Cluster Operating Systems

Russell W Clarke (12278769)
AND
Benjamin T B Lee (12286818)

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

Email: {rwc, benlee}@csse.monash.edu.au

Abstract

Recent advances in network and workstation performance have made clustered computer systems an attractive solution for variable workloads, and both sequential and parallel applications. However, although the hardware is available to construct these distributed systems, rendering the system usable from an operators point of view is still an issue. The problem lies in that single computer operating systems do not scale in any useable manner to the clustered architecture. In this report we will give an overview of several operating systems designed specifically for clustered or distributed systems.

Background / Works in this area

Like any other operating system, a cluster operating system must provide a user-friendly interface between the user, the applications and the cluster hardware.

In a distributed operating system, it is imperative to have a Single System Image (SSI) such that users can interact with the system as if it were a single computer. It is also necessary to implement fault tolerance or error recovery features; that is, when a node fails, or is removed from the cluster, the rest of the processes running on the cluster can continue to function. The cluster operating system should be scalable, and should provide that the system is highly available.

Implementing a SSI involves making the cluster appear to be a single computer to the users, and to the network the cluster is connected to. The SSI should implement a single entry point to the cluster, a single file hierarchy (usually implementing error-recovery using journalling), a single control point, virtual networking, single memory space, single job management, and a single user interface.

For the high availability of the cluster, the operating system should feature a single I/O space, a single process space, and process migration.

Over the last two decades, researchers have constructed many prototype and production quality cluster operating systems. Below is a table briefly outlining several,

Operating System	***Description***	***URL for more information***
Alpha Kernel	Developed at Carnegie Mellon University, started in 1985, built from scratch on custom made distributed system. Intended to be a real-time distributed OS, development seems to have finished around 1995.	http://www.realtime-os.com/alpha.html
Amoeba	Developed at Vrije Universitei, started about 1990. Design around a microkernel, ran on both Motorola and Intel based hardware.	http://www.cs.vu.nl/pub/amoeba/
Angel	Developed at City University of London, started in 1991. Initially not UNIX compatible, based on microkernel, emphasis on single address space.	http://www.soi.city.ac.uk/research/sarc/angel
ChorusOS	Developed at Sun Microsystems. Microkernel based, real-time, embedded, scalable and distributed OS. Used in telecommunications equipment, e.g. PBXs, public switches.	http://www.sun.com/chorusos/
GLUnix	Developed at University of California, Berkeley, began mid-1990s. Implemented at the user level on top of Solaris 2.3- 2.6	http://now.cs.berkeley.edu/Glunix/glunix.html
Guide	Developed by Bull and Universities of Grenoble, object-oriented distributed operating system. Began late-1980s, redevelopment in 1991, moved to Mach based kernel, emphasis on using C++ persistent objects to keep track of jobs.	http://www-bi.imag.fr/GUIDE/presguide.html
Hurricane	Developed at University of Toronto, began early 1990s. Hierarchically clustered operating system implemented on the Hector multiprocessor.	http://www.eecg.toronto.edu/EECG/RESEARCH/ParallelSys/hurricane.html
MOSIX	Developed at The Hebrew University of Jerusalem, began early 1990s. Patches Linux/UNIX kernel to add clustering capabilities.	http://www.mosix.org/
Plan 9	Developed at Bell Labs Computing Sciences Research Centre, distributed operating system and associated utilities. Built from scratch.	http://plan9.bell-labs.com/plan9/

QNX	Developed by QNX Software Systems, commercial, microkernel based, real-time, can be distributed or embedded.	http://www.qnx.com/
RHODOS	Developed at Deakin University, microkernel based, research oriented, has features to monitor statistics about the system.	http://www3.cm.deakin.edu.au/rhodos/
Solaris MC	Developed at Sun Microsystems, implemented as extensions to the base Solaris UNIX ® system.	http://www.sun.com/
Spring	Developed at Sun Microsystems, early 1990s, distributed, object-oriented, microkernel framework. Focused on developing a fast IPC mechanism so that the OS’s performance would not be degraded by memory distribution.	http://www.sun.com/
Sprite	Developed at University of California, Berkeley, began in 1984, research OS, written from scratch, very cool, explored many concepts of distributed OSs, e.g. network file system, journalling filesystem, process migration.	http://www.CS.Berkeley.EDU/projects/sprite/
Sumo	Developed at Lancaster University, began mid-1990s, microkernel based system with facilities to support distributed real-time and multimedia applications, based off Chorus microkernel	http://www.comp.lancs.ac.uk/computing/research/sumo/sumo.html
UnixWare Non Stop Clusters	Developed by SCO, based on the UnixWare 7 kernel, and has been modified via a series of modular extensions and hooks	http://www.sco.com/
VAXclusters	Developed by Compaq / DEC, around 1983, tied VAX processors together in a loose processor coupling that allowed multiple VAXs to operate as a single computer	http://www.digital.com/

Some of these projects continue, some have ended, whilst some have become commercially viable (or were always intended to be, e.g. SCO UnixWare).

We will now outline four cluster operating systems, Solaris MC, UnixWare, MOSIX and GLUnix.

Solaris MC

Solaris MC is a prototype, distributed operating system for multi-computers (or clusters). It provides a single-system image -- a cluster appears to the user and applications as a single computer running the Solaris (TM) UNIX (R) operating system.

Solaris MC is built on top of the Solaris operating system. Most of Solaris MC consists of loadable modules extending the Solaris OS, and minimises the modifications to the existing Solaris kernel. Thus, Solaris MC shows how an existing, widely used operating system can be extended to support clusters.

Solaris MC provides the same MPI/MBI as Solaris, which means that existing application and device driver binaries run unmodified on Solaris MC. To provide this feature, Solaris MC has a global file system, extends process operations across all the nodes, allows transparent access to remote devices, and makes the cluster appear as a single machine on the network. Solaris MC also supports remote process execution and the external network is transparently accessible from any node in the cluster.

The components of Solaris MC are implemented in C++ through a CORBA-compliant (common object request broker architecture) object-oriented system with all new services defined by the IDL (interface definition language) definition language. Objects communicate through a runtime system that borrows from Solaris doors and Spring subcontracts. This shows that a common, high-level language such as C++ can be used to implement clustering.

Solaris MC is designed for high availability. If a node fails, the cluster can continue to operate. Solaris MC runs a separate kernel on each node. A failure of a node does not cause the whole system to fail. A failed node is detected automatically and system services are reconfigured to use the remaining nodes. Only the programs that were using the resources of the failed node are affected by the failure. Solaris MC does not introduce new failure modes into UNIX.

Solaris MC has a distributed caching file system with UNIX consistency semantics, based on the Spring virtual memory and file system architecture. Solaris MC imports from Spring the idea of using a CORBA-compliant object model as the communication mechanism, the Spring virtual memory and file system architecture, and the use of C++ as the implementation language.

Solaris MC's global file system, called the Proxy File System (PXFS), makes file accesses location transparent. A process can open a file located anywhere on the system and each node uses the same pathname to the file. The file system also preserves the standard UNIX file access semantics, and files can be accessed simultaneously by different nodes. PXFS is built on top of the original Solaris file system by modifying the vnode interface, and does not require any kernel modifications.

Solaris MC's global process management system makes the location of a process transparent to the user. The threads of a single process must run on the same node, but each process can run on any node. The system is designed so that POSIX semantics for process operations are supported, as well as providing good performance, supplying high availability, and minimising changes to the existing Solaris kernel.

Solaris MC's I/O subsystem makes it possible to access any I/O device from any node on the system, no matter which node the device is physically connected to. Applications can access all devices on the system as if they were local devices. Solaris MC carries over Solaris's dynamically loadable and configurable device drivers, and configurations are shared through a distributed device server.

Solaris MC's networking subsystem creates a single image environment for networking applications. All network connectivity is consistent for each application, no matter which node the application runs on. This is achieved by using a packet filter to route packets to the proper node and perform protocol processing on that node. Network services can be replicated on multiple nodes to provide lighter throughput and lower response times. Multiple processes register themselves as servers for a particular service. The network subsystem then chooses a particular process when a service request is received. For example, rlogin, telnet, http and ftp servers are replicated on each node by default. Each new connection to these services is sent to a different node in the cluster based on a load balancing policy. This allows the cluster to handle multiple http requests, for example, in parallel.

Solaris MC provides high availability in many ways, including a failure detection and membership service, an object and communication framework, a system reconfiguration service and a user-level program reconfiguration service. Some nodes can be specified to run as backup nodes, is a node attached to a disk drive fails, for example. Any failures are transparent to the user.

Solaris MC uses ideas from earlier distributed operating systems such as Chorus Mix, LOCUS, MOS, OSF/1 AD TNC, Spring, Sprite and VAXclusters. Solaris MC is different in that it is based on a major commercial operating system, and shows how a commercial operating system can be extended to a cluster while maintaining compatibility with a large existing applications base. Solaris MC emphasises high availability, which the other operating systems listed above (with the exception of VAXclusters) do not. Finally, Solaris MC is designed in an object-oriented way, in C++ and built with the CORBA object model.

UnixWare & UnixWare NonStop Clusters

UnixWare (TM) is a high-powered operating system, intended for use as an Internet access server, http server, ftp server, domain name server, mail server, etc. It also supports clustering since version 2.0, and in its latest version, 7.0, provides high availability clustering.

UnixWare is one of SCO's UNIX (R) operating systems. It combines UNIX System V Release 4.2 for Intel 80x86 processors with NetWare client/server connectivity, and many other tools. It provides a graphical user interface based on the X11R5 windowing system, and is capable of running Unix, DOS, and Windows programs. The most recent version is UnixWare 7.

UnixWare was originally developed jointly by Novell and Unix System Laboratories (USL, 77% owned by AT&T). At that time, is was known as Univel. In June 1993, Novell bought USL, and USL and Univel were integrated into the Novell Unix Systems Group (USG).

Earlier versions of UnixWare were developed by the same group of people that worked with Sun to create UNIX System V Release 4 (SVR4). UnixWare 1.0 was based on SVR4.2, released by USL in July 1992, and UnixWare 2.0 was based on SVR4.2 MP, released by USL in December 1993. Since then, it has always supported multiprocessors and clustering.

Novell stopped developing UNIX operating systems in September 1995, and UnixWare was sold to SCO in December 1995. UnixWare is now developed by SCO, in Murray Hill, New Jersey, USA; Watford, UK; Moscow, Russia and Santa Cruz, California.

UnixWare has a large featureset, making it an attractive general-purpose high-end operating system. It conforms to the Single UNIX Specification, a specification drawn up by The Open Group, based on the IEEE POSIX 1003.1 standard, AT&T's System V Interface Definition, X/Open's XPG4 Version 2 interface specification and other third party APIs.

As well as supporting UNIX SVR4.2 MP multiprocessing, it features a 32-bit, fully-multithreaded kernel and preemptive multitasking. Its kernel, disk I/O and networking protocols are multithreaded, and all I/O is asynchronous. Device drivers, file systems, and other modules can be loaded dynamically.

UnixWare also provides a set of user-level threads APIs, which let users write multithreaded applications, which are tightly integrated with the kernel. The lets users write applications for optimal performance and scalability.

The latest version of UnixWare, called UnixWare 7, is based on the SVR5 kernel, which allows for memory up to 64 GB, filesystems up to 1 TB, and total storage of 76,500 TB, and improves overall performance. It also improves on reliability and fault tolerance, including high availability clustering, dynamical kernel drivers which can be loaded and unloaded without restarting, and hot-swappable devices.

UnixWare supports networks such as Fast Ethernet, FDDI and ISDN. It also supports Multipath I/O which gives built-in SCSI HA redundancy support with commodity SCSI HAs.

SCO packages UnixWare into five bundles, called "Enterprise Edition" (for high-end servers and databases), "Departmental Edition" (for medium-large organisations), "Intranet Edition" (for general Internet/intranet servers), "Messaging Edition" (for mail and messaging servers) and "Base Edition" (for building specialised servers). The main differences between the bundles are the amount of networking support that is included. In addition, a limited, free version of UnixWare is available for educational and non-commercial use.

UnixWare's hardware requirements are not large. Even for the Enterprise Edition, it only requires 32 MB of memory (on each node in a cluster) and 500 MB of disk space (1 GB recommended). As always, the more memory, the better.

Many software packages are available for UnixWare, including Netscape Communicator, Netscape FastTrack, WordPerfect, Informix, Oracle, Microfocus COBOL, Cleo SNA Server and numerous free software.

To use UnixWare in a cluster, UnixWare NonStop Clusters (NSC) is required. This is a separate software package which lets you use a network of computers running UnixWare 7 as a cluster. Each node must already have UnixWare 7 installed on it. Nodes in a cluster each run an identical copy of kernel, reducing OS-to-OS message passing. It supports both SMP and non-SMP nodes in the same cluster, support for different versions of the OS in the same cluster.

The UnixWare NSC kernel is based on the UnixWare 7 kernel, and has been modified via a series of modular extensions and hooks. These provide the clustering features, such as cluster-wide filesystem, device access, swap space, IPC, processes, TCP/IP networking, membership and time synchronisation.

UnixWare NSC actively balances the load, meaning the processes can be migrated automatically (or manually) from more heavily-used nodes to less loaded nodes. New nodes can be added while the cluster is still running. UnixWare NSC automatically assimilates new memory and processing. UnixWare NSC implements distributed kernel algorithms and is designed to handle hundreds of nodes.

UnixWare NSC’s single system image (SSI) hides the underlying cluster implementation, which means that applications and users don’t need to know that the system is in fact a cluster. Administration tools are the same as for UnixWare. In fact, applications that run under UnixWare will run unmodified on UnixWare NSC. However, applications can be optimised for UnixWare NSC. Scripts can be written to control availability of nodes and scalability.

File systems and devices are accessible transparently across the cluster, regardless of which node the hardware is actually attached to. A feature called the cluster filing environment (CFE) ensures that file systems are known by the same name across all nodes. The semantics of file system accesses (for example, the read() and write() functions) are the same for all nodes. The file system is journaled (log-based) for data security. There is also built-in support for RAID disks.

Other cluster-wide services are cluster-wide swap space and time synchronisation, which keeps system time on all nodes synchronised to within 10 milliseconds. This is especially important for situations when a process is moved to another node for load balancing (also called load levelling), since UnixWare NSC provides automated process migration and load levelling.

UnixWare NSC also provides cluster-wide process management. Each process has its own unique process ID. It maintains UnixWare interprocess relationships, making all processes on all nodes visible. System V interprocess communication (IPC) is shared across the entire cluster. This supports sharing memory segments, message queues, pipes and semaphores between independent processes. Objects are named using keys and handles, and a single IPC namespace is supported over the cluster.

UnixWare NSC is installed on a node called the initial root node. It controls the root file system. Normally, the cluster boots from the initial root node, but if that node is unavailable, the cluster boots from a designated backup node, called the failover root node. This means that the boot disk must be external, shared by both root nodes. Another option is to mirror the boot disk on both root nodes, called cross-node mirroring. SCO recommends using the external disk option, using a dual-hosted external SCSI disk.

Networking in UnixWare NSC is via IP. Each node is given a network name and IP address. Additionally, each node has a Cluster Virtual IP (CVIP), which allows one or more network names and IP addresses to be associated with the cluster as a whole, rather than to a particular node. This allows the cluster to appear as a single system on a network.

UnixWare NSC contains a "split-brain" avoidance feature. The term "split brain" refers to the condition in which a cluster’s root nodes cannot communicate to determine which node should be the current root node filesystem server. To avoid the split-brain problem, the cluster requires a ServerNet system area network (SAN) switch. UnixWare NSC can then detect a potential split-brain problem and avoids it based on each node’s ability to communicate with the switch.

UnixWare NSC emphasises fault tolerance, and was designed with continuous availability in mind. If a node fails, or an application running on a node fails, processes are automatically migrated to the remaining nodes and resumed. Administrator tools allow any failed nodes to be restarted and returned to operation as quickly as possible. UnixWare NSC’s fault tolerance is partly based on work done by Compaq’s Telco Division (formerly Tandem Computers, Ltd). Most other clustering systems are simply "high availability clusters", whereas UnixWare NSC is a fault-tolerant cluster with high-availability nodes.

UnixWare NSC’s support for application failover uses an "n+1" approach. The backup copy of the application can be restarted on any of the nodes in the cluster, allowing one node to act as a backup node for all the other nodes. Many other cluster architectures, such as the Hewlett-Packard ServiceGuard, apply a "1+1" approach, which requires a dedicated spare node for each active node in the cluster. The ability to migrate processes automatically means that nodes can be added and removed without disruption.

UnixWare NSC takes approximately 2--10 minutes to migrate applications to different nodes. During this time, those applications are not available, but there are many situations where this is acceptable, and is preferable to an hour of downtime.

Traditional clusters typically force applications to fit their availability model. UnixWare NSC's SSI enables applications and upper levels of the operating system to run without modification. From the point of view of the system administrator, there is one root file system, one set of users and one password file, a single host table, and a single network address. Standard system administration tools can be used, and there is no need to run special commands or address individual nodes.

UnixWare NSC supports the ServerNet fault-tolerant interconnect technology. The ServerNet SAN is a low-latency interconnect technology which reduces the performance hit as extra nodes are added. Using ServerNet, near 1:1 scaling has been achieved for 2--6 nodes. Any element in a ServerNet cluster -- processor, disk or I/O device -- can interact with any other element without processor intervention. ServerNet latency is at 0.3 microseconds as opposed to 200 microseconds for Fast Ethernet or 50 microseconds for SCI.

UnixWare NSC's SSI is not just at the application level, but also at the operating system and device driver level. This ensures greater compatibility and ease of administration. Most clusters support SSI at the application level only. Applications do not need to be cluster-aware, which contrasts with other clustering systems, most of which require some modification to run on a cluster. System administration again does not need to be cluster-aware, unlike most other clustering systems. The ability to actively migrate processes also separates UnixWare NSC from most other clustering systems.

UnixWare NSC is ideal in environments where availability is an important requirement, and large data storage and fast data processing are also essential. Typical applications include databases, customer support systems, transaction processing, and Internet and mail servers. Any system that stores and manages large amounts of data or maintains mission critical transactions can benefit from clustering technology.

Comparable products are primarily RISC-based servers from Sun Microsystems, HP and IBM. These servers are also highly scalable and can provide significant quantity of memory and storage, but they cost more than standard Intel-based PCs, and fault-zones increase as memory and storage increase at each node.

MOSIX

MOSIX is a software package that extends the Linux[1] kernel with cluster computing capabilities. The enhanced Linux kernel allows any size cluster of Intel based computers to work together like a single system, very much like a SMP (symmetrical multi processor) system.

MOSIX operates silently, and its operations are transparent to user applications. Users run applications sequentially or in parallel just like they would do on a SMP. Users need not know where their processes are running, nor be concerned with what other users are doing at the time. After a new process is created, MOSIX attempts to assign it to the best available node at that time. MOSIX continues to monitor all running processes (including the new process). In order to maximise overall cluster performance, MOSIX will automatically move processes amongst the cluster nodes when the load is unbalanced. This is all accomplished without changing the Linux interface.

As it may be obvious already, Existing applications do not need any modifications to run on a MOSIX cluster, nor do they need to be linked with any special libraries. In fact, it is not even necessary to specify the cluster nodes on which the application will run. MOSIX does all this automatically and transparently. In this respect, MOSIX acts like a SMP’s "fork and forget" paradigm. Many user processes can be created at a home node, and MOSIX will assign the processes to other nodes if necessary. If the user was to run a "ps", he would be shown all his owned processes as if they running on the node he started them on. This is called providing the user with a single server image. It is because MOSIX is implemented in the OS kernel that its operations are completely transparent to user-level applications.

At the core of MOSIX are its adaptive load-balancing and memory ushering, resource management algorithms. These respond to changes in the usage of cluster resources in order to improve the overall performance of all running processes. The algorithms use pre-emptive (on-line) process migration to assign and reassign running processes amongst the nodes. This ensures that the cluster takes full advantage of the available resources. The dynamic load balancing algorithm ensures that the load across the cluster is evenly distributed. The memory ushering algorithm prevents excessive hard disk swapping by allocating or migrating processes to nodes that have sufficient memory. The MOSIX algorithms are designed for maximum performance, minimal overhead cost and ease-of-use.

The MOSIX resource management algorithms are decentralised. That is, every node in the cluster is both a master for locally created processes, and a server for remote (migrated) processes. The benefit of this decentralisation is that running processes on the cluster are minimally affected when nodes are added or removed from the cluster. This greatly adds to the scalability and the high availability of the system.

Another advantageous feature of MOSIX is its self tuning and monitoring algorithms. These detect the speed of the nodes, monitor their load and available memory, as well as the IPC and I/O rates of each running process. Using this data, MOSIX can make optimised decisions about where to locate processes.

In the same manner that NFS provides transparent access to a single consistent file system, MOSIX enhancements provides a transparent and consistent view of processes running on the cluster.

The single system image model of MOSIX is based on the home node model. In this model, all user's processes seem to run on the user's login node. Every new process is created on the same node/s as its parent process. The execution environment of a user's process is the same as the user's login node. Migrated processes interact with the user's environment through the user's home node; however, wherever possible, the migrated process uses local resources. The MOSIX system is set up such that whilst the load of the user's login node remains below a threshold value, all the user's processes are confined to that node. When the load exceeds the login node’s threshold value, the process migration mechanism will begin to migrate process/es to other nodes in the cluster. This migration is done transparently without any user intervention.

To provide consistent access to the filesystem, MOSIX uses its shared MOSIX File System (MFS). MFS makes all directories and regular files throughout a MOSIX cluster available from all nodes. MFS provides absolute consistency as files are viewed from different nodes.

[1] MOSIX has been developed seven times for various flavours of UNIX and architectures. The first PC version was developed for BSD/OS, and the most recent version is for Linux on Intel based platforms.

The Direct File System Access (DFSA) provision (under research and testing) extends the capability of a migrated process to perform I/O operations locally, in the current (remote) node. This provision decreases communication between I/O bound processes and their home nodes. This allows such processes to migrate with greater flexibility among the cluster's nodes, e.g., for more efficient load balancing, or to perform parallel file and I/O operations. Currently, the MOSIX File System (MFS) meets the DFSA standards.

MOSIX is flexible and can be used to define different cluster types, even clusters with various kinds of machines or LAN speeds. MOSIX supports cluster configurations with a small to large number of computers with minimal scaling overheads. A small low-end configuration may consist of several PCs that are connected by Ethernet. A larger configuration may consist of workstations connected by a higher speed LAN such as Fast Ethernet. A high-end configuration may consist of a large number of SMP and non-SMP computers connected by a high performance LAN, like Gigabit Ethernet or Myrinet. The scalable PC cluster at Hebrew University, where the MOSIX development is based, consists of Pentium servers that are connected by a Myrinet and Fast Ethernet LANs.

Similar to Solaris MC, MOSIX is built upon existing software making it easy to make the transition from standalone systems to a clustered system. As they write on the web site, 'If you have several Linux computers (connected by a LAN) and you want to use them as a single system, download and install the latest release of MOSIX. The installation is straightforward for the RedHat and the SuSE distributions of Linux.'

GLUnix (Global Layer Unix for a Network Of Workstations)

GLUnix was originally designed as a global operating system for the Berkeley Network of Workstations (NOW). The NOW project's goal was to construct a platform that would support both parallel and sequential applications on commodity hardware. In order to fulfil the goal of the NOW project, it was decided that a specialised cluster operating system was required.

The cluster operating system needed to provide:

- Transparent Remote Execution – Applications should be oblivious to the fact that a process may be running on a home node or a remote node.
- Load Balancing – The system should implement an intelligent process placement mechanism.
- Binary Compatibility – The system should provide the transparent remote execution, and load balancing and any other features, without requiring modifications to an application or relinking.
- High Availability – This was been discussed many times and simple means that when a node fails or is removed from the cluster, the system should continue to operate.
- Portable – The system should be readily portable to other architectures.

After considering the alternatives, the Berkeley researchers decided to build a global runtime environment at user level rather than modify any existing operating system. This is different to MOSIX, for example, where the clustering capabilities are added to the kernel.

GLUnix, as actually built, provides remote execution but not entirely transparently. The operating system provides intelligent load balancing but not process migration. GLUnix runs existing binaries. It can handle many node failures but not a failure of the node running the GLUnix master process.

GLUnix provides a cluster wide name space by using global unique network process identifiers (NPIDs) for all GLUnix jobs. Also processes constituting a parallel application are assigned Virtual Node Numbers (VNNs). The process identifiers are 32-bit. GLUnix uses one NPID to identify all the *N* processes constituting a parallel program. This enables the standard job control signals such as Ctrl-C and Ctrl-Z to function as expected for both sequential and parallel programs. Together NPIDs and VNNs (<NPID, [VNN]>), can unique distinguish any process throughout the cluster. Communication between processes is thus accomplished by using NPIDs and VNNs.

GLUnix provides a set of command line tools to manipulate NPIDs and VNNs. These include, glurun, glukill, glumake, glush, glups, glustat. These tool perform similar functions to their standard UNIX counterparts. The glurun utility enables an existing application to be run under GLUnix without modification.

GLUnix also provides some common library functions to spawn processes, set up barriers etc.

The glurun shell command attempts to execute the program on the least loaded node. The system provides the user and the application with the illusion of running on single computer by redirecting input, output and I/O and signals back to the user's shell on the node where the process was created. This redirection functionality is provided by the GLUnix startup process, which is started when the user executes the command glurun.

GLUnix operates using a high performance global file system (developed for NOW) and homogenous operating systems. This assumption simplifies the issues of byte ordering, data type size etc.

GLUnix consists of a per-cluster master, a per-node daemon, and a per-application library. The GLUnix master is responsible for managing the cluster resources, for example, job scheduling. The per-node daemon collects information (load) about the node it is running on, and notifies the master of changes. The GLUnix library provides an API allowing programs to explicitly utilise GLUnix services. As mentioned before applications, both sequential and parallel, are handled by the GLUnix startup process. The startup process sends the execution request to the master and then handles (like a proxy) any user interaction with the program.

GLUnix is currently able to handle failures in the daemon and startup processes. The master continually monitors each daemon and startup process. If it detects that a daemon has failed, it looks up a process database and for any sequential processes running in that node, it marks them killed. It then notifies the startup process. If there were any parallel processes in the node, the master sends a SIGKILL to all the other associated processes. Finally the master removes the node from the node database and continues normal operation.

When a startup process fails, the master sends a SIGKILL to all remote processes, removes the node from the node database, and continues normal operation.

GLUnix has a limitation on its scalability. In order to pipe input and output between the user's shell and the remote program, GLUnix must set up TCP connections between each remote process of a parallel application and the startup process. Three connections are used, one for stdout, one fore stderr, and one for stdin. Because GLUnix uses three socket connections, the operating system underlying GLUnix becomes the limiting factor on how large a parallel program may be. For example, in Solaris 2.5 the number of file descriptors for a single process is limited to 1024, this means that a parallel program running under GLUnix can have only up to 341 processes.

Summary and Conclusions

Cluster operating systems have been developed and researched over the last couple of decades. Several research projects and commercial companies have successfully constructed production quality operating systems. Presently, it not a matter of determining what capabilities a cluster operating system requires, but rather whether it is practical or feasible to implement all required features. As we can see from such projects as GLUnix, certain feature sacrifices were made for backward compatibility and portability.

Clusters are rapidly becoming the solution of choice for many applications, sequential and parallel. We believe existing cluster operating systems have matured enough to be used in production, even though they may lack a feature or two. And with the advent of faster and faster network technology, clustered systems will become ever more popular.

References

[1] Ghormley, D. P., et al., *GLUnix: a Global Layer Unix for a Network of Workstations*, Computer Sciences Division, University of California, Berkeley, CA. 1997.

[2] Sun Microsystems - http://www.sun.com/

[3] The Santa Cruz Operation http://www.sco.com/

[4] MOSIX http://www.mosix.org/