ symbol	Meaning
*	The appearance of an asterisk denotes the start of a new menu. Note that the start of the menus configuration file implicitly assumes the presence of an asterisk, i.e. the first entry is assumed to be the name of a menu.
menu name	On the line after an asterisk, the name of the menu should appear.
menu selection mnemonic	The letter that can be used as keyboard shortcut to select

	this menu.
menu item name	After the menu name and menu selection mnemonic appear the menu items that form the menu. Each menu item consists of four entries beginning with the menu item name. This is the name of the menu item as it is to appear on the menu.
menu item mnemonic	The letter that can be used as keyboard shortcut to select this menu item.
command	The command that is to be executed by the server. This is blank if the menu command is implemented locally, for example the Clear TextArea command. In this case the command is implemented within the Java program.
parsing action	The parsing action choices are explained more fully in the Server section. Briefly, the following codes are used to denote these actions: PP : execute a Perl [HREF13] script with defined parsing NP : no script execution, return the output of the command execution as a string NPP : execute a Perl script with no parsing GP: a generic parsing of the output of the command execution
-	A dash indicates that a separator should be placed in the menu
#	A hash symbol is used to denote a new submenu. Following a hash the name of the submenu and its mnemonic should follow.

Table 1: menus configuration file format

The use of a configuration file allows the system manager to tailor and extend the menus GUI to the uses to which the system is being put. This in effect means that there are just 2 simple steps to extend the system to provide additional functionality:

1. make the command or utility available to users by adding it to the menus configuration file
2. ensure that the command or utility is accessible from the server node

Flexibility through Variable Text Substitution

The flexibility of the system is further enhanced through the implementation of a variable text substitution scheme. This scheme allows the user of the system to use a variable which will be replaced with associated text. Checking for variable text substitution is done at the following times:

i) on commands prior to their execution on server nodes.

ii) on file names prior to them being saved

iii) on the contents of files prior to their being saved

Two schemes are implemented for maximum flexibility and choice.

In the first scheme a simple direct variable substitution mechanism is used. In each of the circumstances listed above, a search through the text of the command, file name or file contents for the special variable $hostname is made. Each occurrence of it found is replaced with the actual hostname of the server node on which the action is taking place. So for example, the command "rm /dd/$hostname/file.txt" running on sever node cagnode00 will be replaced by the command "rm /dd/cagnode00/file.txt" which is the command that will be executed.

The previous scheme is merely a shortcut for what is probably a frequent substitution. In the second scheme, a variables configuration file is used to allow flexible variable substitution to be applied. The format of the file is the hostname of the server followed on the next line(s) by a variable identifier and the text that is to be substituted into the command, filename or file. The variable identifier is of the form $var-xxx where xxx is replaced by any three letter identifier for the variable. If the text $var-xxx is found in a command, file name or file, it is replaced with the associated text from the variables configuration file.

The variables configuration file should be read on each command execution. So effectively the configuration file can be amended between command executions so that different textual substitutions can take place.

Scalability

One of the features of Linux based cluster systems is the ability to seamlessly increase the number of nodes in the system. Consequently a cluster management system must also be able to handle extra nodes being added to the system. This is accomplished in this implementation through the use of a configuration file, which contains a list of all the nodes that form the cluster. Note that this is not a list of all the active nodes in the cluster but a list of all the nodes that may form the cluster at any time.

On client start-up, the client reads the nodes configuration file and uses this to draw part of the GUI. A selectable list of nodes is placed on the main GUI screen from which the user can choose which nodes are to participate in the execution of a command.

Client StartUp

On client startup, the client reads the list of nodes that form the cluster, from the nodes configuration file. This list is used by the client to attempt to make contact with each of the server nodes. It is a requirement of the system that all participating server nodes are started prior to the client. The server nodes once started must store their ORB references in a known location accessible by the client. ORBacus supports the storing of ORB references in a shared file system or at a Web location. As the proposed solution does not at this time incorporate a Web interface, it was decided to store ORB references in the shared file system of the cluster. The client retrieves the reference and uses it as a basis to assess whether a node is active or not. Once the client has built the list of nodes, it starts building the client GUI.

Node selection

The user of the system should be able to execute commands on one, some or all nodes of the cluster. The list of nodes that the client has built from the nodes configuration file is

used by the client GUI to offer a list of selectable nodes. From this the user can select which nodes are to execute a particular command.

Command Execution

The system is designed to provide both a menu-driven system through which various commands can be invoked and also an open specification using a dialog box through which free format text can be entered allowing any command or utility to be invoked.

On the menu system, there will be certain commands that are implemented locally and do not require a call to a server node. Otherwise when a command is chosen, either from the menu system or by entering it in the dialog box, the system establishes which nodes have been selected to execute this command.

A thread is spawned to handle communication and command invocation on all of the selected server nodes. The thread invokes the command on each server implementation in turn. It does not wait for a reply but continues on to the next server to invoke the command. In that sense it operates asynchronously.

When the server has completed the execution of the command, it initiates a callback on the client implementation. It passes the output of the command execution to the call. The client displays this on its GUI. An extension to the original design is needed if the output from the separate server nodes is to be compared. This revised design is shown in Figure 5 below.

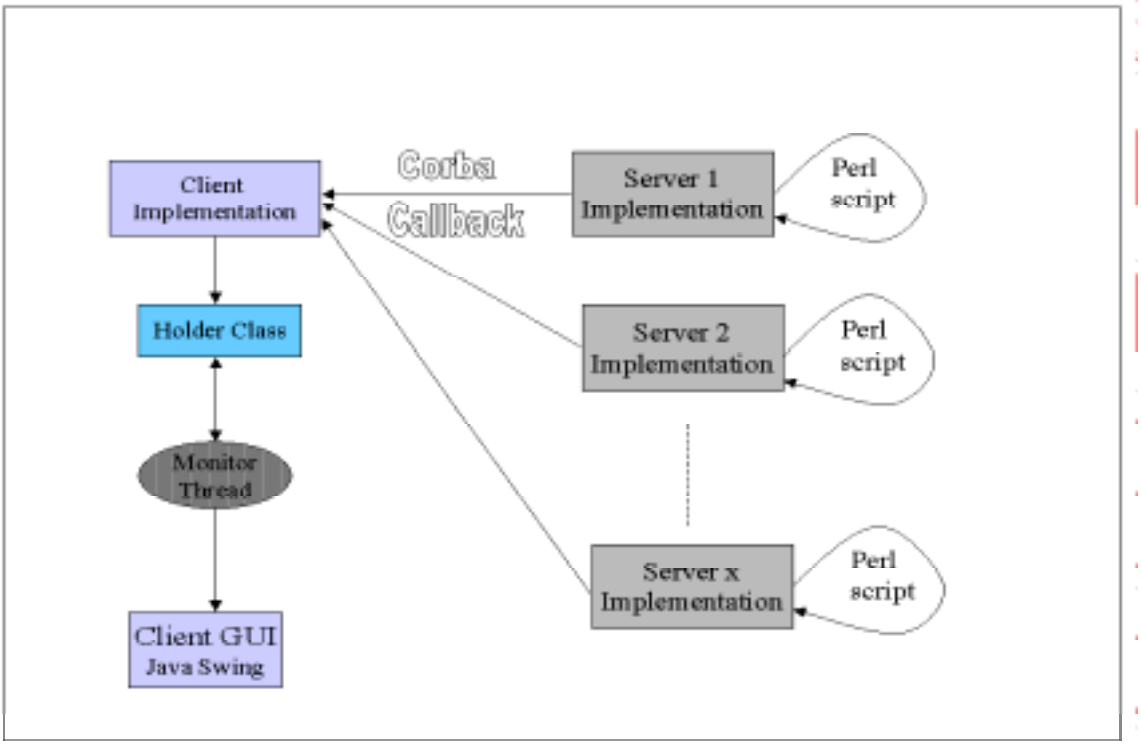

Figure 5: Revised design to handle output comparison

In this revised design, the client still calls the server nodes and in turn, the servers make a call back to the client implementation, on completion of the execution of the specified command. This is the same as the original design. However, now the client implementation stores the command output in a Holder class. A monitor thread can retrieve it from this class for normal command output. When output comparison is required, the output of the different servers in the holder class can be compared and displayed only if a difference exists.

Configuration File Management

The system supports the editing and saving of text files. This feature is useful in that changes to configuration files can easily be accomplished particularly with the flexibility offered by the variable text substitution. Both the editing and saving commands, are available as menu options.

Editing a file

When this option is selected, it begins the process of opening the file for display on the GUI text area. This option represents a special case of command execution in that the command should only be executed on one node, i.e. it is not suspected that the user might want to edit more than one file simultaneously. Therefore the editing of files results in the named file being opened on the local node only. It is intended that the contents of the text area be saved prior to editing. These can then be restored once the editing session is complete.

When the option is selected, a dialog box is opened through which the user enters the name of the file to be edited. If the file exists, it is opened and its contents are displayed in the text area. The file does not remain open and changes are not interactively written to the file.

Saving a file

This option can be selected at any time. The result of choosing this option is that the contents of the text area of the client GUI are saved to the named file. This means that not only can file edits be saved but also normal text area contents so log files of sessions can be constructed. It is worth noting that though the 'Edit File' option is implemented locally, this option is effected on all selected server nodes, that is, the file is saved in the specified location on all selected nodes.

On choosing the option, a dialog box is displayed that requests the filename (including pathname) under which the file is to be stored. The contents of the text area are saved to the file.

Client - Server Interface

The client-server interface describes the interface between the cluster manager and the nodes of the cluster. It outlines how communication occurs between them and what kind of responses the nodes return to the cluster manager.

Communication between nodes & cluster manager

Communication between the cluster manager and nodes occurs using CORBA. It is envisaged that each node in the cluster will run a cluster system management daemon, which can be launched on node start-up. This daemon merely listens for connections from the cluster manager through the CORBA interface. When a call from the cluster manager arrives, the specified command is executed. The client thread does not wait for the response from the server node. Instead the server initiates a callback to the client with the output from the command execution. The list of nodes on which the command is to be executed is selected through the cluster manager GUI.

Server

The server side of the cluster manager system is that part of the system that runs on each node of the cluster as a daemon. The server implements a number of functions that the client invokes. However the essence of the server can be summarised as a resource which executes Unix commands and in most cases, formats the output of the command execution for return to the client.

Command Execution Responses

Dependent on the command that is executed on a server node, the output that is produced is potentially different. These different types of output have been classified into the following broad categories of response in Table 2 below.

Type	**Explanation**
Successful Status Response	This category is for commands that have executed successfully and from which there is no output, i.e. from a command line invocation, successful execution would result in a return to the command prompt. An example of this command is a successful **mount** command, e.g. **mount /export**
Failure Status Response	This is for commands that have generated an error in their execution. The error is reported on STDERR.
Display only response	This response will occur for commands whose result is a display type output, for example the **ls**, or **df** commands.

	Display type responses will require that the output form each node is displayed one node at a time.
Display and Act response	This is an enhancement to the Display type outlined above. In this type, the response is parsed and selected elements of the response can be chosen for further manipulation. As an example, if the load module option is chosen off the cluster manager GUI, this results in the **modprobe –l** command being executed, and a list of the available modules that can be loaded being displayed in a dialog box. From this list the user is able to select the module to load by the **insmod** command.
No response expected	This category of response is for programs or command executions from which no response is needed or expected. This feature allows other utilities to be incorporated into the system.

Table 2 : Output types from command executions

Command Execution

To manage each of the potential responses and to manipulate the output from commands a high degree of flexibility is in-built into the system. This takes the form of an indication from the client to the server implementation of the action that the server is to take. It is envisaged that all command invocations will occur from within a Java runtime object. The different types of action that the server can take and the client codes to achieve this are described below.

- Execution of the command through a Perl script, with encoding of command output by the Perl script, for subsequent parsing by the server node. Perl is an excellent tool for text formatting and manipulation and is used to format or re-structure the output of certain commands.
- Execution of the command through a Perl script, with no encoding of command output but subsequent generic parsing. Generic parsing merely means that the parsing is based on the fact that output from a lot of commands consists of columns and rows. So it can be easily parsed using newlines and spaces as separators.

- Execution of the command through a Perl script, with no encoding of command output and no parsing subsequently. This is for commands for which no script has been written and for which generic parsing is unsuitable.
- Execution of the command directly through the Runtime object, with no encoding of command output no parsing subsequently. Certain programs will not run within a Perl script, for example other Perl scripts.
- Execution of the command directly through the Runtime object, with no output returned to the client.

Indirectly, the user can choose which of these options is to be used through the use of the menus configuration file. The fourth parameter to a menu item entry, consists of one of the following coded character sequences, PP, GP, NP, NPP, NRP. Each of these are explained in detail below.

PP

This encoding signals the server to execute the named Perl script with defined parsing. The command output is parsed and returned to the server. The server parses the command output and returns it to the client. Perl provides excellent text formatting abilities and is used in this system to manipulate the output from the UNIX command that has been executed. The alternative was to code the text formatting functions in Java, but Perl is optimised for text manipulation and is a more efficient implementation. Furthermore, Perl provides better manipulation of the standard and error input streams.

With all categories of response output, the result of Perl manipulation will be to return a coded string to the server node that invoked it. Within the string body, individual elements are separated by delimiters. On the server node the string is parsed once again using these delimiters and then returned to the client. The reason the parsing is done on the server nodes rather than the client is to facilitate scalability.

GP

Execute the command from within a Perl script with no manipulation. Command output is returned to the server unparsed. Generic parsing of this takes place at this point and the output is returned to the client.

Since the output of each command is unique, a separate Perl script to parse the output is needed for each command. However the output of many Unix commands consists of rows and columns. This kind of output can be generically parsed using newlines and spaces as separators. Generic parsing works quite well for the output of many UNIX commands whose output is inherently formatted this way.

NP

Execution of the command occurs through a Perl script but with no encoding or manipulation of command output. Furthermore the string that is returned to the server node is not parsed and is returned as is to the client.

NPP

The command is executed directly from within the Runtime object. The output of the command execution is returned as an unformatted string to the client.

Error messages

For those commands executed through a Perl script, the error stream is captured and redirected to the standard output stream. When an error occurs it is marked by the Perl script by pre-pending the word “ERROR:” to the command output. The string returned to the client begins with the word “ERROR:” which the client uses to direct the output onto the text area used for error messages. The string also contains notification of which server node generated the error and a copy of the original error message.

Modules

The potential of microkernels to provide greater performance has been described. Linux simulates microkernel features through the use of dynamically loadable modules. Clearly a cluster manager system must be able to manipulate this feature of the Linux operating system. The system must be able to allow the user to easily load and unload modules and this will be accomplished by providing GUI tools to facilitate this. The system should provide facilities to view:

- loaded modules and loadable modules
- a list of loaded modules from which they can be selected to be unloaded
- a list of loadable modules from which they can be selected to be loaded

The manipulation of Linux modules in the system could be accomplished in a number ways including:

- through the Java Native Interface interfacing directly with existing C code
- through a substantial re-write of existing C code into Java, retaining only direct system calls
- through a Runtime object calling Unix commands directly

The design does not suggest which option should be used. Instead each option will be investigated and the most suitable option for this system will be used.

Summary

The design of the cluster manger uses a client-server architecture. It includes features to ensure extensibility, flexibility and scalability. These features include the use of configuration files for menus generation, variable substitution and to represent the nodes of the system. Extensibility and flexibility are also incorporated into the design through the provision of an API for alternative component display and different command execution mechanisms. The next chapter describes how these design features were implemented and highlights any problems that were encountered during the implementation.

Chapter 5 Implementation

Introduction

The Cluster Management System is designed to facilitate the administration of a cluster of Linux machines. The design is based on a client-server architecture and incorporates extensibility, flexibility and scalability. The implementation of the design is described below. It details how the various design features were incorporated into the system. The client-server architecture - client, client-server interface, server - is used as a basis under which to describe the implementation. Initially the client GUI is described.

Client GUI

The implementation uses a graphical user interface to interact with the user of the system. Java Swing components which are part of the Java Foundation Classes (JFC) and are designed to supersede the AWT components of previous versions of the Java JDK, are used to build the GUI. With release JDK1.1.7, the version under which the system is developed, Swing is downloaded as a separately installable package. As and from JDK1.2, it is included as a core element of the JDK.

Three classes are used to construct the GUI :

Interface class

eTest class

Borders class

The GUI uses a frame upon which other components are added. A menus configuration file as described in the Design chapter is used to help build the menu bar. The implementation of this design is described later. The rest of the frame consists of the following components:

- a scrollable JPanel, that contains a list of selectable JCheckBoxes that designate server nodes and from which the user can choose the nodes that they want to participate in the execution of a particular command.
- a JPanel that contains a JSplitPane that in turn contains two scrollable JTextAreas. The larger JTextArea is used to display output from the successful execution of

command. The second and smaller JTextArea is used to display error messages in response to the unsuccessful execution of a command as shown in Figure 6 below:

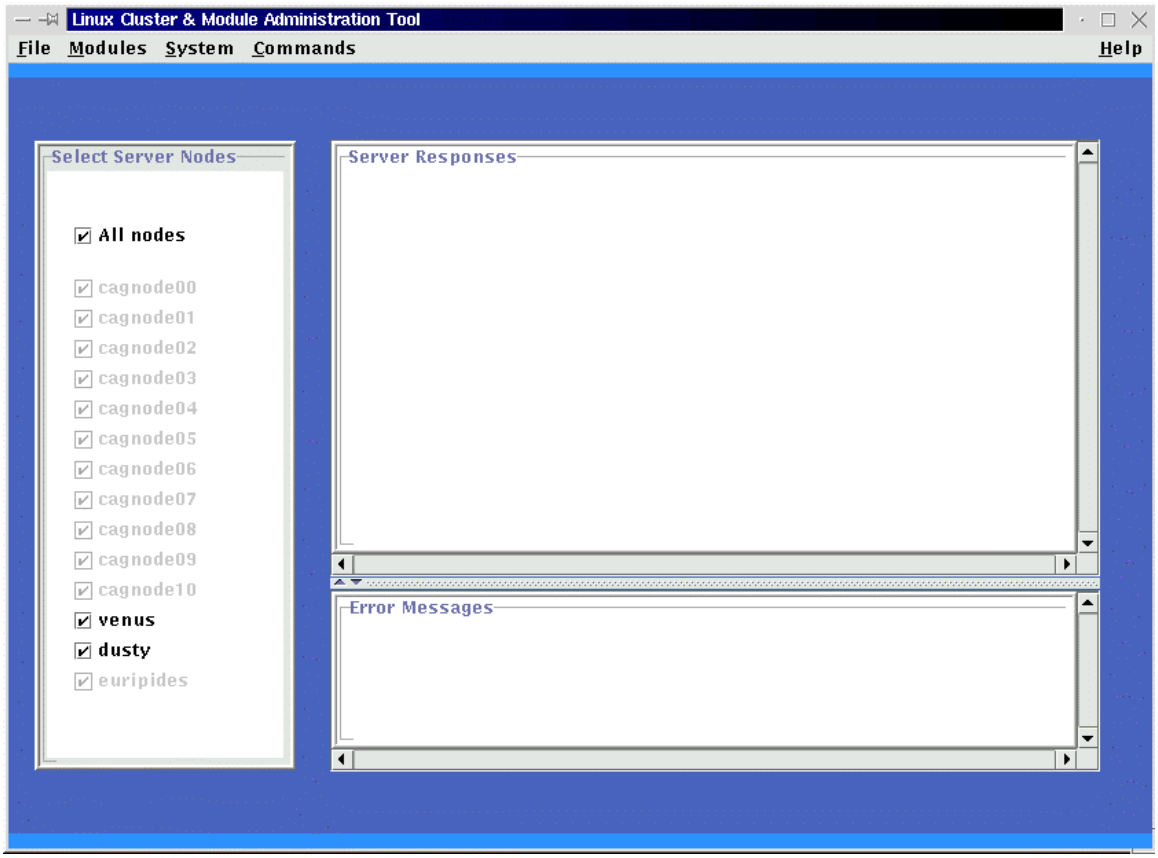

Figure 6: Screenshot of the Cluster Manager Interface

The JFrame part of the GUI and the construction of the menu bar is implemented in the *Interface* class. This class also implements the *ActionListener* for the menu commands. The *ActionListener* specifies what action is to be taken by the system when a menu command is chosen.

The *eTest* class constructs the other components of the GUI. This is a helper class that helps to produce the client GUI. In fact most of the GUI is created in this class. The class is a JPanel itself, created by extending JPanel and is instantiated by the *Interface* class. The design of the *eTest* class allows for easy extensibility. The JPanel that contains the two JTextAreas can be retrieved by any class through a call on the method *getJP()*. If a different form of output is required, the JTextAreas can be hidden and the alternative

component displayed. For example if tree type output is to be displayed to allow file system navigation, then a JTextArea is not the appropriate component upon which to display it. Instead the text areas can be hidden and a tree output displayed.

The class provides methods through which a reference to any particular component can be retrieved. This API provides in-built extensibility and allows the system to be easily extended in the future. The methods are also used by other classes in the system that need a reference to a GUI component. For example, a reference to the text areas is used by the *ExecThreads* class to display output on them.

A *Borders* class is used to create and set borders for the components of the GUI. Two classes are used to implement *ItemListeners* for the list of selectable nodes on the GUI. The *AllCheckListener* class is implements an *ItemListener* for the JcheckBox used to select or deselect all nodes. Class *CheckBoxListene*r implements an *ItemListener* for the JCheckBoxes of the individual nodes. Node selection is described in a later section.

Menus Generation

To make the system very extensible, a menus generation file is used to generate the menus on the GUI. The configuration file, currently hard coded to be "menus.cfg", is used to generate the menus that appear on the GUI screen. The client reads this configuration file on start-up. From this a system of menus and submenus can be generated. The design of the menus configuration file was explained in the Design chapter. Each entry is made on a new line and each menu item consists of four separate entries. A menu has two entries. This design was used in the implementation of the system and an example of its use illustrates the simplicity, extensibility and flexibility of the scheme.

The use of a configuration file allows the system manager to tailor and extend the menus GUI to the uses to which the system is being put. This provides two simple steps to extensibility:

i) make the command or utility available on the server nodes

ii) edit the menus configuration file and add the new menu item

Example

For example if network information relating to the routing table and interface table for each node is routinely required. A new menu is easily added to the system to handle this. Just edit the menus configuration file and add the following lines:

```
*                   // Start a new menu, denoted by an asterisk
Networks            // Name the menu Networks
N                   // The keyboard shortcut
Routing Table       // The name of the menu item that appears on the menu
R                   // The keyboard shortcut
netstat –r          // The Unix command to be executed on the server node
GP                  // The parsing action to be taken
Interface Table     // The name of the menu item that appears on the menu
I                   // The keyboard shortcut
netstat –I          // The Unix command to be executed on the server node
GP                  // The parsing action to be taken
```

The parsing action suggested here is generic parsing. This should work quite well as the output from these commands is in the form of columns of data for which generic parsing is particularly effective. However if this formatting action degrades the display the parsing action can be changed to NP for No parsing or a Perl script could be written to format the output. In this case the parsing action would be PP for Perl parse.

Variable Text Substitution

The flexibility of the system is enhanced through the implementation of a text substitution scheme. The design of this scheme is explained in the Design chapter. The *VariableParser* class implements the variable substitution scheme. As implemented this class is invoked to check the following instances for variable text substitution:

- command lines
- file names
- the contents of files

At execute time the variable is replaced by pre-defined text. The class merely loops through the contents of the command string, filename string or file contents string replacing occurrences of the special character sequences.

The variables configuration file, “variables.cfg”, is read on each occasion that the *VariablesParser* class is instantiated. This occurs prior to command execution and file saving. So effectively the configuration file can be amended between command executions so that different textual substitutions can take place. No limit is set on the number of variables that can be included in the variables configuration file to allow for maximum flexibility.

As a simple example of the use of this scheme, consider the task of saving a “/etc/hosts” file to each node in the cluster. An excerpt from a typical “/etc/hosts” file might appear as something like:

```
…..
127.0.0.1       localhost       localhost.localdomain
134.226.72.150  cagnode00.cs.tcd.ie     cagnode00
134.226.72.150   enya.dsg.cs.tcd.ie      enya
….
```

where cagnode00 is the node on which this “hosts” file resides and enya is the NFS server for the cluster. A similar “hosts” file is required on every node in the cluster. Conventionally this would require the system administrator to remotely login to each node and manually amend the hosts file on that node. However with this scheme, the task is greatly simplified. Just edit the “variables.cfg” file. Only two entries are needed for each node. The entries for cagnode01 are shown below:

cagnode01

$var-add = 134.226.72.151

$var-nam = cagnode01

Then edit the hosts file and enter the following.

127.0.0.1 localhost localhost.localdomain

$var-add $var-nam.cs.tcd.ie $var-nam

134.226.72.150 enya.dsg.cs.tcd.ie enya

Save this to each node in the cluster. Through variable substitution the variables in the hosts file will be replaced with the correct information. This greatly simplifies the task and is much quicker.

Scalability – nodes configuration file

To allow new nodes to be added to the cluster easily a nodes configuration file is used. The nodes configuration file, “nodes.cfg”, contains a list of all the nodes that form the cluster. Note that it is not a list of all the active nodes. The file is a static list of all the nodes that may form the cluster at any time.

On client startup, the client reads the nodes into an array. This array is used by the client to attempt to make contact with the server node. The array is also used by the client to draw part of the GUI. The list of nodes is placed in a scrollable list of JcheckBoxes on the main GUI screen from which the user can choose which nodes are to participate in the execution of a command.

Client StartUp

To start the client, the class *execClient* is called. This builds the list of nodes that form the cluster, from the nodes configuration file. The list is stored in an array that is used by the client to attempt to make contact with each of the server nodes. It is a requirement of the system that all participating server nodes are started prior to the client. The server nodes

once started, store their stringified ORB references in a known location in the shared file system. The file in which the server stores its ORB reference is simply the server hostname and a ".ref" extension. Therefore the client can use the array of node names to search for an ORB reference for that node. If present, the client retrieves and destringifies this reference. The *execClient* class then instantiates the *Interface* class with an array of the orb references - one for each active server and a null entry for inactive servers - and the array of server hostnames. There is a direct correlation between the two arrays. i.e. the same index in both arrays refers to the same server node. The array of hostnames is used by the *eTest* class to draw part of the GUI. The list of hosts is placed in a scrollable list on the main GUI screen from which the user can choose which nodes are to participate in the execution of a command. If the associated ORB reference in the orb references array is null - presumably through the server being inactive - that server node is marked as inoperable and its entry in the client GUI is disabled. Node selection and command replication are explained below.

The *Interface* class instantiates the *eTest* class to help building the client GUI as described previously.

Node selection

As explained above, the nodes array is used by the *eTest* class primarily for client GUI construction. A second array of booleans, called *selectedNodes*, containing one entry for each node is also constructed. Again there is a direct correlation between the two arrays. Initially each entry in this array is set to true. A value of true for a node in the selectedNodes array means that the command should be executed on that node. Conversely a false value means that the command should not be executed on that node. If the client has been unable to retrieve the ORB reference for a particular server node, then the entry for that node in the GUI list is disabled and consequently the value of the boolean in the *selectedNodes* array is ignored. The value for any server node in the *selectedNodes* array is changed by selecting or deselecting its checkbox. When a checkbox for a node is deselected, the *selectedNodes* value is set to false. It is set true again when the checkbox is selected again.

Whenever a command is to be executed, the *selectedNodes* array is checked to see which nodes are to participate in the execution of the command. Only those nodes whose entry is set to true are called to execute the command. In Figure 7 below, in the list of server nodes, only the nodes *venus* and *dusty* are enabled and both of these nodes are selected. The same command is executed on each node, save for variable text substitution potentially modifying the format as described previously.

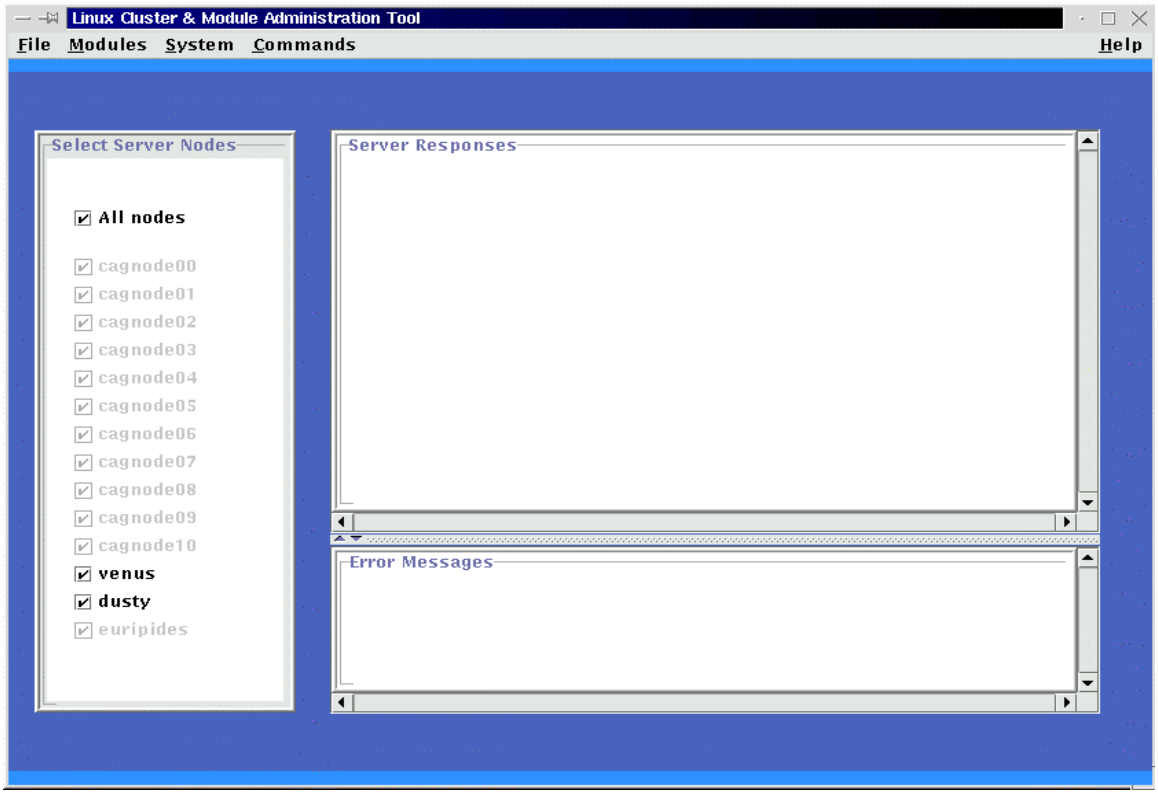

Figure 7: Node selection

The ‘All nodes’ check box is essentially a toggle box, used to select or deselect all nodes. It does not indicate whether all nodes should execute a particular command. This is only determined through examining the *selectedNodes* array.

Command Execution

The client GUI facilitates two ways by which a command may be executed. It supports a menu system through which various commands can be invoked. It also implements an open specification through which any command can be called. Commands are entered in a dialog box.

On the menu system, there are certain commands that are implemented locally and do not require a call to a server node. For example the “Clear Text Area” command is implemented locally on the client. Otherwise when a command is chosen, either from the menu system or by entering it in the dialog box, the system checks the *selectedNodes* array to see which nodes have been selected to execute this command.

For each node that is selected, a new type of Thread class is instantiated on the client, to handle communication with and command invocation on, the server node. There are two types of Thread class implemented on the client, *ExecThreads* & *SaveThreads*. Both are responsible for invoking the server implementation and in most cases waiting for the server reply.

Redesign

This is a departure from the original design, which had suggested callbacks for communication of server replies to the client. However the vast majority of Unix commands are catered for in this modified design. The use of callbacks is needed from streaming output from commands such as ping and it is suggested in the Analysis section of this thesis that callbacks be implemented as a future enhancement. This modified design however represents a more efficient handling of calls that execute immediately (which for the cluster manager tool represents the vast majority of commands). This was noted in the original design. Within the context of this project this new design supports the vast majority of needed commands and a more realisable implementation within the timeframe, at the expense of being able to handle streaming output. Figure 8 below shows the modified design for communication between the client and server.

In this redesign, for each node that is selected, a thread is spawned on the client to handle communication with and command invocation on, the server node. For most calls to the server the thread waits until a reply has been received. When a reply has been received, it is displayed on the client GUI. Normal responses are displayed in the main text area while error messages are displayed in the error text area.

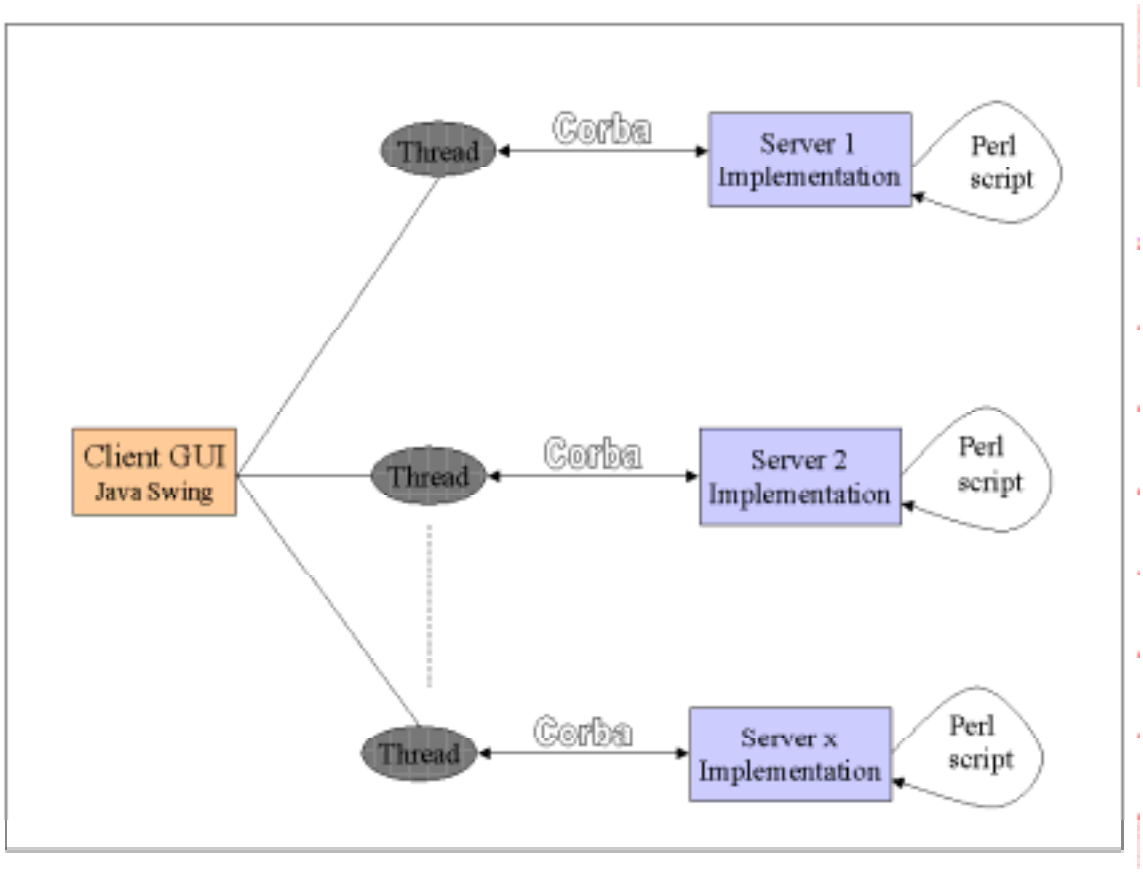

Figure 8 : Client –Server Communication Implementation

ExecThreads

The ExecThreads class is the most common Thread class that is instantiated. It takes as constructor parameters, a string representing the command that is to be executed by the server node and a string that represents the parsing action to be carried out. Using this latter string allows fine tuning of the server response. Parameters for references to the server's ORB, the two GUI text areas and the hostname of the server are also passed to the class via its constructor.

For each selected server node, a new *ExecThread* object is created to handle the call and reply to/from that server. For most calls to the server the thread waits until a reply has been received. It is blocking to that node in the sense that though further calls to that node can be made, no replies will be displayed until the original thread has completed. The execNoReply() call on the server does not expect a reply and is therefore non-blocking. When a reply has been received, the class uses the invokeLater() method of the *SwingUtilities* class to co-ordinate output from the various threads to the normal text area assuming successful command execution and to the error text area assuming an error message response from the server.

SwingUtilities invokeLater()

Using the invokeLater() method of the Swing utilities class effectively means that control of the text areas is passed to the event-dispatching thread. The invokeLater() method can be called from any thread and requests that the event-dispatching thread run certain code. The code must be put in the run method of a Runnable object and the Runnable object is then specified as the argument to invokeLater(). The event-dispatching thread then schedules for the code to run. An example of using invokeLater() to put a message on either of the text areas is given below:

```
Runnable paintGUI = new Runnable() {
        public void run() {
            if (reply.startsWith("ERROR"))
                jte.append(reply+"\n\n");
            else
                jta.append(reply+"\n\n");
        } };
        SwingUtilities.invokeLater(paintGUI);
```

SaveThreads

The SaveThreads class is the second type of Thread class implemented by the system. Its function is similar to that of the ExecThreads class but is used exclusively to create and/or save text files. The text to be saved is taken directly from the text area of the client

GUI. For each selected server node, a new SaveThread object is created to handle the call and reply to/from that server. The reply is merely an acknowledgement that the file has been saved successfully or an error message in the case where the file has not been saved. The class uses the invokeLater() method of the SwingUtilites class to co-ordinate output from the various threads to the text area.

Configuration File Manipulation

Configuration file management or just plain text file manipulation is possible through the editing and saving file options. The flexibility of these options is greatly enhanced through variable text substitution as described previously. Both commands are available as menu options.

'Edit File'

This option is one of the few that is implemented locally in that no call is made to a server node. Consequently the file is assumed to be local. If it is not an alternative way of editing it is described in the 'Cat File' section below.

On choosing 'Edit File', the *Interface* class stores the contents of the GUI textarea in a buffer. These can then be restored once the editing session is complete. A dialog box is then opened through the instantiation of the *EditDialog* class. This dialog box allows the user to enter the name of the file to be edited. The class also implements a WindowListener, PropertyChangeListener and an ActionListener to control the dialog box and detect when a filename has been input. When this happens, a check is done to ensure that the file exists on the local file system. If it does not an error dialog box is created. If the file does exist it is opened and its contents are displayed in the text area. The borders of the text area are changed to indicate that the user is now in editing mode. This is merely an exercise in cosmetics however as changes are not reflected back until the 'Save File' option has been chosen.

'Save File'

The 'Save File' option is selected after all changes to the file have been effected. It is worth noting that though the 'Edit File' option is implemented locally, this option is

effected on all selected server nodes, that is, the file is saved in the specified location on all selected nodes.

On choosing the option, a dialog box is displayed that requests the filename (including pathname) under which the file is to be stored. If this 'Save File' is called while in an editing session, then as a prompt the name of the file being edited is displayed in the dialog box. As long as the filename entered is not zero length it is assumed to be valid at this stage. Note though that invalid filenames will result in failure on the server nodes. Now for each selected node in the *selectedNodes* array, a new *SaveThread* object is created, which is used exclusively to create and/or save text files. The text to be saved is taken directly from the text area of the client GUI and is passed as a parameter to the *SaveThread* constructor along with the filename, references to the server node's ORB and references to the GUI text areas. The reply from the server node is merely an acknowledgement that the file has been saved successfully or an error message in the case where the file has not been saved.

'Cancel Edit'

The 'Cancel Edit' option is a related option that if chosen, as the name suggests, ends the editing session and replaces the current text area contents with the contents of the text area that were saved prior to editing. Since the file currently being edited, is not open, it does not need to be closed.

'Cat File'

For simplicity, when the 'Edit file' option is chosen the system searched only on the local node for the specified file. Sometimes it may be necessary to edit a file that is not present on the local node but is on one of the other server nodes. A way around this is to only select the server node on which the file resides for command execution. Then issue the command 'cat filename' on that node. This will result in the contents of the file being displayed on the text area. Changes can be made as normal and the contents of the text area saved back to file again.

Log Files

This facility is merely an extension of the file manipulation options described previously. The ‘Save File’ option merely takes the current contents of the text area and saves them in a file named by the user. No checking is done to see if an editing session is taking place and therefore it is possible to use this feature to save the details of your current session to a type of log file. This may be used subsequently to generate an executable script.

Client – Server Interface

The cluster manager-server node interface describes the interface between the cluster manager and the nodes of the cluster. It outlines how communication occurs between them, how the cluster manager invokes the specified command on each of the designated nodes and how the output from the execution of the command on the node is returned to the cluster manager.

Communication between nodes & cluster manager

The list of nodes on which the command is to be executed is selected through the cluster manager GUI. Communication between the cluster manager and nodes occurs using CORBA. It is envisaged that each node in the cluster will run a cluster system management daemon, which can be launched on node start-up. This daemon merely listens for connections from the cluster manager through the CORBA interface. [Ideally, there will also be a script to start up and closedown ORBs on all nodes as and when needed.] Currently however, the server nodes must be manually started, though they then remain operational until called to shut down.

When a call from the cluster manager arrives, the specified command is executed and the client thread associated with that call, in most cases, waits for the response from the server node. The exception to this rule is where no response is expected and the command is merely executed on the server node.

Server

Each node in the cluster will run a cluster system management daemon, which can be launched on node start up. This daemon merely listens for connections from the cluster manager through the CORBA interface.

Server Operation

When a call from the client arrives, an attempt is made to execute the specified command. The server currently supports different types of command execution that affect the response that the client will receive. Indirectly, the user can choose which of these options is to be used through the use of the menus configuration file. Recall that the fourth parameter to a menu item entry, consisted of a coded character sequence that the server interpreted to define what action to take. The implementation of these is explained below.

PP - Perl Script

Execute a Perl script with defined parsing, command output is parsed and returned as an encoded string to the Java Runtime object. On the server the string is parsed once again and returned to the client as a serialized object of type *DisplayClass*.

Perl is used to manipulate the output from the UNIX command that has been executed. With all categories of response output, the result of Perl manipulation will be to return a coded string to the Runtime object on the server node that invoked it. For example, if the response is a successful execution that results in no output from the UNIX command, then the Perl script generates a response output of the form “Successful command execution on server node xx”.

Within the string body, individual elements are separated by delimiters. The delimiter “|” is used to separate individual column elements, while the delimiter “¬” is used to separate columns. On the server node the string is parsed once again and returned to the client as a serialized object of type *DisplayClass*. This class acts as holder class for the output from command execution on the server nodes. It provides a number of *set* methods whereby information from the server can be stored. The *get* methods of this class are used by the

client implementation to retrieve the information. Note the class implements the Serializable interface as this class is first serialized to a byte array before being returned to the client. The command output is stored in a 2-dimensional array. There is also a header that is included for situations where the return from the server is empty.

As an example of this sequence, consider the output of the **mount** command on a server node. The encoded command response from the Perl script following the execution of this command on the server node might appear as shown below with individual column elements separated by a "|" and rows by a "¬".

/dev/hda2|none|/dev/hda8|/dev/hda6|/dev/hda3|/dev/hda7|none|automount(pid623)|automount(pid635)|clare:/export/home1/fogartjd|//WILDE/FOGARTJD¬/|/proc|/export|/tmp|/usr|/var|/dev/pts|/misc|/home|/home/fogartjd|/smb_mnt/wilde¬ext2|proc|ext2|ext2|ext2|ext2|devpts|autofs|autofs|nfs|smbfs

The server node parses this output and saves it to the *DisplayClass*. The client thread responsible for communication with the server node, receives the serialized *DisplayClass* object. It de-serializes the object and retrieves the command output information and displays it on the text area in the form shown below in Figure 9:

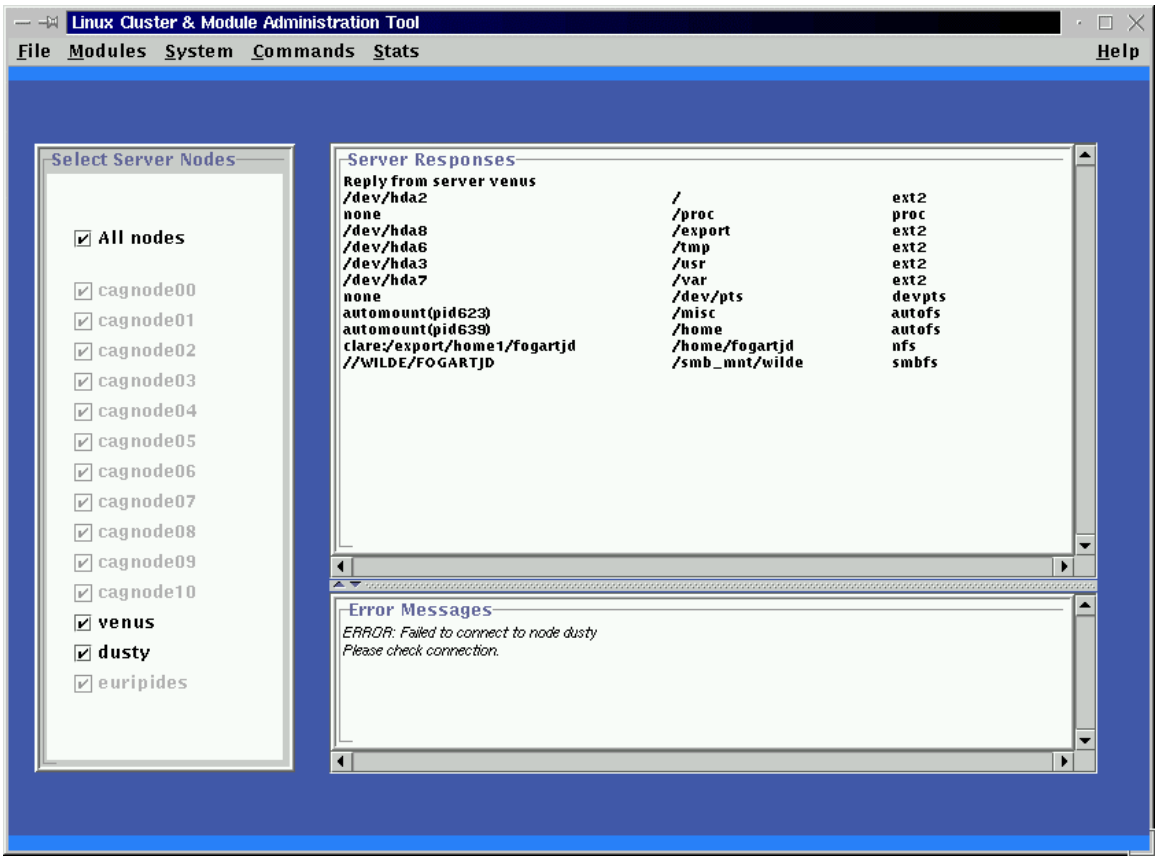

Figure 9 : Output of the 'mount' command

GP - Generic Parsing

The command is executed within a Perl script with no manipulation. Command output is returned to the Runtime object as an unparsed string. Generic parsing of the string takes place at this point and the output is returned to the client as a serialized *DisplayClass* object.

This mechanics of generic parsing are quite similar to those of the previous case. However in this case, no encoding of the command output occurs in the Perl script. The string is returned with no encoding to the Runtime object. This generically parses the string using newlines and spaces as separators. Generic parsing works quite well for the output of many UNIX commands whose output is inherently formatted this way. The parsed elements of the string are stored in the *DisplayClass* which is returned to the client for display.

NP - Perl but no parsing
Execute the command from within a Perl script, with no manipulation. Command output is returned to the Runtime object as an unchanged, unformatted string. Furthermore the string that is returned to the Runtime object is not parsed and is returned as is to the client. The exception to this last statement occurs in the case of a successful status response where the string is empty. In this case, a message to indicate the successful execution of the command and the node on which the command executed, is included in the string.

The reason both NP and GP are executed from within a Perl script is to allow easy capture of the error stream. Attempting to do this from within the Runtime object can result in blocking of either of the Input Streams, i.e. the normal InputStream or the Error Stream, unless precautions are taken to avoid this. These were considered overly complex to be implemented at this stage. However certain programs cannot be invoked from within a Perl script, e.g. another Perl script, and therefore NPP below caters for this.

NPP - No Perl, no parsing
No script execution occurs, the command is executed directly from within the Runtime object. The output of the command execution is returned as an unformatted string to the client except in the circumstances as outlined in the previous section i.e. in the case of a successful status response where the string is empty.

NRP - No reply.
This is an additional code to the original design. It was needed once the use of callbacks was not being implemented. In the revised design the calling thread would hang waiting for a reply. In this scheme no reply is expected or waited for, from the execution of the command. Execution of the command occurs directly through the Runtime object. This is used for utilities, for example *xosview*, which do not return a reply to the cluster manager. No message indicating success or otherwise of the command execution is returned to the client.

Error messages
For those commands executed through a Perl script, the error stream is captured and redirected to the standard output stream. When an error occurs it is marked by the Perl script by pre-pending the word “ERROR:” to the command output. The string returned to the client begins with the word “ERROR:” which the client uses to direct the output onto the text area used for error messages. The string also contains notification of which server node generated the error and a copy of the original error message.

Menus Implemented

The system facilitates easy manipulation of the menus displayed by the client GUI through editing of the menus configuration file. As implemented the menus listed below are displayed. Some of the options on these menus have already been explained in the context of the implementation of that feature of the system.

File
This menu has the following menu items:

- Edit File
- Cancel Edit
- Save File
- View mounted file systems
- Mount a file system
- Unmount a file system
- Exit Client

This last option unloads the client software and closes the client GUI. The servers remain active.

System
Any node in the cluster can be shutdown, rebooted or have its server daemon shut down.

Shutdown server daemon

When a call to shut down a server daemon is made, again the first action taken is to check the *selectedNodes* array to check on which nodes the command is to be executed. For

each node that is selected, a call is made to that server's CORBA implementation to deactivate the BOA_impl. The client also deletes the stringified ORB reference file that the server and client use to communicate. This is necessary so that when the client is restarted it does not use a reference to an ORB that is potentially out of date if the server had not been restarted.

ShutDown / Reboot

Arbitrary nodes can be shutdown or rebooted. Effectively this just means running the UNIX shutdown or reboot command on the node. The effect of this, in terms of the cluster manager system is to unload the server daemon, though the ORB reference is not deleted. As mentioned previously, when the node reboots the server daemon does not automatically restart but must be manually invoked. Furthermore the client does not automatically detect the newly booted server as it checks for active servers only at the start up of its own program. Hence for the client to be made aware of the rebooted server node, it is necessary to reboot the client program also.

Modules

The manipulation of modules in the system could be accomplished in a number ways:

- through the Java Native Interface (JNI) interfacing directly with existing C code
- through a substantial re-write of existing C code into Java, retaining only direct system calls
- through a Runtime object calling Unix commands directly

The JNI offers a set of standard interface functions through which JNI functions can be called from native method code to do such things as access and manipulate Java objects, release Java objects, create new objects, call Java methods, and so on. Also Java methods can call various native methods implemented in C or C++.

This approach and the second option represented the initial research direction. A number of "proof of concept" experiments were successfully conducted. However a point was reached where further research in this direction would have represented further detailed

examination of Linux operating system configurability and precluded development of the cluster administration tool.

In assessing which of the options to use within the cluster, it was concluded that the UNIX commands that will be made available through the cluster management system are tightly coded in well-written C and it is unlikely that making direct system calls will significantly improve call times. Instead a runtime object is invoked to execute the UNIX command on the target node. This option is also used with regard to managing modules in the cluster.

Module Manipulation

Module manipulation in the system occurs in much the same way as execution of any other UNIX commands. The module related commands available through the menu interface of the system are:

- View loaded modules
- View loadable modules
- Load module
- Unload module

When the 'View loaded modules' option is chosen from the menu system, the UNIX command *lsmod* is executed on each node that is selected in a manner similar to that described previously in the **Command Execution** section. The 'View loadable modules' results in the UNIX command *modprobe –l* being executed.

For either of the two menu options, 'Load module' or 'Unload module' a special case pertains, as two commands are executed on the selected servers.

'Load Module'

This option results in the command *modprobe –l* being executed on all selected nodes. The client gathers the replies from each of the server nodes which consists of a list of the loadable modules on that server. From this it creates a unique list of modules that can be

loaded. [Note that the modules in this list are not necessarily loadable on every server. So if a particular module is chosen it may fail to load on some servers but this is handled in the code and results solely in an error message being displayed in the error text area. The alternative is to have a unique list of modules loadable by every server but it was felt that this scheme was too restrictive]. The list constructed by the client is shown in a dialog box (see Figure 10 below) created by the *ListDialog* class. The user chooses the module they want to load from this list and an attempt to load the module on each of the selected server nodes is made.

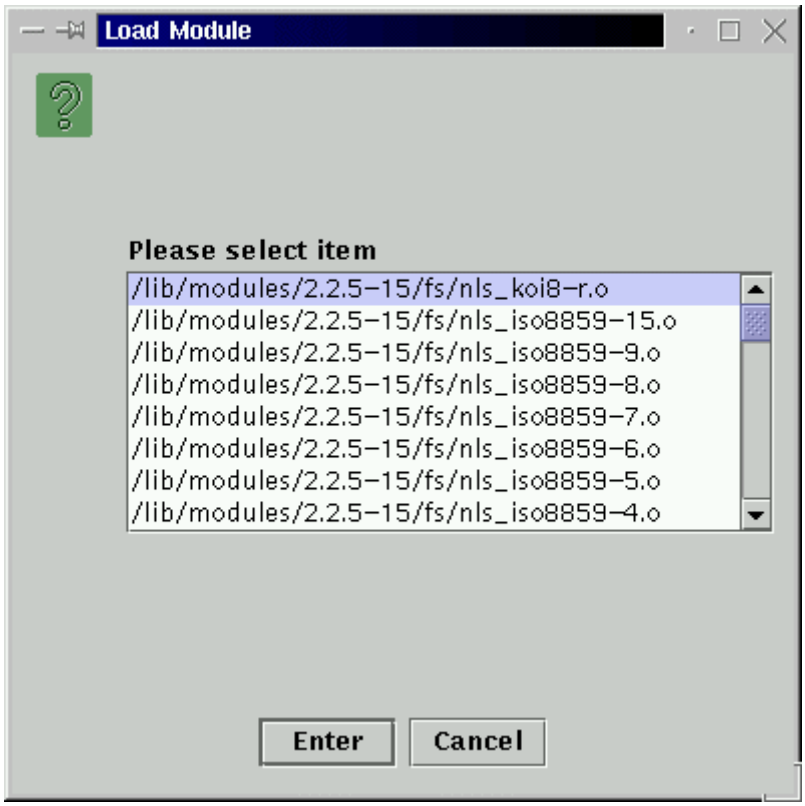

Figure 10 : The Load Module list box

‘Unload module’

This option results in execution of the Perl script *perlRmmodExec* on each of the selected nodes. This script executes the Unix command *lsmod*. The script then parses the output of that command to return just a list of the modules loaded on that server. This list is returned to the client, from which it creates a unique list of modules that can be unloaded.

[Note that the modules in this list cannot necessarily be unloaded on every server as the module is not necessarily loaded on every server. So if a particular module is chosen it may fail to unload on some servers but this is handled in the code and results solely in an error message being displayed in the error text area. The alternative is to have a unique list of loaded modules that can be unloaded by every server but it was felt that this scheme was too restrictive.]. The list constructed by the client is shown in a dialog box (see Figure 11 below) created by the *ListDialog* class.

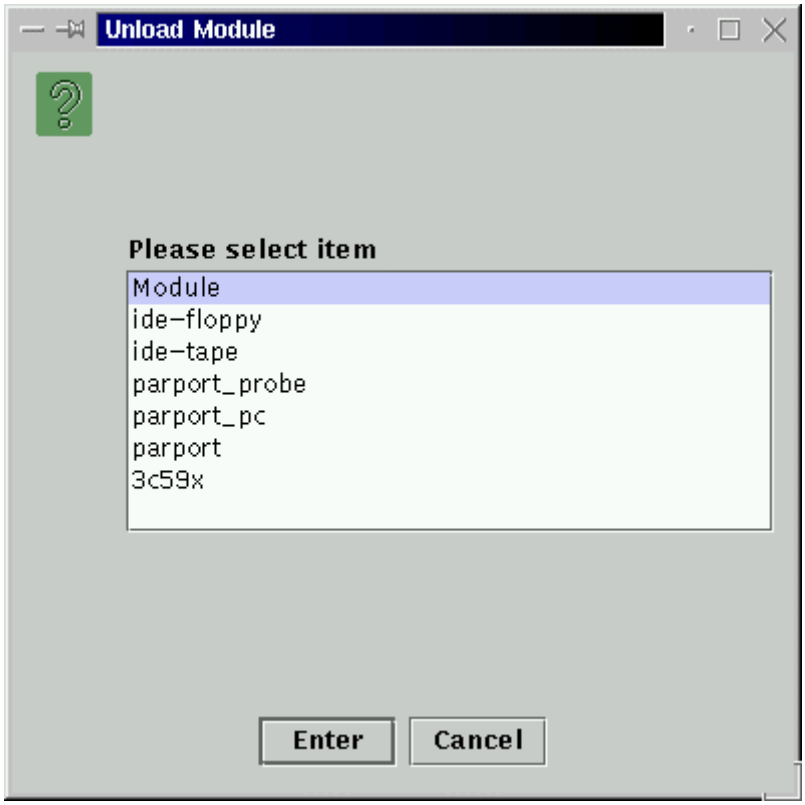

Figure 11 : The UnLoad Module list box

The user chooses the module they want to unload from this list and an attempt to unload the module on each of the selected server nodes is made.

The design of the *ListDialog* class means it can be used to display any list of selectable items thereby enhancing the extensibility of the system.

System Statistics

The system uses the *xosview* package to capture system statistics. *xosview* is an application originally developed for Linux that can be summarised as a graphical performance meter. A sample screenshot of an *xosview* window that was invoked from the Cluster Management system is shown in Figure 12. The choice of this package was for illustrative purposes only. The design of the menus generation facility makes the addition of any available utilities a trivial matter.

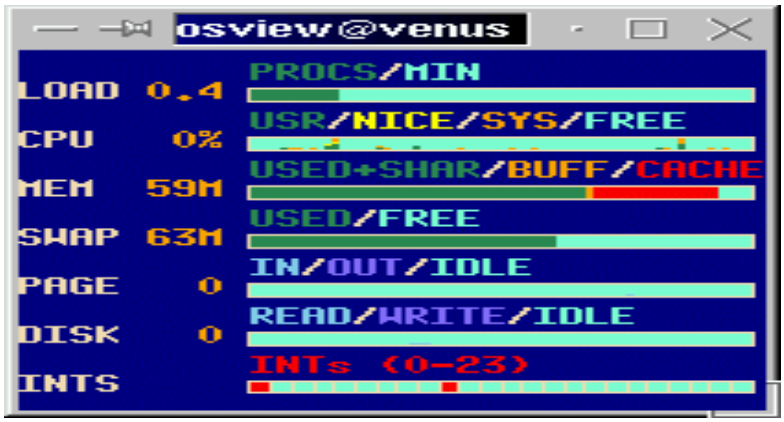

Figure 12 : The *xosview* utility

Implementation Problems

There were very few problems encountered during the implementation phase. The design was followed closely and did not present any insurmountable nor persistent difficulties. The main problem needed to be overcome during implementation appears to be a bug in the Java Swing classes which manifests intermittently. The effect is total unresponsiveness of all GUI components. The server nodes are unaffected. The only solution is to cancel the client program and reload it.

Summary

The system uses Java Swing classes to implement the GUI. The design has by and large been followed with the exception of callbacks and comparison of nodal output. Otherwise the design was followed closely and it was not necessary to deviate from it. The next chapter presents a review of the work carried out. An analysis of the system is given and suggestions for possible future work to further extend the capabilities of the system are elucidated.

Chapter 6 Analysis

Introduction

The previous chapter explained how the cluster management tool has been implemented. This chapter presents an analysis of the cluster management tool. It assesses whether the goals and requirements of the project have been met. The chapter also suggests what future work might be carried out to extend and improve the system. But first a review of the work that has been carried out is presented.

Review of Work

The realm of cluster computers is enjoying unprecedented popularity due to continued reductions in price combined with increasing performance which has made them affordable for many organisations that could not afford a traditional supercomputer. Naturally, that they scale well facilitating incremental growth also endears them to departments managing tight budgets.

Few software tools have been developed to administer Unix based clusters. The majority of tools facilitate the management of the parallel nature of these systems i.e. load balancing, parallel programming tools etc. Windows NT based clusters have a number of product offerings that offer administration tools.

This project addresses the lack of administration tools for Linux clusters which continues to render the management particularly of large scale clusters more difficult. The goals of the project were to design and implement an extensible system that would facilitate the configuration and administration of a Linux cluster. Extensibility would allow new commands to be easily added to the system. Also the system should easily accommodate commands that return different types of output. Cluster systems are inherently scalable and therefore the cluster administration tool should also scale well.

Analysis

The system is designed to facilitate the configuration and administration of a Linux cluster. In the absence of an administration tool to present a single system image, the

common alternative is to remotely login to each node and effect the requisite administrative tasks. This tool allows the administration and configuration of the entire cluster from a single node. Administration of the cluster is simplified by way of command replication across one, some or all nodes. Configuration of the cluster is made possible through the use of a flexible, variables substitution scheme, which allows common configuration files to reflect differences between nodes.

Goals

The main goals of the system were extensibility and scalability. These are analysed below. Additionally the extent to which other features of the system - flexibility, ease of use and portability – have been achieved is also examined.

Extensibility

The system is very extensible and exhibits such extensibility in a number of key areas.

Commands

Through the use of a menus configuration file the system is easily extended to include additional commands. This is simply accomplished by the system user with no reprogramming needed for the majority of commands. There are just two simple steps to extend the system interface and functionality:

i) make a command available by adding it to the menus configuration file

ii) ensure that the command or utility executable is accessible from the server node

GUI components

The standard GUI interface uses a panel with two text areas to display the textual output from commands. However an API has been provided to allow these text areas to be hidden and a different type of component displayed on the panel. The system thus exhibits extensibility in supporting different types of display components. The value of this extensible scheme is evident should the system be extended to support for example statistics monitoring that uses a graphical output. The API would allow the new graphical component to be easily added.

Scalability

Through the use of a configuration file for the nodes in the system, the system can scale well. New nodes are easily added through a single entry in the nodes configuration file. Ultimately however a problem is encountered in the logistical problem of trying to display output from a large number of nodes on a small screen of finite size. A proposed solution to this particular problem is presented in the future section below.

Scalability is also incorporated into the design of the system, through as much work as possible being distributed to the server nodes. All parsing of command output is conducted on the server side. Variable substitution is also effected on the server. This design provides a very scalable solution. The minimal amount of work is devolved to the client side, which merely dispatches command invocations to servers nodes and displays the resultant output.

Flexibility

Flexibility is exhibited in the very tractable scheme of variable substitution. The flexibility and scalability inherent in the design of this scheme makes configuration file management possible across a large number of nodes. The primary part of the scheme uses a variables configuration file that contains a list of variables, if needed, for each node in the cluster.

Flexibility is also evident in the cluster manager tool through the mechanism by which commands can be executed on the server nodes i.e. Perl scripts, generic parsing etc. This mechanism is easily set by the system user, in the menus configuration file and allows the system to gradually evolve. For example, a Perl script is not necessary to be able to add a new command to the system. It can be included in the system with one of the alternate command execution options set. If a Perl script is subsequently developed for that command, it is a trivial matter of altering the configuration file to reflect this.

Ease of use

The system uses the Java Swing classes to build the GUI that is presented to the user. Consistent screen views make the system intuitively easy to use. Node selection is easily achieved through a list of check boxes, so that it is readily apparent which nodes are selected to participate in the execution of a command. Almost all options are accessible through one or two clicks of the mouse button i.e the menus system retains a relatively flat structure and does not descend to confusing depths. The design for extensibility incorporates an API that ensures that new components retain the standard interface on the GUI. A standard Panel is used as the top-level container on which all other components should be drawn.

Portability

Through the use of Java and Corba the system is very portable and should work on all types of Unix clusters. Indeed there is no reason why the system should not work on more loosely connected classes of system such as distributed networks of computers.

Functional Requirements

The functional requirements of the system were:

Command replication across a number of nodes

The system allows easy selection of the nodes that are to participate in the execution of a command, on the client GUI.

Variable substitution

A flexible, scalable scheme of variable substitution has been implemented. This feature enhances the flexibility of other features of the system, particularly configuration file management.

Module management

It was decided that module management was well facilitated through the use of UNIX commands. Manipulating modules directly through the Java Native Interface would have added little in terms of functionality but represented an alternative research direction.

Modules can be loaded and unloaded across the cluster through the menu system of the administration tool.

Formatted output from some menu commands

Perl scripts are used to format the output from some menu commands. This is either used for display on the client GUI or used to construct a list of selectable items from which the user chooses an item.

Shutdown or reboot of nodes

One some or all nodes can be shutdown or rebooted. Also the server program can be unloaded.

Comparison of nodal output

This facility was not implemented. The implementation of this feature was predicated on the original design being followed whereby callbacks were used for communication between server and client. This would allow the client to store the output of the server in a holder class. When all servers had replied, the output from each could be compared and only displayed if it differed from the others. This feature would enhance the scalability of the server. As the number of servers grow, it is increasingly infeasible to display the output from each on the cluster manager screen. It is suggested as a future work item that this feature be implemented.

Comparison with other tools

SMILE CMS

In terms of functionality, both systems are quite similar. Both offer command replication across a number of nodes and the ability to shutdown or reboot any nodes. The SMILE package integrates system and performance monitoring whereas this tool outsources the function and uses the *xosview* package.

The main perceived advantage of the SMILE package is that it provides a Web interface and thus can be invoked on any computer with Web access. However it is intended that a web interface be developed for this tool.

The use of Java and CORBA in this tool brings many advantages. It is very portable and can be used on more loosely connected systems than a cluster. Java is extensively used on the Web and this is of greater significance should a web interface be developed for this tool. Java also promotes re-use and the design of this tool emphasises extensibility and flexibility. New features are easily incorporated into the system. The system provides a valuable framework upon which additional functionality can be added.

Despite repeated attempts, it was not possible to run the SMILE system. This would have enabled a more thorough comparison of the two systems.

Future Work

The cluster manager provides a useful tool for the administration of Linux clusters. However there are a number of significant enhancements that were not possible to implement within the timeframe of this project.

Provide a web interface

Clearly the implementation of a web interface enhances the overall flexibility of the system. Access to the system is possible from any system with web access. That the system is written in Java will simplify this process.

Extend the menus framework

The menus configuration file makes the system very extensible, allowing new commands to be easily added to the system without the need for re-programming or re-compiling. However though this mechanism can deal with the majority of commands, re-programming is needed for certain command types. For example those commands that produce lists, such as ‘Load Module’ and in effect use two commands to execute fully – one used to produce the list from which the user chooses an item and the second command to take the appropriate action with the command taken. This type of command

will require some programming because the menus configuration file as implemented cannot cater for this type of command.

Implement callbacks

The use of callbacks was in the original design and they are a required addition so that the system can accommodate streaming output. It is anticipated that for streaming output, only one server node will be active at a time. Additionally callbacks would provide a more flexible implementation for dealing with the output from many nodes, where it is desired that such output only be displayed if different from other nodes. It is anticipated that the response from the callback would be stored in a buffer class. When all replies had been received they could be compared for differences with each other. Only a single copy of equivalent output is displayed.

Automatic on-line node checking

Currently the system only checks for active server nodes when the client is started. Ideally the client should automatically check periodically to identify active server nodes. Alternatively the user should be able to invoke a command to do this.

Build a dynamic Perl parser

Formatting of commands requires a new Perl script for each command as each command produces a new type of output. To facilitate the overall customisability of the system by the user of the system, a metadata file can be used to describe the generic output of each command type. This obviates the need for a Perl script programmer and is in keeping with the overall design philosophy of the system, such as a user configurable menus configuration file.

Integrate with existing cluster tools

The cluster manager system certainly facilitates the administration of a Linux cluster. However there are many other tasks that the administrator of such a cluster must perform that are not available through this tool but exist as stand alone applications. Ideally all these tools would be integrated into a single system through which the various functions used to administer a cluster could be invoked.

Conclusions

The cluster manager is a first-step towards providing an integrated management tool for a Linux cluster. As currently implemented the tool will facilitate the administration of a cluster, providing facilities for command replication across multiple nodes and features to facilitate configuration management across a cluster. Within the development timeframe involved, much was achieved and the architectural groundwork for further development has been laid.

Personal Achievements

This project represented on a personal level an opportunity to develop skills and knowledge in the following areas:

Java in general and Java Swing & Java Native Interface in particular

operating system architectures and research in this area

CORBA

Linux

Without exception my skills in these areas are much improved.

Summary

This chapter presents an analysis of the cluster management tool. It reviews the work carried out and assesses whether the goals and requirements of the project have been met.

Appendix 1 Linux Modules

Introduction

This Appendix explains the concept of Linux modules and how they are used. It includes an depth examination of the part of the Linux kernel relating to module manipulation.

LINUX has a monolithic kernel. As of version 2.0 of LINUX for the Intel architecture, the kernel consisted of around 470,000 lines of C code and 8000 lines of Assembler [BEC98]. The assembler coding is principally used in emulating the maths co-processor, booting the system and controlling the hardware. Of the C code, approximately only 5% is concerned with what might be considered the core activities of the operating system in a microkernel context i.e. process and memory management. The remainder implements other aspects of the system such as file systems, device drivers and network drivers. As discussed below, within this monolithic architecture LINUX does present many of the desirable features of microkernel design in its flexibility and customisability. The feature of the kernel design which facilitates this is loadable modules, which are object code that can be dynamically loaded and integrated fully with the kernel at run time.

Linux Modules

Modules are object code that can be dynamically loaded/unloaded and integrated fully with the kernel at run time. This means that the modules do not run as separate processes and do not affect the monolithic nature of the kernel. Mostly they are device drivers, pseudo-device drivers such as network drivers or files systems. The idea of loadable modules is not unique to LINUX and also exists in other UNIX implementations such as Solaris albeit in a different form [CAR98]. The functions modules export to the kernel are added to its symbol table and it is through this that they are invoked. When a module is loaded into the kernel, it runs with the same rights as the kernel and runs in system mode.

Advantages

Small kernels can be built with other functions only being added as and when required

Easier to develop, test and debug modules

If kernels that only differ slightly have to be built, it is much easier to build just one kernel and accommodate differences through loadable modules.

There is a slight memory and performance penalty associated with kernel modules. Loadable modules are slightly longer than if they were coded directly into the kernel and this and the extra data structures take a little more memory. There is also a level of indirection introduced that makes accesses of kernel resources slightly less efficient for modules. However this not withstanding, the advantages outlined above point to continued use of loadable modules.

Module Elements

Elements of the operating system that are available as modules include [GEL1]:

Most file systems: minix, xiafs, msdos, umsdos, sysv, isofs, hpsmbfs, nfs
mid-level SCSI support
most low-level SCSI support
all SCSI high-level drivers: disk, tape CDROM, generic
most ethernet drivers
most CDROM drivers
many miscellaneous modules, such as: line printer, elf loader, java loader, cdrom interface, the serial interface.

Implementation

LINUX facilitates both manual and automatic loading of modules. The manual method is rather cumbersome and therefore automatic loading is the normal method of choice. Also it is possible to pass parameters to modules at load time.

Manual loading : Command line operation

Manual loading of modules can only be carried out by the super user, using the following three commands:

insmod	takes a module name as a parameter and loads the module
rmmod	takes a module name as a parameter and unloads the module
lsmod	displays the contents of the file */proc/modules* which is essentially a list of the loaded modules and the number of pages each module is consuming (4k on Intel machines).

Automatic loading : kerneld

Automatic loading or on demand loading allows modules to be loaded as and when required. They are also unloaded automatically when they are no longer needed. The operations are carried out by a kernel daemon, kerneld, which runs as a normal user process albeit with super user privileges. To install this feature, the CONFIG_KERNELD option must be enabled during configuration of the kernel. Also System V IPCs must be enabled when compilation is performed as the kernel and kerneld communicate using IPC via a message queue.

Compiling the kernel

The kernel must be configured in such a way that it can manage modules. If the option: *Enable loadable module support (Config Modules) [Y/N]* is not activated during the configuration process, then it will not be possible to use loadable modules. Furthermore it is necessary to identify in the configuration process any modules that are to be available as loadable modules. Compiled modules are identified as file objects in the library /lib/modules.

System calls specific to modules

LINUX provides three system calls for implementing modules: *create_module*, *init_module* and *delete_module*. These system calls are used by loadable module handling programs such as *insmod*. A further system call *get_kernel_syms*, is used by the user process to get the kernel's symbol table. The administration of modules in LINUX makes use of a list, *module_list*, in which all of the modules that are loaded are included.

As far as the kernel is concerned, modules are loaded in two steps corresponding to the system calls *create_module* and *init_module*. For the user process this divides into four phases:

Firstly the object code for the module is loaded into the address space of the user process, which resolves what references it can for relocating the module in kernel space.

The system call *create_module* is invoked to get the final address of the object module and then to reserve memory in kernel space for it. This is accomplished by entering a structure module in the list of modules and for which memory is allocated. The module has a status of *mod_uninitialized* (i.e it cannot be used at the moment). The return value gives the address to which the module will later be copied.

The general structure of the call is:

int create_module (char *module_name, unsigned long size)

The load address received by *create_module* is used to relocate the module. This still occurs in the user process address space. Unresolved references can be solved using the kernel symbols. The system call *get_kernel_syms* if passed NULL as a parameter returns the size of the symbol table. Using this to reserve memory, the process can then reissue the system call with other parameters to get the symbol table. This table must be updated when a new module is loaded to identify the address of each of the module's functions. To achieve the greatest possible degree of flexibility, the modules themselves can add symbols to the kernel's symbol table. This allows modules to use the functions of a module loaded earlier. This mechanism is known as module stacking. For example the VFAT file system requires the services of the FAT file system module. All the symbols exported by a module are collected in a separate symbol table.

The system call *init_module* is now called to move the module into kernel address space. It puts the code passed as a parameter into the list of modules (field addr which corresponds to the memory space allocated by means of *create_module*). It is only after this call that the module is usable. As its parameters it takes pointers to a structure *mod_routines* and the modules symbol table. The modules administration functions are entered in the *mod_routines* structure. The admnistration function *init()* is called now, to initialise the module, while *cleanup()* is called when the module is de-installed to free up resources.

The general structure of the call is:
int init_module (char *module_name, unsigned long size, unsigned
codesize, struct mod_routines *routines, struct
symbol_table *symtab)

The symbol table for the kernel is defined in the file *kernel/ksyms.c*. Each exported function has an entry in the table *symbol_table*. The name of the function or variable in each case is transferred to *symbol_table* by the macro X(). The kernels exported symbols are kept in pairs containing the symbol's name and its value, for example its address. The kernel's exported symbol table is held in the first module data structure in the list of modules maintained by the kernel and pointed at by the *module_list* pointer. Not every symbol in the kernel is exported to its modules.

Modules that have other modules dependent on them must maintain a list of references at the end of their symbol table and pointed at by their module data structure. In this way dependencies between modules can be used to prevent the untimely de-installation of a module, i.e. one on which other modules depend. A further mechanism to prevent this is the use of USE_COUNT, which is incremented every time a loaded module is opened and decremented every time it is closed. To prevent modules ever being removed it is possible to increment the USE_COUNT during the init() function.

The system call *delete_module* is used to unload modules. Two preconditions must exist for this to work, there must be no references to the module and the modules USE_COUNT must be zero. This call uses two functions: *get_mod_name* and *find_module* which retrieve the name of the module and a pointer to the structure of the module respectively. The call either unloads a single module if its name is specified, or if none is, then it scans the whole list of modules and tries to unload each unused module in turn.
After retrieving the name and the pointer to the module the state of the module is set to MOD_DELETED. A call to the unloading function *free_modules* is then carried out. This scans the list of all modules and deletes those whose state is MOD_DELETED. The

deletion of a module is performed in two stages. Firstly, if the references to the module are used by other modules, a call to *drop_refs* is performed. This function re-orders references between modules. Then *free_modules* deletes the table of symbols and erases the module code. Before the module is released its *cleanup()* function is called which releases any resources held by the module.
The general structure of the call is:
int delete_module (char *module_name)

The call *get_kernel_syms* can either return the number of symbols available in the kernel or one of the symbols themselves. If the table passed as a parameter is NULL, the system call will return the number of symbols that are actually available in the kernel. It does this by scanning the list of loaded modules and counting the number of symbols associated with each. This technique can be used to allocate memory space needed to create the table receiving the data. If a valid *kernel_sym* structure is passed as a parameter to the system call, all names and address of the different symbols contained in the loaded modules are copied into the table. Note that only modules in the MOD_RUNNING state are considered. The *value* field of *kernel_sym* contains the address in the kernel, of the structure that describes the module.
The general form of the call is:
int get_kernel_syms (struct kernel_sym *table)

The kernel daemon

The kernel daemon, *kerneld*, is a user level process, albeit with super user privileges that automatically loads and unloads modules without the user ever noticing it. When it is started, usually at system boot time, it opens a shared message queue of type *IPC_KERNELD* with the kernel, through which communication over IPC takes place [RUSxx]. The functions described below rely on the function *kerneld_send* which is coded in the file */ipc/msg.c*. It puts messages on the message queue. Sending and receiving of messages is performed by the functions *real_msgend* and *real_msgrcv*. The

data which moves through this message queue is expressed using the structure *kerneld_msg* as shown below:

```
struct kerneld_msg {
long mtype;     // contains the message.
long id;        // indicates whether the kernel expects an answer
short version;
short pid;      // holds the PID of the process that triggered the kernel request
char text[1];
};
```

Responsibility for loading and releasing modules lies with the functions:

request_module	the kernel requests the loading of a module and waits until the request is completed
release_module	removes a module
delayed_release_module	allows a module to be removed after a specified delay
cancel_release_module	cancels the previous function

Dynamically loadable modules require two programs when they are installed:

depmod - enables a dependency file to be generated based on the symbols located in the module group which is subsequently used by the second command.

modprobe - enables a module or group of modules to be loaded and also to load basic modules needed for a smooth boot-up of the machine (e.g. NFS, etc).

The kernel daemon converts requests for loading and releasing modules into calls to *modprobe*. This system program can assign the names of the modules to be loaded from the generic requests. It already possesses the names of all the modules used in the LINUX kernel, so that only new modules must be registered with an entry in the file /etc/modules.conf or /etc/conf.modules.

Loaded modules are unloaded automatically after a certain period if they are no longer being used. The default is set to 60 seconds but can be amended using the *delay* option.

Bibliography

[AND95] T. Anderson et al. (1995) A Case for Network of Workstations (NOW) *IEEE Mic*ro, Feb, 1995.

[BAK95] M. Baker (1995) Cluster Computing Review
http://www.npac.syr.edu/techreports/hypertext/sccs-748/index.html

[BEC98] M. Beck et al. *LINUX Kernel Internals,* 2nd Ed., Addison-Wesley, 1998

[BRE97] E. Brewer. (1997) "Clustering Multiply and Conquer". Data Communications, July 1997.
http://www.data.com/tutorials/clustering.html

[CAR98] R. Card et al. *The LINUX Kernel Book,* Wiley & Sons Ltd., 1998

[CHO98] P. Chowdhry & H. Baltazar. (1998). "Extreme Linux is Potent but Complex. Red Hat's Beowulf based clustering gives PC's awesome power.". PC Week, June 5, 1998.
http://www.zdnet.com/products/stories/reviews/0,4161,323718,00.html

[FIS97] G. Pfister (1997). In Search of Clusters, 2nd edition. Prentice-Hall. ISBN 0-13-899709-8, page 72.

[GAR99] M.Garvey, M. Hayes, S. Johnston. "More nodes for Win Clusters". InformationWeek, March 29, 1999.
http://www.informationweek.com/727/nt.htm

[GEL1] J. Gelinas et al. *modules.txt*
http://metalab.unc.edu/navigator-bin.cgi?Documentation/modules.txt

[GRE97] D. Greenberg et al. (1997). "A System Software Architecture for High-End Computing" In Proceedings of SuperComputing '97. November 15 – 21, 1997. San Jose, USA.

[HAL99] D. Halstead, B. Bode, D. Turner, V. Lewis. (1999). "Giga-Plant Scalable Cluster", Usenix Conference, Monterey, CA. June 7-11, 1999.

[HOF1] F. Hoffman, W Hargrove, A. Schultz. ( ) The Stone SouperComputer
http://www.esd.ornl.gov/facilities/beowulf/

[POL99] T. Poletti. (1999) "Plugged In: Linux Showing Up in Supercomputers".
http://www.eece.unm.edu/pages/plugged-in.html

[RID97] D. Ridge, D. Becker, P. Merkey, T. Sterling. (1997). “Beowulf: Harnessing the Power of Parallelism in a Pile of PCs”. In Proceedings, IEEE Aerospace Conference. 1997.

[RUL97] J. Ruley. (1997). “Wolfpack Alternatives”. http://winweb.winmag.com/library/1997/0501/ntent024.htm

[RUS98] D. Rusling. *The Linux Kernel* http://david.ghb.fh-furtwangen.de/LDP/tlk-0.8-2.html/tlk-toc.html

[SHA99] R. Shah. (1999). “Clustering for NT: Smooth scaling?”. Windows Techedge, July 03, 1999 http://www.windowstechedge.com/wte/wte-1999-07/wte-07-clustering.html

[STE97] T. Sterling. (1997). “Pile of PCs: A Beowulf Perspective”. Pentium Pro Workshop. April 10, 1997.

[STE98] T. Sterling, T. Cwik, D. Becker, J. Salmon, M. Warren, and B. Nitzberg. (1998). An assessment of Beowulf-class computing for NASA requirements: Initial findings from the first NASA workshop on Beowulf-class clustered computing. In Proceedings, IEEE Aerospace Conference. March 21-28, 1998, Aspen CO.

[STO97] H.Storner. Kerneld mini-HOWTO, http://ameli.experiment.db.erau.edu/ldp/HOWTO/mini/Kerneld.html

[UTH98a] P. Uthayopas et al. (1998). “Building a Parallel Computer from Cheap PCs: SMILE Cluster Experiences”, Proceedings of the Second Annual National Symposium on Computational Science and Engineering, Bangkok, Thailand, March 1998.

[UTH98b] P. Uthayopas et al. (1998). “Building a Resources Monitoring System for SMILE Beowulf Cluster”, Proceedings of the High Performance Computing Conference ASIA, Singapore, September 1998.

[WAR97a] M. Warren et al. (1997). “Parallel Supercomputing with Commodity Components”. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA’97), pages 1372-1381, June 30 - July 3, 1997, Monte Carlo Resort & Casino, Las Vegas, USA.

[WAR97b] M. S. Warren, J. K. Salmon, D. J. Becker, M. P. Goda, T. Sterling, and G. S. Winckelmans. “Pentium Pro inside: I. a treecode at 430 Gigaflops on ASCI Red, II. Price/performance of $50/Mflop on Loki and Hyglac”. In Supercomputing '97, Los Alamitos, 1997. IEEE Comp. Soc.

[WAR98] M. S. Warren, T. C. Germann, P. S. Lomdahl, D. M. Beazley and J. K. Salmon. (1998). “Avalon: An Alpha/Linux Cluster Achieves 10 Gflops for $150k”. In Proceedings of SuperComputing ’98. Los Alamitos, USA.

[HREF1] IEEE Task Force on Cluster Computing, TFCC Newsletter. Volume 1. No. 1. April, 1999.
http://www.eg.bucknell.edu/~hyde/tfcc/

[HREF2] Microsoft Research : Scalable Servers
http://www.research.microsoft.com/scalable

[HREF3] “Clustering vs SMP”. Data Communications.
http://www.data.com/tutorials/clustering_smp.html

[HREF4] Beowulf Project
http://www.beowulf.org

[HREF5] NCSA NT Cluster
http://www.ncsa.uiuc.edu/General/CC/ntcluster

[HREF6] Microsoft NT SERVER Clustering Overview
http://www.microsoft.com/ntserver/ntserverenterprise/exec/overview/Clustering/Default.asp

[HREF7] “Introduction to Clusters”. SCL Cluster Cookbook.
http://www.scl.ameslab.gov/Projects/ClusterCookbook/index.html

[HREF8] High Performance Computing on Windows NT
http://www-csag.ucsd.edu/

[HREF9] Compaq Cluster Management Vision
http://www.compaq.com/solutions/enterprise/highavailability-clustermgmt-vision-art1.html

[HREF10] CORBA
http://www.omg.org

[HREF11] ORBacus
http://www.ooc.com/ob

[HREF12] Computer Architecture Group Cluster
http://www.cs.tcd.ie/Brian.Coghlan/cluster/index.html

[HREF13] Perl Practical Extraction and Language
http://www.perl.org

[HREF14] Blackdown JDK
http://www.blackdown.org

[HREF15] IEEE Scalable Coherent Interface
http://standards.ieee.org/catalog/bus.html