Cluster Computing

Georgetta Rivero

First Edition, 2012

ISBN 978-81-323-3398-2

Published by:
Research World
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter 1 - Computer Cluster

Chapter 2 - Beowulf (Computing)

Chapter 3 - ProActive and LOCUS (Operating System)

Chapter 4 - OpenSSI and MOSIX

Chapter 5 - OpenVMS

Chapter 6 - Oracle Grid Engine and OpenMosix

Chapter 7 - Oracle RAC and IBM Parallel Sysplex

Chapter 8 - Xgrid

Chapter 9 - Single System Image and Red Hat Cluster Suite

Chapter 10 - CHAOS (Operating System) and FhGFS

Chapter 11 - FinisTerrae and VMScluster

Chapter 12 - Solaris Cluster, Veritas Cluster Server and GPU Cluster

Chapter 13 - High-availability Cluster

Chapter 14 - Common Address Redundancy Protocol and Linux Virtual Server

Chapter 15 - Computer Cluster in Virtual Machines and Two-node Cluster

Chapter-1

Computer Cluster

The Silicon Graphics Cluster-SGI; an example of a cluster computer

A **computer cluster** is a group of linked computers, working together closely thus in many respects forming a single computer. The components of a cluster are commonly, but not always, connected to each other through fast local area networks. Clusters are usually deployed to improve performance and availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability.

Cluster categorizations

High-availability (HA) clusters

High-availability clusters (also known as Failover Clusters) are implemented primarily for the purpose of improving the availability of services that the cluster provides. They operate by having redundant nodes, which are then used to provide service when system components fail. The most common size for an HA cluster is two nodes, which is the minimum requirement to provide redundancy. HA cluster implementations attempt to use redundancy of cluster components to eliminate single points of failure.

There are commercial implementations of High-Availability clusters for many operating systems. The Linux-HA project is one commonly used free software HA package for the Linux operating system. The LanderCluster from Lander Software can run on Windows, Linux, and UNIX platforms.

Load-balancing clusters

Load-balancing is when multiple computers are linked together to share computational workload or function as a single virtual computer. Logically, from the user side, they are multiple machines, but function as a single virtual machine. Requests initiated from the user are managed by, and distributed among, all the standalone computers to form a cluster. This results in balanced computational work among different machines, improving the performance of the cluster systems.

Compute clusters

Often clusters are used primarily for computational purposes, rather than handling IO-oriented operations such as web service or databases. For instance, a cluster might support computational simulations of weather or vehicle crashes. The primary distinction within computer clusters is how tightly-coupled the individual nodes are. For instance, a single computer job may require frequent communication among nodes - this implies that the cluster shares a dedicated network, is densely located, and probably has homogenous nodes. This cluster design is usually referred to as Beowulf Cluster. The other extreme is where a computer job uses one or few nodes, and needs little or no inter-node communication. This latter category is sometimes called "Grid" computing. Tightly-coupled compute clusters are designed for work that might traditionally have been called "supercomputing". Middleware such as MPI (Message Passing Interface) or PVM (Parallel Virtual Machine) permits compute clustering programs to be portable to a wide variety of clusters.

Implementations

The TOP500 organization's semiannual list of the 500 fastest computers usually includes many clusters. TOP500 is a collaboration between the University of Mannheim, the University of Tennessee, and the National Energy Research Scientific Computing Center at Lawrence Berkeley National Laboratory. As of January 2011 the top supercomputer is the Tianhe-1A in Tianjin, China, with performance of 2566 TFlops measured with the High-Performance LINPACK benchmark.

Clustering can provide significant performance benefits versus price. The System X supercomputer at Virginia Tech, the 28th most powerful supercomputer on Earth as of June 2006, is a 12.25 TFlops computer cluster of 1100 Apple XServe G5 2.3 GHz dual-processor machines (4 GB RAM, 80 GB SATA HD) running Mac OS X and using InfiniBand interconnect. The cluster initially consisted of Power Mac G5s; the rack-mountable XServes are denser than desktop Macs, reducing the aggregate size of the cluster. The total cost of the previous Power Mac system was $5.2 million, a tenth of the cost of slower mainframe computer supercomputers. (The Power Mac G5s were sold off.)

The central concept of a Beowulf cluster is the use of commercial off-the-shelf (COTS) computers to produce a cost-effective alternative to a traditional supercomputer. One project that took this to an extreme was the Stone Soupercomputer.

However it is worth noting that Flops (floating point operations per second), aren't always the best metric for supercomputer speed. Clusters can have very high Flops, but they cannot access all data in the cluster as a whole at once. Therefore clusters are excellent for parallel computation, but much poorer than traditional supercomputers at non-parallel computation.

JavaSpaces is a specification from Sun Microsystems that enables clustering computers via a distributed shared memory.

Consumer game consoles

Due to the increasing computing power of each generation of game consoles, a novel use has emerged where they are repurposed into High-performance computing(HPC) clusters. Some examples of game console clusters are Sony PlayStation clusters and Microsoft Xbox clusters. It has been suggested on a news website that countries which are restricted from buying supercomputing technologies may be obtaining game systems to build computer clusters for military use, though most of the intelligence in the article has since been debunked.

History

The history of cluster computing is best captured by a footnote in Greg Pfister's *In Search of Clusters*: “Virtually every press release from DEC mentioning clusters says ‘DEC, who invented clusters…’. IBM did not invent them either. *Customers* invented clusters, as soon as they could not fit all their work on one computer, or needed a backup. The date of the first is unknown, but it would be surprising if it was not in the 1960s, or even late 1950s.”

The formal *engineering* basis of cluster computing as a means of doing parallel work of any sort was arguably invented by Gene Amdahl of IBM, who in 1967 published what has come to be regarded as the seminal paper on parallel processing: Amdahl's Law. Amdahl's Law describes mathematically the speedup one can expect from parallelizing any given otherwise serially performed task on a parallel architecture. Here we, defined the engineering basis for both multiprocessor computing and cluster computing, where the primary differentiator is whether or not the interprocessor communications are supported "inside" the computer (on for example a customized internal communications bus or network) or "outside" the computer on a *commodity* network.

Consequently the history of early computer clusters is more or less directly tied into the history of early networks, as one of the primary motivation for the development of a network was to link computing resources, creating a de facto computer cluster. Packet switching networks were conceptually invented by the RAND corporation in 1962. Using the concept of a packet switched network, the ARPANET project succeeded in creating in 1969 what was arguably the world's first commodity-network based computer cluster by linking four different computer centers (each of which was something of a "cluster" in its own right, but probably not a *commodity* cluster). The ARPANET project grew into the Internet—which can be thought of as "the mother of all computer clusters" (as the union of nearly all of the compute resources, including clusters, that happen to be connected). It also established the paradigm in use by *all* computer clusters in the world today—the use of packet-switched networks to perform interprocessor communications between processor (sets) located in otherwise disconnected frames.

The development of customer-built and research clusters proceeded hand in hand with that of both networks and the Unix operating system from the early 1970s, as both TCP/IP and the Xerox PARC project created and formalized protocols for network-based communications. The Hydra operating system was built for a cluster of DEC PDP-11 minicomputers called C.mmp at Carnegie Mellon University in 1971. However, it was not until circa 1983 that the protocols and tools for *easily* doing remote job distribution and file sharing were defined (largely within the context of BSD Unix, as implemented by Sun Microsystems) and hence became generally available commercially, along with a shared filesystem.

The *first* commercial clustering product was ARCnet, developed by Datapoint in 1977. ARCnet was not a commercial success and clustering per se did not really take off until DEC released their VAXcluster product in 1984 for the VAX/VMS operating system. The ARCnet and VAXcluster products not only supported parallel computing, but also shared file systems and peripheral devices. The idea was to provide the advantages of parallel processing, while maintaining data reliability and uniqueness. VAXcluster, now VMScluster, is still available on OpenVMS systems from HP running on Alpha and Itanium systems.

Two other noteworthy early commercial clusters were the *Tandem Himalaya* (a circa 1994 high-availability product) and the *IBM S/390 Parallel Sysplex* (also circa 1994, primarily for business use).

No history of commodity computer clusters would be complete without noting the pivotal role played by the development of Parallel Virtual Machine (PVM) software in 1989. This open source software based on TCP/IP communications enabled the *instant* creation of a virtual supercomputer—a high performance compute cluster—made out of any TCP/IP connected systems. Free form heterogeneous clusters built on top of this model rapidly achieved total throughput in FLOPS that greatly exceeded that available even with the most expensive "big iron" supercomputers. PVM and the advent of inexpensive networked PCs led, in 1993, to a NASA project to build supercomputers out of commodity clusters. In 1995 the invention of the "beowulf"-style cluster—a compute cluster built on top of a commodity network for the specific purpose of "being a supercomputer" capable of performing tightly coupled parallel HPC computations. This in turn spurred the independent development of Grid computing as a named entity, although Grid-style clustering had been around at least as long as the Unix operating system and the Arpanet, whether or not it, or the clusters that used it, were named.

Technologies

MPI is a widely-available communications library that enables parallel programs to be written in C, Fortran, Python, OCaml, and many other programming languages.

The GNU/Linux world supports various cluster software; for application clustering, there is Beowulf, distcc, and MPICH. Linux Virtual Server, Linux-HA - director-based clusters that allow incoming requests for services to be distributed across multiple cluster nodes. MOSIX, openMosix, Kerrighed, OpenSSI are full-blown clusters integrated into the kernel that provide for automatic process migration among homogeneous nodes. OpenSSI, openMosix and Kerrighed are single-system image implementations.

Microsoft Windows Compute Cluster Server 2003 based on the Windows Server platform provides pieces for High Performance Computing like the Job Scheduler, MSMPI library and management tools. NCSA's recently installed Lincoln is a cluster of 450 Dell PowerEdge 1855 blade servers running Windows Compute Cluster Server 2003. This cluster debuted at #130 on the Top500 list in June 2006.

gridMathematica provides distributed computations over clusters including data analysis, computer algebra and 3D visualization. It can make use of other technologies such as Altair PBS Professional, Microsoft Windows Compute Cluster Server, Platform LSF and Sun Grid Engine.

gLite is a set of middleware technologies created by the Enabling Grids for E-sciencE (EGEE) project.

Another example of consumer game products being added to high-performance computing is the Nvidia Tesla Personal Supercomputer workstation, which gets its processing power by harnessing the power of multiple graphics accelerator processor chips.

Algorithmic Skeletons are a high-level parallel programming model for parallel and distributed computing which take advantage of common programming patterns to hide the complexity of parallel and distributed applications. Starting from a basic set of patterns (skeletons), more complex patterns can be built by combining the basic ones.

Global Storage Architecture (GSA) – a highly scalable cloud based NAS solution - combines proprietary IBM HPC technology (storage and server hardware and IBM's high-performance shared-disk clustered file system - GPFS) with open source components like Linux, Samba and CTDB to deliver distributed storage solutions. GSA exports the clustered file system through industry standard protocols like CIFS, NFS, FTP and HTTP. All of the GSA nodes in the grid export all files of all file systems simultaneously.

Chapter-2

Beowulf (Computing)

The Borg, a 52-node Beowulf cluster used by the McGill University pulsar group to search for pulsations from binary pulsars.

Originally referring to a specific computer built in 1994, **Beowulf** is a class of computer clusters similar to the original NASA system. Originally developed by Thomas Sterling and Donald Becker at NASA, Beowulf systems are now deployed worldwide, chiefly in support of scientific computing. They are high-performance parallel computing clusters of inexpensive personal computer hardware. The name comes from the main character in the Old English poem *Beowulf*, which was bestowed by Sterling because the epic poem describes Beowulf as having "thirty mens' heft of grasp in the grip of his hand."

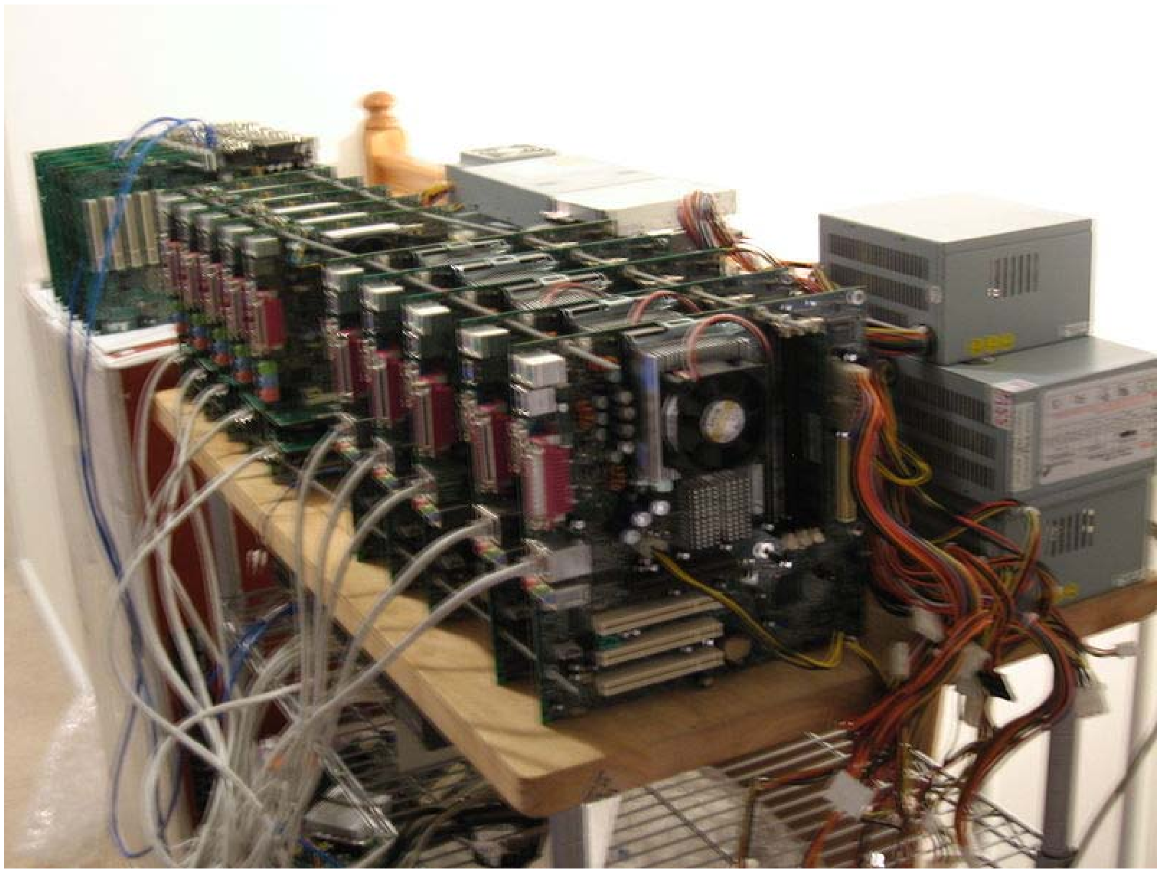
A *Beowulf cluster* is a group of what are normally identical, commercially available computers, which are running a Free and Open Source Software (FOSS), Unix-like operating system, such as BSD, GNU/Linux, or Solaris. They are networked into a small TCP/IP LAN, and have libraries and programs installed which allow processing to be shared among them.

There is no particular piece of software that defines a cluster as a Beowulf. Commonly used parallel processing libraries include Message Passing Interface (MPI) and Parallel Virtual Machine (PVM). Both of these permit the programmer to divide a task among a group of networked computers, and collect the results of processing. Examples of MPI software include OpenMPI (OpenMPI) or MPICH (MPICH). There are additional MPI implementations available.

Provisioning of Operating System and other software for a Beowulf Cluster can be automated using software, Open Source Cluster Application Resources (OSCAR) for example. OSCAR installs on top of a standard installation of a supported GNU/Linux distribution on a cluster's head node.

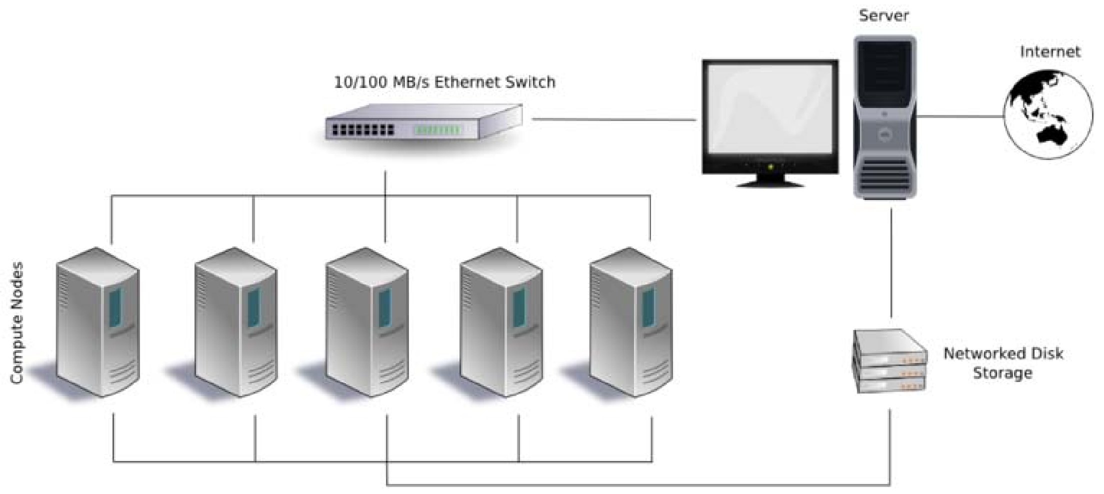
Development

The following is the definition of a Beowulf cluster from the original how-to which was published by Jacek Radajewski and Douglas Eadline under the Linux Documentation Project in 1998.



An illustration of the headless and diskless nature of a beowulf cluster, in the TinyHPC cluster

Beowulf is a multi-computer architecture which can be used for parallel computations. It is a system which usually consists of one server node, and one or more client nodes connected together via Ethernet or some other network. It is a system built using commodity hardware components, like any PC capable of running a Unix-like operating system, with standard Ethernet adapters, and switches. It does not contain any custom hardware components and is trivially reproducible. Beowulf also uses commodity software like the FreeBSD, GNU/Linux or Solaris operating system, Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). The server node controls the whole cluster and serves files to the client nodes. It is also the cluster's console and gateway to the outside world. Large Beowulf machines might have more than one server node, and possibly other nodes dedicated to particular tasks, for example consoles or monitoring stations. In most cases client nodes in a Beowulf system are dumb, the dumber the better. Nodes are configured and controlled by the server node, and do only what they are told to do. In a disk-less client configuration, a client node doesn't even know its IP address or name until the server tells it.



The typical setup of a beowulf cluster

One of the main differences between Beowulf and a Cluster of Workstations (COW) is that Beowulf behaves more like a single machine rather than many workstations. In most cases client nodes do not have keyboards or monitors, and are accessed only via remote login or possibly serial terminal. Beowulf nodes can be thought of as a CPU + memory package which can be plugged in to the cluster, just like a CPU or memory module can be plugged into a motherboard.

Beowulf is not a special software package, new network topology, or the latest kernel hack. Beowulf is a technology of clustering computers to form a parallel, virtual supercomputer. Although there are many software packages such as kernel modifications, PVM and MPI libraries, and configuration tools which make the Beowulf architecture faster, easier to configure, and much more usable, one can build a Beowulf class machine using standard GNU/Linux distribution without any additional software. If you have two networked computers which share at least the `/home` file system via NFS, and trust each other to execute remote shells (rsh), then it could be argued that you have a simple, two node Beowulf machine.

Operating systems

A home-built Beowulf cluster

Currently there are a number of GNU/Linux distributions, and at least one BSD, that are designed for building Beowulf clusters. These include:

- KestrelHPC
- ABC Linux, based on Ubuntu - Allows the building of Beowulf clusters automatically, either live or by installing the software in the front end. All nodes run diskless.
- MOSIX is geared toward computationally-intensive, IO-low, applications.
- ClusterKnoppix, based on Knoppix - Last updated August 31, 2004
- Kerrighed

- ParallelKnoppix, Also based on Knoppix - Last updated May 29, 2008
- PelicanHPC (Successor to parallelknoppix based on Debian Live) - Last updated January 18, 2011
- Rocks Cluster Distribution
- Scyld
- DragonFly BSD
- Bootable Cluster CD
- Quantian, Live DVD with scientific applications, based on Knoppix and ClusterKnoppix. Last updated February 26, 2006.

A cluster can be set up by using Knoppix bootable CDs in combination with OpenMosix. The computers will automatically link together, without need for complex configurations, to form a Beowulf cluster using all CPUs and RAM in the cluster. A Beowulf cluster is scalable to a nearly unlimited number of computers, limited only by the overhead of the network.

Chapter-3

ProAction and LOCUS (Operating System)

ProActive

	ProActive
Developer(s)	OW2 Consortium
Written in	Java
Operating system	Cross-platform
Type	Grid Computing
License	GNU General Public License

ProActive is Java grid middleware for parallel, distributed, and multi-threaded computing. It is developed by the OW2 Consortium, including INRIA, CNRS, University of Nice Sophia Antipolis, and ActiveEon. It is open-source software released under the GPL license.

ProActive provides a comprehensive framework and parallel programming model to simplify the programming and execution of parallel applications running on multi-core processors, distributed on Local Area Network (LAN), on clusters and data centers, on intranets, and on Internet grids.

The ProActive programming model combines the active object design pattern with futures objects.

Programming model

The model was created by Denis Caromel, professor at University of Nice Sophia Antipolis. Several extensions of the model were made later on by members of the OASIS team at INRIA. The book *A Theory of Distributed Objects* presents the ASP calculus that formalizes ProActive features, and provides formal semantics to the calculus, together with properties of ProActive program execution.

Active objects

Active objects are the basic units of activity and distribution used for building concurrent applications using ProActive. An active object runs with its own thread. This thread only executes the methods invoked on this active object by other active objects, and those of the passive objects of the subsystem that belongs to this active object. With ProActive, the programmer does not have to explicitly manipulate Thread objects, unlike in standard Java.

Active objects can be created on any of the hosts involved in the computation. Once an active object is created, its activity (the fact that it runs with its own thread) and its location (local or remote) are perfectly transparent. Any active object can be manipulated as if it were a passive instance of the same class.

An *active object* is composed of two objects: a *body*, and a standard Java object. The body is not visible from the outside of the active object.

The body is responsible for receiving calls (or *requests*) on the active object and storing them in a queue of pending calls. It executes these calls in an order specified by a synchronization policy. If a synchronization policy is not specified, calls are managed in a "First in, first out" (FIFO) manner.

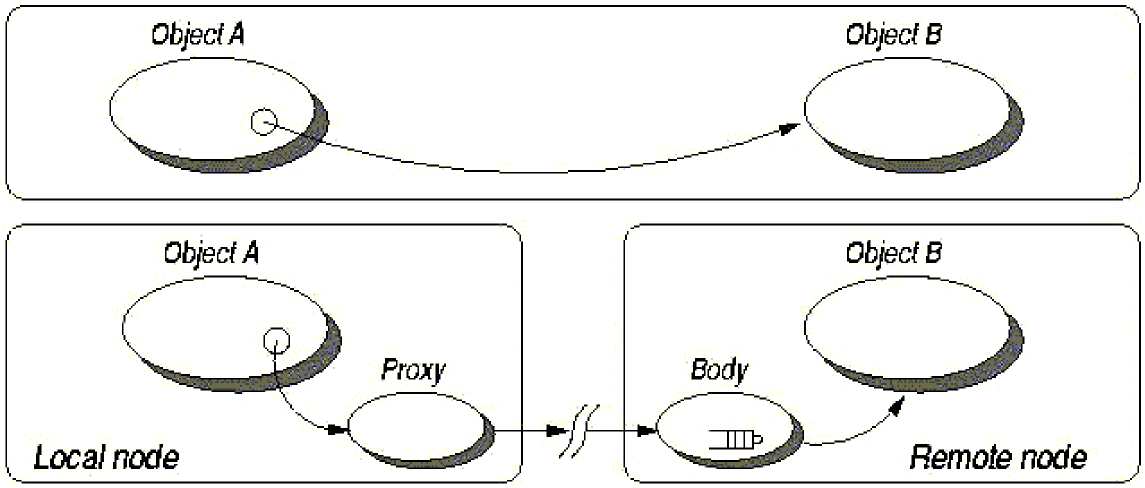
The thread of an active object then chooses a method in the queue of pending requests and executes it. No parallelism is provided inside an active object; this is an important decision in ProActive's design, enabling the use of "pre-post" conditions and class invariants.

On the side of the subsystem that sends a call to an active object, the active object is represented by a *proxy*. The proxy generates future objects for representing future values, transforms calls into Request objects (in terms of metaobject, this is a reification) and performs deep copies of passive objects passed as parameters.

Active object basis

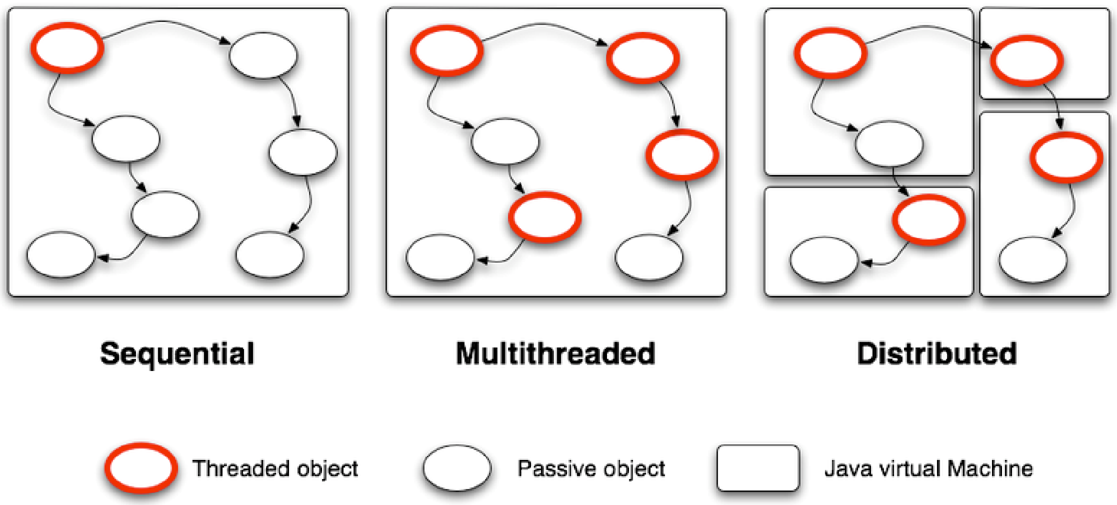
ProActive is a library designed for developing applications in the model introduced by Eiffel//, a parallel extension of the Eiffel programming language.

In this model, the application is structured in *subsystems*. There is one active object (and therefore one thread) for each subsystem, and one subsystem for each active object (or thread). Each subsystem is thus composed of one active object and any number of passive objects—possibly no passive objects. The thread of one subsystem only executes methods in the objects of this subsystem. There are no "shared passive objects" between subsystems.



A call onto an active object, as opposed to a call onto passive one

These features impact the application's topology. Of all the objects that make up a subsystem—the active object and the passive objects—only the active object is known to objects outside of the subsystem. All objects, both active and passive, may have references onto active objects. If an object *o1* has a reference onto a passive object *o2*, then *o1* and *o2* are part of the same subsystem.



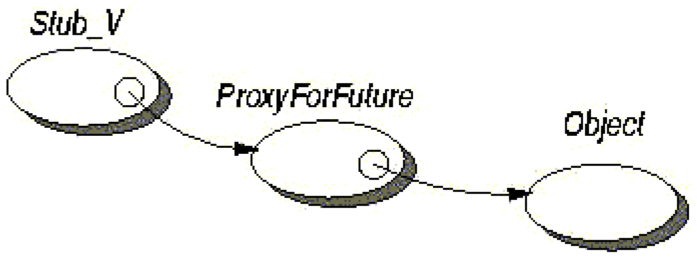
The model: Sequential, multithreaded, distributed

This has also consequences on the semantics of message-passing between subsystems. When an object in a subsystem calls a method on an active object, the parameters of the call may be references on passive objects of the subsystem, which would lead to shared passive objects. This is why passive objects passed as parameters of calls on active objects are always passed by deep-copy. Active objects, on the other hand, are always passed by reference. Symmetrically, this also applies to objects returned from methods called on active objects.

Thanks to the concepts of asynchronous calls, futures, and no data sharing, an application written with ProActive doesn't need any structural change—actually, hardly any change at all—whether it runs in a sequential, multi-threaded, or distributed environment.

Asynchronous calls and futures

Whenever possible, a method call on an active object is reified as an asynchronous request. If not possible, the call is synchronous, and blocks until the reply is received. If the request is asynchronous, it immediately returns a *future object*.



A future object

The future object acts as a placeholder for the result of the not-yet-performed method invocation. As a consequence, the calling thread can go on with executing its code, as long as it doesn't need to invoke methods on the returned object. If the need arises, the calling thread is automatically blocked if the result of the method invocation is not yet available. Although a future object has structure similar to that of an active object, a future object is not active. It only has a Stub and a Proxy.

A simple example

The code excerpt below highlights the notion of future objects. Suppose a user calls a method `foo` and a method `bar` from an active object `a`; the `foo` method returns void and the `bar` method returns an object of class `V`:

```
// a one way typed asynchronous communication towards the (remote) AO a
// a request is sent to a
a.foo (param);

// a typed asynchronous communication with result.
// v is first an awaited Future, to be transparently filled up after
// service of the request, and reply
V v = a.bar (param);
...
// use of the result of an asynchronous call.
// if v is still an awaited future, it triggers an automatic
// wait: Wait-by-necessity
v.gee (param);
```

When `foo` is called on an active object `a`, it returns immediately (as the current thread cannot execute methods in the other subsystem). Similarly, when `bar` is called on `a`, it returns immediately but the result `v` can't be computed yet. A future object, which is a placeholder for the result of the method invocation, is returned. From the point of view of the caller subsystem, there is no difference between the future object and the object that would have been returned if the same call had been issued onto a passive object.

After both methods have returned, the calling thread continues executing its code as if the call had been effectively performed. The role of the future mechanism is to block the caller thread when the `gee` method is called on `v` and the result has not yet been set : this inter-object synchronization policy is known as *wait-by-necessity*.

LOCUS (operating system)

	LOCUS
Company / developer	UCLA
OS family	Unix
Working state	Historic
Source model	Closed source
Kernel type	Monolithic kernel
License	Proprietary

LOCUS was a distributed operating system developed at UCLA during the 1980s. It was notable for providing an early implementation of the single-system image idea, where a cluster of machines appeared to be one larger machine.

A desire to commercialize the technologies developed for LOCUS inspired the creation of the Locus Computing Corporation which went on to include ideas from LOCUS in various products, including OSF/1 AD and, finally, the SCO–Tandem UnixWare NonStop Clusters product.

Description

The LOCUS system was created at UCLA between 1980 and 1983, initial implementation was on a cluster of PDP-11/45s using 1 and 10 megabit ring networks, by 1983 the system was running on 17 VAX-11/750s using a 10 megabit Ethernet. The system was Unix compatible and provided both a single root view of the file system and a unified process space across all nodes.

The development of LOCUS was supported by an ARPA research contract, DSS-MDA-903-82-C-0189.

File system

In order to allow reliable and rapid access to the cluster wide filesystem LOCUS used replication, the data of files could be stored on more than one node and LOCUS would keep the various copies up to date. This provided particularly good access times for files that were read more often than they were written, the normal case for directories for example.

In order to ensure that all access was made to the most recent version of any file LOCUS would nominate one node as the "current synchronization site" (CSS) for a particular file system. All accesses to files a file system would need to be coordinated with the appropriate CSS.

Node dependent files

As with other SSI systems LOCUS sometimes found it necessary to *break the illusion* of a single system, notably to allow some files to be different on a per-node basis. For example it was possible to build a LOCUS cluster containing both PDP-11/45 and VAX 750 machines, but instruction sets used were not identical, so two versions of each object program would be needed

The solution was to replace the files that needed to be different on a per node basis by special hidden directories. These directories would then contain the different versions of the file. When a user accessed one of these hidden directories the system would check the users *context* and open the appropriate file.

For example, if the user was running on one of the PDP-11/45's and typed the command `/bin/who` then the system would find that `/bin/who` was actually a hidden directory and run the command `/bin/who/45`. Another user on a VAX node who typed `/bin/who` would run the command `/bin/who/vax`.

Devices

LOCUS provided remote access to I/O devices.

Processes

LOCUS provided a single process space. Processes could be created on any node on the system. Both the Unix fork and exec calls would examine an *advice list* which determined on which node the process would be run. LOCUS was designed to work with heterogeneous nodes, (e.g. a mix of VAX 750s and PDP 11/45s) and could decide to execute a process on a different node if it needed a particular instruction set. As an optimization a *run* call was added which was equivalent to a combined fork and exec, thus avoiding the overhead of copying the process memory image to another node before overwriting it by the new image.

Pipes

Processes could use pipes for inter node communication, including named pipes,

Partitioning

The LOCUS system was designed to be able to cope with network partitioning - one or more nodes becoming disconnected from the rest of the system. As the file system was replicated the disconnected nodes could continue to access files. When the nodes were reconnected any files modified by the disconnected nodes would be merged back into the system. For some file types (for example mailboxes) the system would perform the merge automatically, for others the user would be informed (by mail) and tools were provided to allow access to the different versions of the file.

Chapter-4

OpenSSI and MOSIX

OpenSSI

OpenSSI is an open source single-system image clustering system. It allows a collection of computers to be treated as one large system, allowing applications running on any one machine access to the resources of all the machines in the cluster.

OpenSSI is based on the Linux operating system and was released as an open source project by Compaq in 2001. It is the final stage of a long process of development, stretching back to LOCUS, developed in the early 1980s.

Description

OpenSSI allows a cluster of individual computers (*nodes*) to be treated as one large system. Processes run on any node have full access to the resources of all nodes. Processes can be migrated from node to node automatically to balance system utilization. Inbound network connections can be directed to the least loaded node available.

OpenSSI is designed to be used for both high performance and high availability clusters, it is possible to create an OpenSSI cluster with no single point of failure, for example the file system can be mirrored between two nodes, so if one node crashes the process accessing the file will *fail over* to the other node. Alternatively the cluster can be designed in such a manner that every node has direct access to the file system.

Features

Single Process space

OpenSSI provides a single process space - every process is visible from every node, and can be managed from any node using the normal Linux commands (ps, kill, renice and so on). The Linux /proc virtual filesystem shows all running processes on all nodes.

The implementation of the single process space is accomplished using the VPROC abstraction invented by Locus for the OSF/1 AD operating system.

Migration

OpenSSI allows migration of running processes between nodes. When running processes are migrated they continue to have access to any open files, IPC objects or network connections.

Processes can be *manually* migrated, either by the process calling the special OpenSSI *migrate(2)* system call, or by writing a node number to a special file in the processes /proc directory.

Processes may also, if the user wants, be automatically migrated in order to balance load across the cluster. OpenSSI uses an algorithm developed by the MOSIX project for determining the load on each node.

Single root

OpenSSI provides a single root for the cluster - from any node the same files and directories are available. OpenSSI uses several mechanisms to provide the single root - CFS (the OpenSSI Cluster File System), SAN cluster filesystems and parallel mounts of network file systems.

OpenSSI uses the context dependent symbolic link (CDSL) feature, inspired by HP's TruCluster system, to allow access to node-specific files in a manner transparent to non cluster-aware applications. A CDSL may point to different files on each node in the cluster.

CFS

CFS, the OpenSSI Cluster File System provides transparent inter-node access to an underlying *real* file system on one node.

CFS is *stacked* on top of the real file system and co-ordinates accesses from different nodes using a *token* mechanism. One node has physical access to the underlying file system and performs all read and write operations. At any one time one node *owns* a token, representing a part of the underlying file, this implies that that part of the file is in the cache of the owning node. If another node tries to access that part of the file the token is *stolen* and the cache contents are copied to the stealing node. The OpenSSI CFS implementation is remarkably similar to that used by HP TruCluster.

CFS is also used to co-ordinate access to shared memory segments.

CFS can be used in a fault tolerant system by using shared disk subsystems (dual ported SCSI or SAN), or by using DRBD. If the node that is currently directly accessing the file system crashes then the CFS mount *fails over* to the other node that is directly connected to the disk and the cluster now accesses the file system via that node.

SAN clustered file systems

OpenSSI can use SAN based clustered file systems for its root provided they provide a POSIX compatible file system interface. Currently Lustre and GFS have been tested.

With a clustered file system, each node mounts the file system in parallel and access to the files goes directly from the node to the file system.

NFS

OpenSSI mounts NFS files systems in parallel on each node. Every node accesses the NFS server directly.

Single I/O space

OpenSSI provides cluster-wide access to all I/O devices on the system, with some limitations - it is not possible for a node to mount a block device from another node.

The udev device manager is used to manage the /dev directory. Each node runs its own copy of udev to create the appropriate device nodes in a subdirectory of /dev, /dev/1 for node 1, /dev/2 for node 2 and so on.

Single IPC space

OpenSSI provides internode access to all the standard Linux inter-process communication mechanisms, shared memory, semaphores, SYSV message queues, pipes and Unix domain sockets.

In order to implement cluster wide shared memory - distributed shared memory - OpenSSI uses the CFS *token* system. At any one time a memory segment may be readable by one or more nodes, or writable by one node. If a node without write access to a segment tries to write then the segment is marked unreadable on all other nodes and writable on the current node. If a node without read access tries to read a segment then the current value is copied from a node where it was valid and if it was writable it is marked readable.

Cluster IP address

OpenSSI uses LVS to provide fault-tolerant load balanced IP services. Inbound network connections are received by a *director* node which redirects them to the least loaded server node. (A node may be both a director and server). In the event of director node failure another director node takes over and the system continues to accept inbound connections.

Distributions

The OpenSSI software is available for various Linux distributions. The OpenSSI kernel is distribution independent but various distribution specific Linux user level systems need to be modified, for example the init process and the system startup scripts.

Currently the supported distributions are:

1. Fedora Core 3
2. Debian Sarge

Work is in progress to port OpenSSI to Debian Etch and Lenny

History

The origins of OpenSSI date back to the early 1980s when the LOCUS distributed operating system was developed at UCLA. The team that developed LOCUS went on to form the Locus Computing Corporation and produced various versions of the LOCUS technology under several names, culminating in the development of the UnixWare NonStop Clusters product at Tandem Computers, which had by that time acquired the LOCUS team and rights to the technology. NonStop Clusters for Unixware was commercialized by SCO as an add-on for UnixWare. When SCO stopped selling NonStop Clusters, the former Locus team, now working for Compaq (which had acquired Tandem in the interim), ported the NonStop Clusters code to Linux and released it as open source. The team at Compaq continued to develop the system, now called OpenSSI, for some time after HP acquired Compaq. OpenSSI is currently developed by an independent team.

MOSIX

MOSIX is a distributed operating system. Although early versions were based on older UNIX systems, since 1999 it focuses on Linux clusters and grids. In a MOSIX cluster/grid there is no need to modify or to link applications with any library, to copy files or login to remote nodes, or even to assign processes to different nodes – it is all done automatically, like in an SMP.

History

MOSIX has been researched and developed since 1977 at The Hebrew University of Jerusalem by the research team of Prof. Amnon Barak. So far, ten major versions have been developed. The first version, called MOS, for *Multicomputer OS*, (1981-83) was based on Bell Lab's Seventh Edition Unix and ran on a cluster of PDP-11 computers. Later versions were based on Unix System V Release 2 (1987-89) and ran on a cluster of VAX and NS32332-based computers, followed by a BSD/OS-derived version (1991-93) for a cluster of 486/Pentium computers. Since 1999 MOSIX is tuned to Linux for x86 platforms.

MOSIX2

The latest version of MOSIX, called MOSIX2, is compatible with Linux-2.6. MOSIX2 is implemented as an OS virtualization layer that provides to users and applications a single system image with the Linux run-time environment. It allows applications to run in remote nodes as if they run locally. Users run their regular (sequential and parallel) applications while MOSIX transparently and automatically seek resources and migrate processes among nodes to improve the overall performance.

MOSIX2 can manage a cluster and a multicluster (grid) as well as workstations and other shared resources. Flexible management of a grid allows owners of clusters to share their computational resources, while still preserving their autonomy over their own clusters and their ability to disconnect their nodes from the grid at any time, without disrupting already running programs.

A MOSIX grid can extend indefinitely as long as there is trust between its cluster owners. This must include guarantees that guest applications will not be modified while running in remote clusters and that no hostile computers can be connected to the local network. Nowadays these requirements are standard within clusters and organizational grids.

MOSIX2 can run in native mode or in a virtual machine (VM). In native mode, performance is better, but it requires modifications to the base Linux kernel, whereas a VM can run on top of any unmodified operating system that supports virtualization, including Microsoft Windows, Linux and Mac OS X.

MOSIX2 is most suitable for running compute intensive applications with low to moderate amount of input/output (I/O). Tests of MOSIX2 show that the performance of several such applications over a 1 Gb/s campus grid is nearly identical to that of a single cluster.

Main features

- Provides aspects of a single-system image:
    - Users can login on any node and do not need to know where their programs run.
    - No need to modify or link applications with special libraries.
    - No need to copy files to remote nodes.
- Automatic resource discovery and workload distribution by process migration:
    - Load-balancing.
    - Migrating processes from slower to faster nodes and from nodes that run out of free memory.
- Migratable sockets for direct communication between migrated processes.
- Secure run time environment (sandbox) for guest processes.

- Live queuing – queued jobs preserve their full generic Linux environment.
- Batch jobs.
- Checkpoint and recovery.
- Tools: automatic installation and configuration scripts, on-line monitors.

MOSIX for HPC

MOSIX is most suitable for running HPC applications with low to moderate amount of I/O. Tests of MOSIX show that the performance of several such applications over a 1 Gb/s campus grid is nearly identical to that of a single cluster. It is particularly suitable for:

- Efficient utilization of grid-wide resources, by automatic resource discovery and load-balancing.
- Running applications with unpredictable resource requirements or run times.
- Running long processes, which are automatically sent to grid nodes and are migrated back when these nodes are disconnected from the grid.
- Combining nodes of different speeds, by migrating processes among nodes based on their respective speeds, current load, and available memory.

A few examples:

- Scientific applications – genomic, protein sequences, molecular dynamics, quantum dynamics, nano-technology and other parallel HPC applications.
- Engineering applications – CFD, weather forecasting, crash simulations, oil industry, ASIC design, pharmaceutical and other HPC applications.
- Financial modeling, rendering farms, compilation farms.

openMosix

After MOSIX became proprietary software in late 2001, Moshe Bar forked the last free version and started the openMosix project on February 10, 2002. On July 15, 2007, Bar plans to end the openMosix project effective March 1, 2008, claiming that "the increasing power and availability of low cost multi-core processors is rapidly making single-system image (SSI) clustering less of a factor in computing". These plans were reconfirmed in March 2008. The LinuxPMI project is continuing development of the former openMosix code.

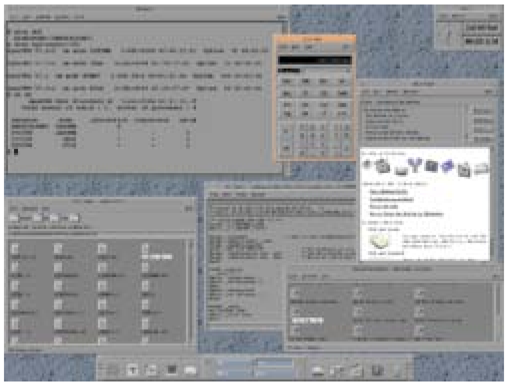
Chapter-5

OpenVMS

OpenVMS

New logo

Old logo



OpenVMS V7.3-1 running the CDE-based

	DECwindows GUI
Company / developer	Digital Equipment Corporation, Hewlett-Packard
Programmed in	BLISS, VAX Macro, C, Ada, PL/I, Fortran, UIL, SDL, Pascal, MDL, C++, DCL, Message, Document
OS family	DEC OS family
Working state	Current
Source model	Closed source
Latest stable release	OpenVMS 8.4 / June 21, 2010; 9 months ago
Marketing target	High-end computer server
Available language(s)	English
Available programming languages(s)	Ada, BASIC, BLISS, C, C++, COBOL, DIBOL, DCL, Fortran, Lisp, MACRO32/64, Modula-2, OPS5, Pascal, Perl, Python, PL/I, Java
Update method	Concurrent Upgrades, Rolling Upgrades
Package manager	PCSI and VMSINSTAL
Supported platforms	VAX, Alpha, Itanium
Kernel type	Monolithic kernel with loadable modules
Default user interface	DCL CLI and DECwindows GUI
License	Proprietary

OpenVMS (**Open Virtual Memory System**), previously known as **VAX-11/VMS**, **VAX/VMS** or (informally) **VMS**, is a high-end computer server operating system that runs on VAX, Alpha and Itanium-based families of computers. Contrary to what its name suggests, OpenVMS is not open source software; however, the source listings are available for purchase. Unlike some other mainframe-oriented operating systems, OpenVMS has a graphical user interface (GUI) with complete graphics support. Digital Equipment Corporation's VAX was one of the three top-selling workstations lines in the 1980s and 1990s. VMS had support for professional DTP and CAE software running.

AXP VMS supported OpenGL and Accelerated Graphics Port (AGP) graphics adapters. It has also been used in education and for home hobbyist use.

OpenVMS is a multi-user, multiprocessing virtual memory-based operating system (OS) designed for use in time sharing, batch processing, real-time (where process priorities can be set higher than OS kernel jobs), and transaction processing. It offers high system availability through clustering, or the ability to distribute the system over multiple physical machines. This allows the system to be "disaster-tolerant" against natural disasters that may disable individual data-processing facilities. VMS also includes a process priority system that allows for real-time processes to run unhindered, while user processes get temporary priority "boosts" if necessary.

OpenVMS commercialized many features that are now considered standard requirements for any high-end server operating system. These include:

- Integrated computer networking (originally DECnet and later, TCP/IP)
- Symmetrical, asymmetrical, and NUMA multiprocessing, including clustering
- A distributed file system (Files-11)
- Integrated database features such as RMS and layered databases including Rdb
- Support for multiple computer programming languages
- A standardized interoperability mechanism for calls between different programming languages
- An extensible shell command language (DIGITAL Command Language)
- Hardware partitioning of multiprocessors
- High level of security

Enterprise-class environments typically select and use OpenVMS for various purposes including as a mail server, network services, manufacturing or transportation control and monitoring, critical applications and databases, and particularly environments where system uptime and data access is critical. System up-times of a decade or more have been reported, and features such as Rolling Upgrades and clustering allow clustered applications and data to remain continuously accessible while operating system software and hardware maintenance and upgrades are performed, or when a whole data center is destroyed. Customers using OpenVMS include banks and financial services, hospitals and healthcare, network information services, and large-scale industrial manufacturers of various products.

History

Origin and name changes

In April 1975, Digital Equipment Corporation embarked on a hardware project, code named *Star*, to design a 32-bit virtual address extension to its PDP-11 computer line. A companion software project, code named *Starlet*, was begun in June 1975 to develop a totally new operating system, based on RSX-11M, for the Star family of processors. These two projects were tightly integrated from the beginning. Gordon Bell was the VP lead on the VAX hardware and its architecture. Roger Gourd was the project lead for the Starlet program, with software engineers Dave Cutler (who would later lead development of Microsoft's Windows NT), Dick Hustvedt, and Peter Lippman acting as the technical project leaders, each having responsibility for a different area of the operating system. The Star and Starlet projects culminated in the VAX 11/780 computer and the VAX-11/VMS operating system. The Starlet name survived in VMS as a name of several of the main system libraries, including STARLET.OLB and STARLET.MLB.

Over the years the name of the product has changed. In 1980 it was renamed, with version 2.0 release, to VAX/VMS (at the same time as the VAX-11 computer was renamed to simply VAX). With the introduction of the MicroVAX range such as the MicroVAX I, MicroVAX II and MicroVAX 2000 in the mid-to-late 1980s, DIGITAL released MicroVMS versions specifically targeted for these platforms which had much more limited memory and disk capacity; e.g. the smallest MicroVAX 2000 had a 40MB RD32 hard disk and a maximum of 6MB of RAM, and its CPU had to emulate some of the VAX floating point instructions in software. MicroVMS kits were released for VAX/VMS 4.4 to 4.7 on TK50 tapes and RX50 floppy disks, but discontinued with VAX/VMS 5.0. In 1991 it was renamed again to OpenVMS to indicate its support for industry standards such as POSIX and Unix compatibility, and to drop the hardware connection as the port to DIGITAL's 64-bit Alpha RISC processor was in process. The OpenVMS name first appeared after the version 5.4-2 release.

Port to DEC Alpha

The VMS port to Alpha resulted in the creation of a second and separate source code libraries (based on a source code management tool known as VDE) for the VAX 32-bit source code library and a second and new source code library for the Alpha (and the subsequent Itanium port) 64-bit architectures. 1992 saw the release of the first version of OpenVMS for Alpha AXP systems, designated *OpenVMS AXP V1.0*. The decision to use the 1.x version numbering stream for the pre-production quality releases of OpenVMS AXP caused confusion for some customers and was not repeated in the next platform port to the Itanium.

In 1994, with the release of OpenVMS version 6.1, feature (and version number) parity between the VAX and Alpha variants was achieved. This was the so-called Functional Equivalence release, in the marketing materials of the time. Some features were missing however, e.g. based shareable images, which were implemented in later versions. Subsequent version numberings for the VAX and Alpha variants of the product have remained consistent through V7.3, though Alpha subsequently diverged with the availability of the V8.2 and V8.3 releases.

Port to Intel Itanium

In 2001, just prior to its acquisition by Hewlett-Packard, Compaq announced the port of OpenVMS to the Intel Itanium architecture. This port was accomplished using source code maintained in common within the OpenVMS Alpha source code library, with conditional and additional modules where changes specific to Itanium were required. The OpenVMS Alpha pool was chosen as the basis of the port as it was significantly more portable than the original OpenVMS VAX source code, and because the Alpha source code pool was already fully 64-bit capable (unlike the VAX source code pool). With the Alpha port, many of the VAX hardware-specific dependencies had been previously moved into the Alpha SRM firmware for OpenVMS. Features necessary for OpenVMS were then moved from SRM into OpenVMS I64 as part of the Itanium port.

Unlike the port from VAX to Alpha, in which a snapshot of the VAX code base circa V5.4-2 was used as the basis for the Alpha release and the 64-bit source code pool then diverged, the OpenVMS Alpha and I64 (Itanium) versions of OpenVMS are built and maintained using a common source code library and common tools. The core software source code control system used for OpenVMS is the VMS Development Environment (VDE).

Two pre-production releases, OpenVMS I64 V8.0 and V8.1, were available on June 30, 2003 and on December 18, 2003. These releases were intended for HP organizations and third-party vendors involved with porting software packages to OpenVMS I64.

The following are recent OpenVMS I64 releases:

OpenVMS I64 V8.2, the first production-quality Itanium release, was shipped January 13, 2005. A V8.2 release is also available for Alpha platforms.

OpenVMS I64 V8.2-1, adding support for Integrity Superdome and cell based systems, was released in September 2005. V8.2-1 is available for Itanium platforms only.

OpenVMS I64 V8.3, was released for Itanium platforms in September 2006. V8.3 is also available for Alpha systems.

OpenVMS I64 V8.3-1H1, was released in October 2007. It features full c-Class Integrity BladeServer blade support.

OpenVMS I64 and Alpha V8.4, was released in June 2010.

Major release timeline

Date	Version	Note
October 25, 1977	V1.0	Initial commercial release
April, 1980	V2.0	VAX-11/750
April, 1982	V3.0	VAX-11/730
September, 1984	V4.0	VAX 8600 and MicroVMS (for MicroVAX)

April, 1988	V5.0	VAX 6000
November, 1992	V1.0	first OpenVMS AXP (Alpha) specific version
June, 1993	V6.0	VAX 7000 and 10000
April/May, 1994	V6.1	merging of VAX and Alpha AXP version numbers
January, 1996	V7.0	full 64-bit virtual addressing on Alpha
1997	V7.1	
June, 2003	V8.0	limited availability eval for Integrity
February, 2005	V8.2	Common Alpha and Itanium release
September, 2006	V8.3	Alpha, Itanium dual-core support
October, 2007	V8.3-1H1	c-Class Integrity blade server support
June, 2010	V8.4	Added support for running as a virtual machine guest under HPVM

Features

Graphical user interface

OpenVMS uses the DECwindows Motif user interface (based on CDE) layered on top of OpenVMS's X11 compliant windowing system.

Clustering

OpenVMS supports clustering (first called *VAXcluster* and later VMScluster), where multiple systems share disk storage, processing, job queues and print queues, and are connected either by specialized hardware or an industry-standard LAN (usually Ethernet). A LAN-based cluster is often called a LAVc, for Local Area Network VMScluster, and allows, among other things, bootstrapping a possibly diskless *satellite node* over the network using the system disk of a *bootnode*.

VAXcluster support was first added in VMS version 4, which was released in 1984. This version only supported clustering over CI. Later releases of version 4 supported clustering over LAN (LAVC), and support for LAVC was improved in VMS version 5, released in 1988.

Mixtures of cluster interconnects and technologies are permitted, including Gigabit (GbE) Ethernet, SCSI, FDDI, DSSI, CI and Memory Channel adapters.

OpenVMS supports up to 96 nodes in a single cluster, and allows mixed-architecture clusters, where VAX and Alpha systems, or Alpha and Itanium systems can co-exist in a single cluster (Various organizations have demonstrated triple-architecture clusters and cluster configurations with up to 150 nodes, but these configurations are not supported by HP).

Unlike many other clustering solutions, VAXcluster offers transparent and fully distributed read-write with record-level locking, which means that the same disk and even the same file can be accessed by several cluster nodes at once; the locking occurs only at the level of a single record of a file, which would usually be one line of text or a single record in a database. This allows the construction of high-availability multiply redundant database servers.

Cluster interconnections can span upwards of 500 miles, allowing member nodes to be located in different buildings on an office campus, or in different cities.

Host-based volume shadowing allows volumes (of the same or of different sizes) to be shadowed (mirrored) across multiple controllers and multiple hosts, allowing the construction of disaster-tolerant environments.

Full access into the distributed lock manager (DLM) is available to application programmers, and this allows applications to coordinate arbitrary resources and activities across all cluster nodes. This obviously includes file-level coordination, but the resources and activities and operations that can be coordinated with the DLM are completely arbitrary.

With the supported capability of rolling upgrades and with multiple system disks, cluster configurations can be maintained on-line and upgraded incrementally. This allows cluster configurations to continue to provide application and data access while a subset of the member nodes are upgraded to newer software versions.

File system

OpenVMS has a very feature-rich file system, with support for stream and record-oriented IO, ACLs, and file versioning. The typical user and application interface into the file system is the RMS.

Details are in the RMS Utilities and RMS programming manuals, and in the I/O User's Reference Manual, all part of the OpenVMS documentation set.

Timekeeping

OpenVMS represents system time as the 64-bit number of 100 nanosecond intervals (that is, ten million units per second; also known as a 'clunk') since the epoch. The epoch of OpenVMS is midnight preceding November 17, 1858, which is the start of Modified Julian Day numbering. The clock is not necessarily updated every 100 ns; for example, systems with a 100 Hz interval timer simply add 100 000 to the value every hundredth of a second. The operating system includes a mechanism to adjust for hardware timekeeping drift; when calibrated against a known time standard, it easily achieves an accuracy better than 0.01%. All OpenVMS hardware platforms derive timekeeping from an internal clock not associated with the AC supply power frequency.

While the system is shut down, time is kept by a Time-of-Year ("TOY") hardware clock. This clock keeps time to a lower resolution (perhaps 1 second) and generally, a lower accuracy (often 0.025% versus 0.01%). When the system is restarted, the VMS 64-bit time value is recomputed based on the time kept by the TOY clock and the last recorded year (stored on the system disk).

The 100 nanosecond granularity implemented within OpenVMS and the 63-bit absolute time representation (the sign bit indicates *absolute time* when clear and *relative time* when set) should allow OpenVMS trouble-free time computations up to **31-JUL-31086 02:48:05.47**. At this instant, all clocks and time-keeping operations in OpenVMS will suddenly fail, since the counter will overflow and start from zero again.

Though the native OpenVMS time format can range far into the future, applications based on the C runtime library will likely encounter timekeeping problems beyond January 19, 2038 due to the Year 2038 problem. Many components and applications may also encounter field-length-related date problems at year 10000.

Detailed information on time and timekeeping, and on daylight saving time and timezone differential factor operations, is contained in the OpenVMS FAQ.

Programming

The common language programming environment is described in the OpenVMS Calling Standard and the OpenVMS Programming Concepts manuals. This provides mixed-language calls, and a set of language-specific, run-time library (RTL), and system service routines. The language calls and the RTLs are implemented in user-mode shareable images, while the system services calls are generally part of the operating system, or part of privileged-mode code. This distinction between languages and RTLs and system services was once fairly clean and clear, but the implementations and specifics have become rather more murky over the years.

Various utilities and tools are integrated, as are various add-on languages and tools.

Debugging

The VMS Debugger supports all DEC compilers and many third party languages. It allows breakpoints, watchpoints and interactive runtime program debugging either using a command line or graphical user interface. OpenVMS Debugger Manual

Common Language Environment

Among OpenVMS's notable features is the Common Language Environment, a strictly defined standard that specifies calling convention for functions and routines, including use of stacks, registers, etc., independently of programming language. Because of this, it is possible to call a routine written in one language (e.g. Fortran) from another (e.g. COBOL), without needing to know the implementation details of the target language. OpenVMS itself is implemented in a variety of different languages (primarily BLISS, VAX Macro and C) (per comp.os.vms newsgroup postings from members of HP OpenVMS Engineering), and the common language environment and calling standard supports freely mixing these languages, and Ada, PL/I, Fortran, BASIC, and others. This is in contrast to a system such as Unix, which is implemented nearly entirely in the C language.

Macro32 (an assembler on OpenVMS VAX, and a compiler on OpenVMS Alpha and on OpenVMS I64) is available within and integrated into OpenVMS. BLISS compilers are available for download from the OpenVMS Freeware, as are various ports of Perl, PHP, Ruby and other languages. Java is available from the HP Java website. C, Fortran and other languages are commercial products, and are available for purchase.

Run-time Libraries

OpenVMS contains a very rich set of Run-time Libraries (RTLs). These cover a wide range of functions, including String manipulation (STR$ routines), Mathematical operations (MTH$ routines), the Run-time Library (LIB$) routines, Screen Management operations (SMG$ routines) and a number of other categories grouped together as General Purpose functions (OTS$ routines). These functions, combined with the low-level System Services, make it easy to write complex programs.

Before writing a simple program in a high-level programming language, however, users should consider if the needed operation can be performed using DCL's functions from a command file. Start with the OpenVMS User's Guide.

Security

OpenVMS provides various security features and mechanisms, including security identifiers, resource identifiers, subsystem identifiers, ACLs, and detailed security auditing and alarms. Specific versions evaluated at DoD NCSC Class C2 and, with the SEVMS security enhanced services support, at NCSC Class B1, per the NCSC Rainbow Series. OpenVMS also holds an ITSEC E3 rating.

Cross Platform Applications

OpenVMS supports the following 'industry standard' tools and applications (as well as many more).

- SAMBA (CIFS)
- Apache Web Server
- TomCat
- Diskeeper

- Zip/Unzip (Info-Zip)

Documentation

The OpenVMS operating documentation for various recent releases and for various core OpenVMS layered products is available online**.**

SPDs are introductory and legal descriptions of various products, listing the various supported capabilities and product features.

The OpenVMS Frequently Asked Questions (FAQ) contains information and pointers associated with OpenVMS, and is available in various formats**.**

Releases, software support status

The current OpenVMS release are OpenVMS V8.4 for Alpha and Integrity servers, and OpenVMS V7.3 for VAX servers.

HP provides Current Version Support (CVS) and Prior Version Support (PVS) for various OpenVMS releases. The OpenVMS Roadmap guarantees PVS status for specific releases (V5.5-2, V5.5-2H4, V6.2, V6.2-1H3, V7.3-2) until 2012, and only then ending with 24 month's prior notice. CVS is provided for the current release and for the immediately prior release.

Applicable industry standards

The following are some of the industry standards claimed in the OpenVMS Software Product Description (SPD) document:

- ANSI X3.4-1986: ASCII
- ANSI X3.22-1973/FIPS 3-1: Magtape, 800 BPI NRZI
- ANSI X3.27-1987/FIPS 79: Magtape, Labels and Volume Structures
- ANSI X3.39-1986/FIPS 25: Magtape, 1600 BPI PE
- ANSI X3.40-1983: Magtape, unrecorded
- ANSI X3.41-1974: ASCII 7-bit control sequences
- ANSI X3.42-1975: Numeric values in character strings
- ANSI X3.54-1986/FIPS 50: Magtape, 6250 BPI GCR
- ANSI X3.131-1986/ISO 9316(1989): SCSI-1
- ANSI X3.131-1994/ISO 10288(1994): SCSI-2
- ANSI/IEEE 802.2-1985: logical link control
- ANSI/IEEE 802.3-1985: Ethernet CSMA/CD
- FIPS 1-2: Code for Information Interchange; includes ANSI X3.4-1977(86)/FIPS 15; ANSI X3.32-1973/FIPS 36; ANSI X3.41-1974/FIPS 35; FIPS 7
- FIPS 16-1/ANSI X3.15-1976: Serial Comms Bit Sequencing; FED STD 1010
- FIPS 22-1/ANSI X3.1-1976: Synch signaling for DTE/DCE comms; FED STD 1013

- FIPS 37/ANSI X3.36-1975: Synch High-Speed signaling for DTE/DCE comms; GIPS 1001
- FIPS 86/ANSI X3.64-1979: Additional Controls for Use with ASCII
- ISO 646: ISO 7-bit Coded Character Set for Information Exchange
- ISO 1001: Magtape, Labels and Volume Structures
- ISO 1863: Magtape, 800 BPI NRZI
- ISO 1864: Magtape, unrecorded / NRZI and PE
- ISO 2022: Code extensions for ISO 646
- ISO 3307: Time and Date Representations
- ISO 3788: Magtape, 1600 BPI PE
- ISO 4873: 8-Bit Character Codes
- ISO 5652: Magtape, 6250 BPI GCR
- ISO 6429: Control Sequences
- ISO 9660: CD-ROM volume and file structures

OpenVMS Hobbyist Program

Despite being a proprietary commercial operating system, in 1997 OpenVMS and a number of layered products were made available free of charge for hobbyist, non-commercial use as part of the OpenVMS Hobbyist Program. Since then, several companies producing OpenVMS software have made their products available under the same terms, such as Process Software and MVP Systems.

As of 2006, the time required to obtain a hobbyist license was approximately one week from start to finish; from registration with a user group through acquisition of licenses and media. Hobbyist CD media is available for US$30, including international shipping. No anonymous FTP software downloads are available to hobbyists.

A number of hobbyist systems are open to the public, including the Deathrow Cluster.

Poetry Hacklab provides telnet and ssh access (username and password is luther) to two VAX/VMS machines located at the Freaknet Computer Museum.

Central OpenVMS-related topics

OpenVMS-related terms and acronyms include:

- ACMS - Digital's transaction processing (TP) system, often used with the DECdtm distributed transaction manager system service components of OpenVMS, and with the DECforms and Rdb products in applications with transactional requirements
- Asynchronous system trap (AST)
- DECforms - Digital's successor to the Forms Management System
- DECnet - Digital's proprietary networking architecture which also includes MOP.
- DELTA and XDELTA - OpenVMS debuggers

- DIGITAL Command Language (DCL) - Digital Command Language - command line interpreter.
- DECwindows - Digital's implementation of the X Window System.
- Event flag - a simple synchronization mechanism
- Files-11 - low level filesystem
- File Description Language (FDL) - defines file record/field structure
- Forms Management System (FMS) - Digital's first generation language-independent Form driver
- Local Area Transport (LAT) - is a LAN-based non-routable communications protocol to support DEC and other Terminal Servers
- QIO Queued Input Output; the low-level I/O interface
- Oracle Rdb - An SQL compliant relational database created by DEC but now owned by Oracle
- Record Management Services (RMS) - high-level, language/device-independent Input/output
- Runtime libraries (RTL) - shared routines and functions, callable from any language
- OpenVMS Galaxy - co-habitating OpenVMS installations; a form of system partitioning
- OpenVMS Clusters - for redundancy, incremental hardware upgrades, or disaster tolerance
- System 1032 (S1032) - A high-performance database management system and application development environment designed to support the OpenVMS user community. Used at some companies in the 1980s and 90s, but, in little use today.
- XQP - the eXtended QIO Processor (XQP), which implements the Files-11 filesystem.

Chapter-6

Oracle Grid Engine and OpenMosix

Oracle Grid Engine

Oracle Grid Engine

Developer(s)	Oracle (formerly Sun Microsystems) in association with the community
Stable release	6.2u7 / December 24, 2010; 3 months ago
Operating system	Cross-platform
Type	Grid computing
License	SISSL

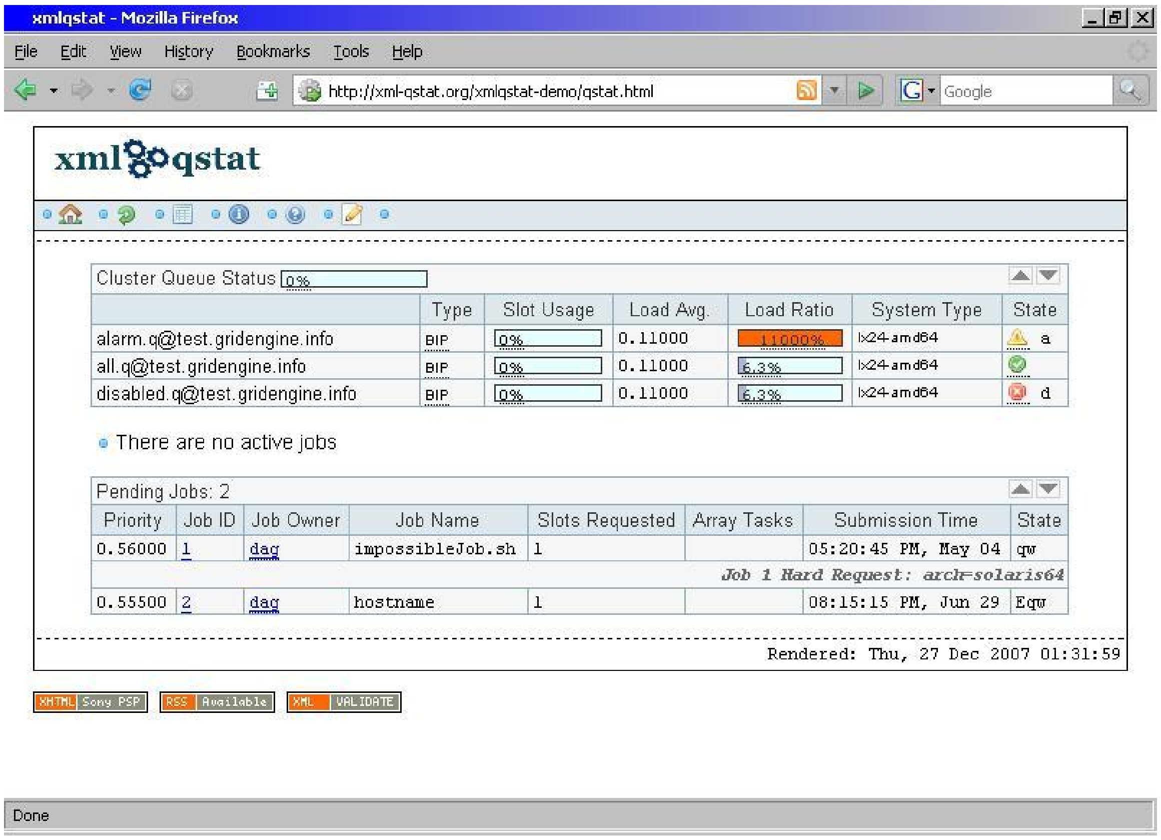
Oracle Grid Engine, previously known as *Sun Grid Engine* (**SGE**), previously known as **CODINE** (COmputing in DIstributed Networked Environments) or **GRD** (Global Resource Director), is an open source batch-queuing system, developed and supported by Sun Microsystems. Sun once also sold a commercial product based on SGE, known as **N1 Grid Engine** (**N1GE**).

Grid Engine is open source and free to use from the project website under the Sun Industry Standards Source License. There is a commercial version available from the Oracle site. It appears that all further versions, starting from 6.2u6, will be commercial (with a 90-day free trial). Licenses cost 500 USD per processor).

SGE is typically used on a computer farm or high-performance computing (HPC) cluster and is responsible for accepting, scheduling, dispatching, and managing the remote and distributed execution of large numbers of standalone, parallel or interactive user jobs. It also manages and schedules the allocation of distributed resources such as processors, memory, disk space, and software licenses.

SGE is the foundation of the Sun Grid utility computing system, made available over the Internet in the United States in 2006, later becoming available in many other countries.

Features


xmlqstat - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://xml-qstat.org/xmlqstat-demo/qstat.html
Google
xml qstat
Done

Cluster Queue Status 0%

	Type	Slot Usage	Load Avg.	Load Ratio	System Type	State
alarm.q@test.gridengine.info	BIP	0%	0.11000	11000%	lx24-amd64	a
all.q@test.gridengine.info	BIP	0%	0.11000	6.3%	lx24-amd64	
disabled.q@test.gridengine.info	BIP	0%	0.11000	6.3%	lx24-amd64	d

There are no active jobs

Pending Jobs: 2

Priority	Job ID	Job Owner	Job Name	Slots Requested	Array Tasks	Submission Time	State
0.56000	1	dag	impossibleJob.sh	1		05:20:45 PM, May 04	qw
						Job 1 Hard Request: arch=solaris64	
0.55500	2	dag	hostname	1		08:15:15 PM, Jun 29	Eqw

Rendered: Thu, 27 Dec 2007 01:31:59

XHTML Sony PSP RSS Available XML VALIDATE

A screenshot of the xml-qstat web interface.

Features new in version 6.2

- Advance reservation
- Array job interdependencies
- Rule-based *Resource Quota* control
- Enhanced remote execution (without using external rshd/rlogind/sshd processes)
- Multi-clustering
- Daemons managed by the Service Management Facility on Solaris
- Pseudo TTY (pty) support for interactive jobs
- Job Submission Verifier (client-side and server-side job verification)
- GUI Installer and SGE Inspect
- Topology-aware scheduling and thread binding
- Hadoop integration, Amazon EC2 integration for cloud computing

Other features of SGE include:

- Multiple advanced scheduling algorithms allow powerful policy-based resource allocation
- Cluster queues
- Job and scheduler fault tolerance - Grid Engine continues to operate as long as there is one or more hosts available
- Job checkpointing
- Job arrays and job tasks
- DRMAA (Job API)
- Resource reservation
- XML status reporting (*qstat* and *qhost*), and the *xml-qstat* web interface
- Parallel jobs (MPI, PVM, OpenMP), and scalable parallel job startup with qrsh
- Usage accounting
- Accounting and Reporting COnsole (ARCO)
- parallel make: distmake, dmake (Sun Studio), and SGE's own qmake
- FLEXlm integration and multi-cluster software license management with *LicenseJuggler*

Platforms

SGE runs on multiple platforms, including:

- AIX
- BSD - FreeBSD, NetBSD, OpenBSD
- HP-UX
- IRIX
- Linux
- Mac OS X
- Solaris
- SUPER-UX
- Tru64
- Windows via SFU (Interix) or SUA (Microsoft Windows Services for UNIX) (as execution hosts only)
- Z/OS (in progress)

Cluster architecture

A typical Grid Engine cluster consists of a master host, and one or more execution hosts. Moreover, multiple *shadow masters* can be configured as hot spares, which take over the role of the master when the original master host crashes.

Support and training

Sun provides support contracts for the commercial version of Grid Engine on most UNIX platforms and Windows. Professional services, consulting, training, and support are also provided by Sun Partners. Sun partners with Georgetown University to deliver Grid Engine administration classes. *The Bioteam* runs short SGE training workshops that are 1 or 2 days long.

Users can obtain community support on the Grid Engine mailing lists.

Grid Engine Workshops were held in 2002, 2003, 2007, and 2009 in Regensburg, Germany.

Prominent users

Notable deployments of SGE include:

- Sun Grid
- the TSUBAME supercomputer at the Tokyo Institute of Technology, which was number 7 on June 2006 TOP500 list.
- Ranger at the Texas Advanced Computing Center (TACC). Ranger has 62,976 processor cores in 3,936 nodes and a peak performance of 504TFlops. Ranger was the 4th most powerful TOP500 supercomputer in 2008.
- San Diego Supercomputer Center (SDSC)
- Geophysical Fluid Dynamics Laboratory (NOAA GFDL)

History

In 2000, Sun acquired Gridware, Inc. a privately owned commercial vendor of advanced computing resource management software with offices in San Jose, Calif., and Regensburg, Germany. Later that year, Sun offered a free version of Gridware for Solaris and Linux, and renamed the product Sun Grid Engine.

In 2001, Sun made the source code available, and adopted the open source development model. Ports for Mac OS X and *BSD were contributed by the non-Sun open source developers.

In 2010, after the purchase of Sun by Oracle, the Grid Engine 6.2 update 6 source code was not included with the binaries, and changes were not put back to the project's source repository. In response to this, the Grid Engine community started the *Open Grid Scheduler* project to continue to develop and maintain a free implementation of Grid Engine.

On January 18, 2011, it was announced that Univa had recruited several principal engineers from the former Sun Grid Engine team and that Univa would be developing their own forked version of Grid Engine. The newly announced Univa Grid Engine will include commercial support and would compete with the official version of Oracle Grid Engine.

Other Grid Engine based products

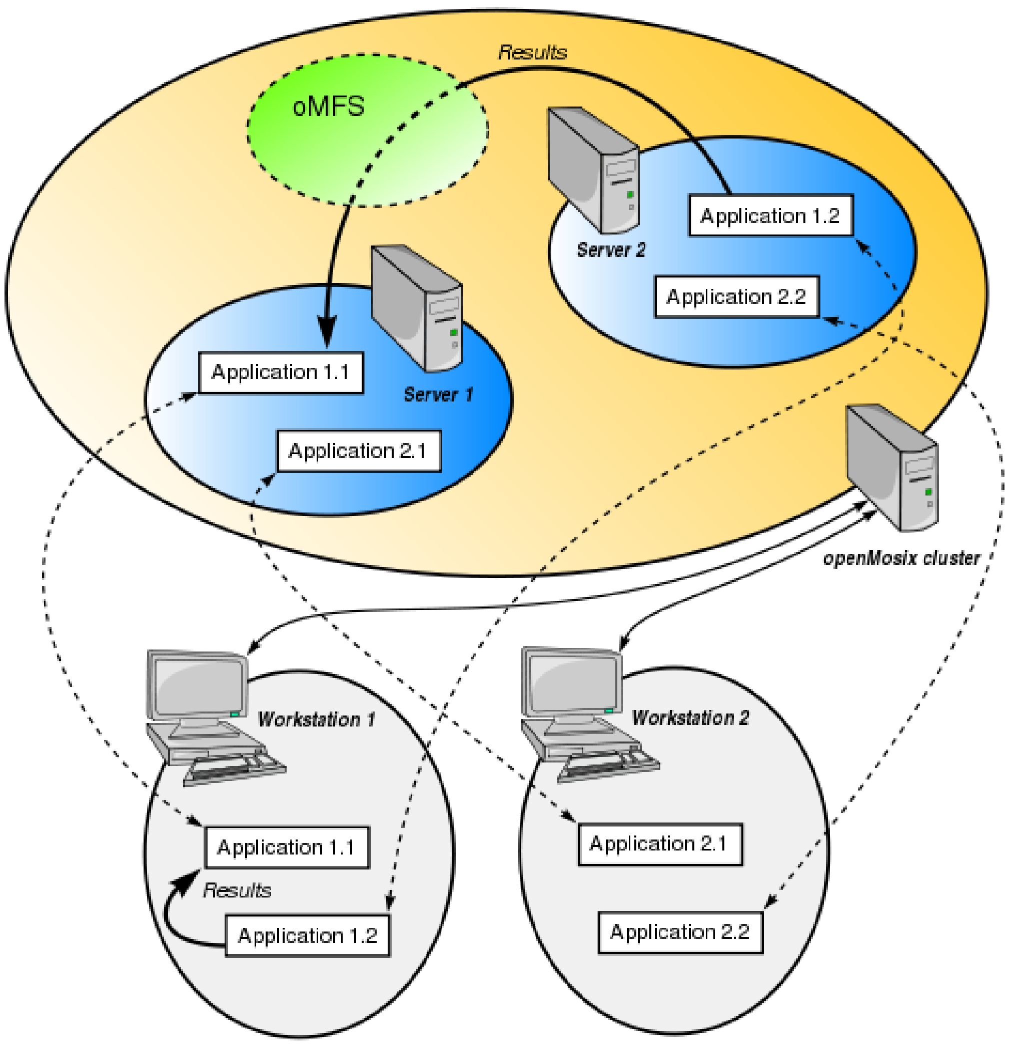
- Sun Constellation System
- Sun Visualization System
- Sun Compute Cluster
- ClusterVisionOS Distribution
- Rocks Cluster Distribution
- EGEE
- Univa's UniCluster Express
- BioTeam's iNquiry
- Nimbus - uses Grid Engine as a virtual machine scheduler in a cloud computing environment

Add-on software

A number of SGE add-ons are available:

- Solaris Cluster integration
- *Service Domain Management* module in order to meet service level objectives
- Transfer-queue Over Globus (TOG). Globus added support for Grid Engine in Globus Toolkit 5.0.0
- JOb Scheduling Hierarchically (JOSH)

openMosix



Transfers in an openMosix cluster.

openMosix was a free cluster management system that provided single-system image (SSI) capabilities, e.g. automatic work distribution among nodes. It allowed program processes (not threads) to migrate to machines in the node's network that would be able to run that process faster (process migration). It was particularly useful for running parallel and intensive input/output (I/O) applications. It was released as a Linux kernel patch, but was also available on specialized Live CDs. openMosix development has been halted by its developers, but the LinuxPMI project is continuing development of the former openMosix code.

History

openMosix was originally forked from MOSIX by Moshe Bar on February 10, 2002 when MOSIX became proprietary software.

openMosix was considered stable on Linux kernel 2.4.x for the x86 architecture, but porting to Linux 2.6 kernel remained in the alpha stage. Support for the 64-bit AMD64 architecture only started with the 2.6 version.

On July 15, 2007, Bar announced that the openMOSIX project would reach its end of life on March 1, 2008, due to the decreasing need for SSI clustering as low-cost multi-core processors increase in availability.

OpenMosix used to be distributed as a Gentoo Linux kernel choice, but it was removed from Gentoo Linux's *Portage tree* in February 2007.

As of March 1, 2008, openMosix read-only source code is still hosted at SourceForge. The LinuxPMI project is continuing development of the former openMosix code.

Chapter-7

Oracle RAC and IBM Parallel Sysplex

Oracle RAC

In database computing, **Oracle Real Application Clusters (RAC)** — an option for the Oracle Database software produced by Oracle Corporation and introduced in 2001 with Oracle9i — provides software for clustering and high availability in Oracle database environments. Oracle Corporation includes RAC with the Standard Edition of Oracle Database (aka Baby RAC), but makes it an extra-charge option for the Enterprise Edition.

Functionality

Oracle RAC allows multiple computers to run Oracle RDBMS software simultaneously while accessing a single database, thus providing a clustered database.

In a non-RAC Oracle database, a single instance accesses a single database. The *database* consists of a collection of data files, control files, and redo logs located on disk. The *instance* comprises the collection of Oracle-related memory and operating system processes that run on a computer system.

In an Oracle RAC environment, two or more computers (each with an instance) concurrently access a single database. This allows an application or user to connect to either computer and have access to a single coordinated set of data.

Aims

Implementation

Oracle RAC depends on the infrastructure component **Oracle Clusterware** to coordinate multiple servers and their sharing of data storage.

Cache Fusion

Prior to Oracle 9, network-clustered Oracle databases used a storage device as the data-transfer medium (meaning that one node would write a data block to disk and another node would read that data from the same disk), which had the inherent disadvantage of lackluster performance. Oracle 9i addressed this issue: RAC uses a dedicated network-connection for communications internal to the cluster.

Since all computers/instances in an RAC access the same database, the overall system must guarantee the coordination of data changes on different computers such that whenever a computer queries data it receives the current version — even if another computer recently modified that data. Oracle RAC refers to this functionality as *Cache Fusion*. Cache Fusion involves the ability of Oracle RAC to "fuse" the in-memory data cached physically separately on each computer into a single, global cache.

Versions

- Oracle Real Application Clusters One Node (RAC One Node) applies RAC to single-node installations running Oracle Database 11g Release 2 Enterprise Edition.

Evolution

Relative to the single-instance Oracle database, Oracle RAC adds additional complexity. While database automation makes sense for single-instance databases, it becomes even more necessary for clustered databases because of their increased complexity.

Oracle Real Application Clusters (RAC), introduced with Oracle9i in 2001, supersedes the Oracle Parallel Server (OPS) database option. Whereas Oracle9i required an external clusterware (known as vendor clusterware like Veritas or Sun Cluster) for most of the Unix flavors (except for Linux and Windows where Oracle provided free clusterware called Cluster Ready Services or CRS), as of Oracle 10g, Oracle's clusterware product was available for all operating systems. With the release of Oracle Database 10g Release 2 (10.2), Cluster Ready Services was renamed to Oracle Clusterware. When using Oracle 10g or higher, Oracle Clusterware is the only clusterware that you need for most platforms on which Oracle RAC operates (except for Tru cluster, in which case you need vendor clusterware). You can still use clusterware from other vendors if the clusterware is certified for Oracle RAC.

In RAC the node performing the write-transaction must take ownership of the relevant area of the database: typically this involves a request across the cluster interconnection (local IP network) to transfer the data-block ownership from another node to the one wishing to do the write. This takes a relatively long time (from a few milliseconds to tens of milliseconds) compared to single database-node using in-memory operations. For many types of applications, the time spent coordinating block access across systems is low relative to the many operations on the system, and RAC will scale comparably to a single system. Moreover, high read-transactional databases (such as data-warehousing applications) work very well under RAC, as no need for ownership-transfer exists. (Oracle 11g has made many enhancements in this area and performs a lot better than earlier versions for read-only workloads.)

The overhead on the resource mastering (or ownership-transfer) is very minimal for fewer than three nodes, as the request for any resource in the cluster can be obtained in a maximum of three hops (owner-master-requestor). This makes Oracle RAC horizontally scalable with many nodes. Application vendors (such as SAP) use Oracle RAC to demonstrate the scalability of their application. Most of the biggest OLTP benchmarks are on Oracle RAC. Oracle RAC 11g supports up to 100 nodes.

For some applications, RAC may require careful application-partitioning to enhance performance. An application which scales linearly on an SMP machine may scale linearly under RAC. However, if the application cannot scale linearly on SMP, it will not scale when ported to RAC. In short the application scalability is based on how well the application scales in a single instance.

Competitive context

Shared-nothing and shared-everything architectures each have advantages over the other. DBMS vendors and industry analysts regularly debate the matter; for example, Microsoft touts a comparison of its SQL Server 2005 with Oracle 10g RAC by Performance Tuning Corporation.

Oracle Corporation offered a Shared Nothing architecture RDBMS with the advent of the IBM SP and SP2 with the release of 7.x MPP editions, in which virtual shared drives (VSD) were used to create a Shared Everything implementation on a Shared Nothing architecture.

Shared-Everything

Some commercially-available databases offer a "shared-everything" architecture. IBM DB2 for z/OS (the IBM mainframe operating-system) has provided a high-performance data-sharing option since the mid 1990s when IBM released its mainframe hardware and software-clustering infrastructure. In late 2009, IBM announced DB2 pureScale, a shared-disk clustering scheme for DB2 9.8 on AIX that mimics the parallel sysplex implementation behind DB2 data sharing on the mainframe.

In February 2008, Sybase released its Adaptive Server Enterprise, Cluster Edition. It resembles Oracle RAC in its shared-everything design.

Although technically not shared-everything, Sybase also provides a column-based relational database focused on analytic and datawarehouse applications called Sybase IQ which can be configured to run in a shared disk mode.

Shared-Nothing

Competitive products offering shared-nothing architectures include:

- MySQL Cluster (Oracle Corporation has owned MySQL since 2009)
- IBM InfoSphere Warehouse editions that include the Database Partitioning Feature (formerly known as DB2 Extended Enterprise Edition)
- Greenplum
- Paraccel
- Netezza (aka. Netezza Performance Server)
- Teradata
- Lexst Database Cluster

IBM Parallel Sysplex

In computing, a **Parallel Sysplex** is a cluster of IBM mainframes acting together as a single system image with z/OS. Used for disaster recovery, Parallel Sysplex combines data sharing and parallel computing to allow a cluster of up to 32 systems to share a workload for high performance and high availability.

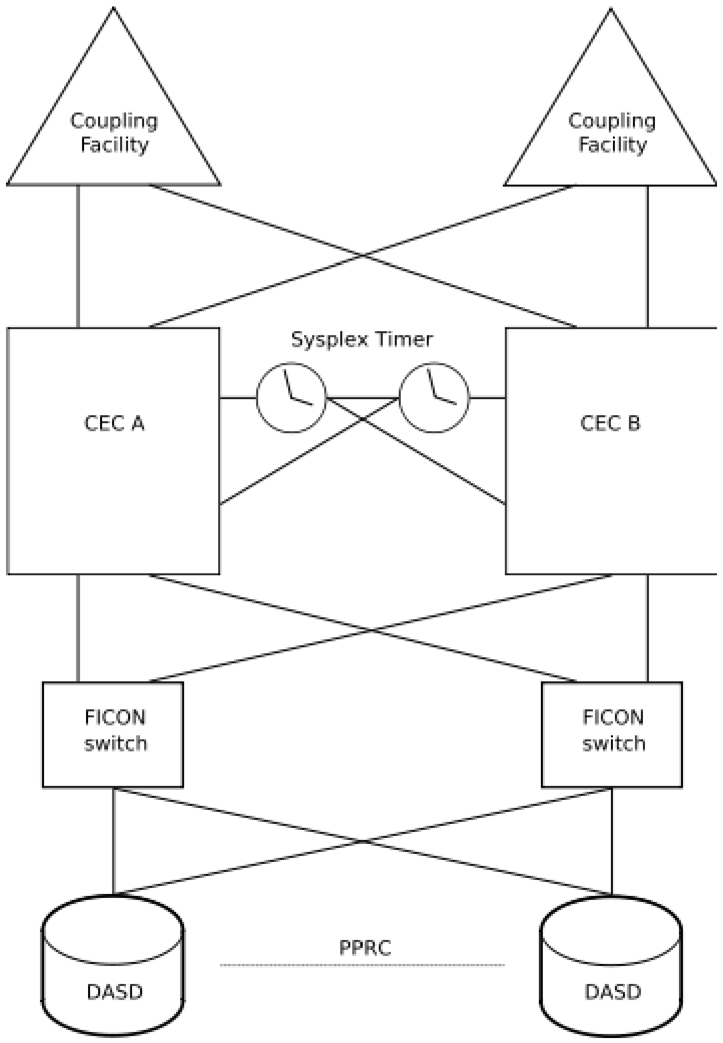
Sysplex

In IBM mainframe computers, a **Systems Complex**, commonly called a **Sysplex**, allows multiple processors to be joined into a single unit, sharing the same *Sysplex name* and Couple Data Sets. Put another way, a Sysplex is a single logical system running on one or more physical systems. Sysplexes are often isolated within a single system, but Parallel Sysplex technology allows multiple mainframes to act as one.

Components of a Sysplex include:

- A Sysplex Timer which synchronizes all member systems' clocks;
- Global Resource Serialization (GRS), which allows multiple systems to access the same resources concurrently, serializing where necessary to ensure exclusive access;
- Cross System Coupling Facility (XCF), which allows systems to communicate peer-to-peer;
- Couple Data Sets (CDS);

Parallel Sysplex



Schematic representation of a Parallel Sysplex

The forerunner to Parallel Sysplex was **Virtual Coupling**, a technique which allowed up to 12 IBM ESA/390 systems to execute jobs in parallel. The true Parallel Sysplex was introduced with then-new mainframe models in April 1994. Major components of a Parallel Sysplex include:

- Coupling Facility (CF or ICF) hardware, allowing multiple processors to share, cache, update, and balance data access;
- Sysplex Timers or Server Time Protocol to synchronize the clocks of all member systems;
- High speed, high quality, redundant cabling;
- Software (operating system services and, usually, middleware such as DB2).

The Coupling Facility may be either a dedicated external system (a small mainframe, such as a System z9 BC, specially configured with only coupling facility processors) or integral processors on the mainframes themselves configured as ICFs (Internal Coupling Facilities). It is recommended that at least one external CF be used in a parallel sysplex. A Parallel Sysplex has at least two CFs and/or ICFs for redundancy. Every mainframe participating in a Parallel Sysplex does not need an ICF or its own external CF — mainframes merely attach, via cables, to the external CFs or ICFs. Server Time Protocol (STP) replaced the Sysplex Timers beginning in 2005 for System z mainframe models z990 and newer. A Sysplex Timer is a physically separate piece of hardware from the mainframe, whereas STP is an integral facility within the mainframe's microcode. With STP and ICFs it is possible to construct a complete Parallel Sysplex installation with two connected mainframes. Moreover, a single mainframe can contain the internal equivalent of a complete physical Parallel Sysplex, useful for application testing and development purposes.

The IBM Systems Journal dedicated a full issue to all the technology components.

Geographically Dispersed Parallel Sysplex

Geographically Dispersed Parallel Sysplex (**GDPS**) is an extension of Parallel Sysplex of mainframes located, potentially, in different cities. GDPS includes configurations for single site or multiple site configurations:

- GDPS/HyperSwap Manager: It is a synchronous Peer to Peer Remote Copy (PPRC) technology for use within a single data center. Data is copied from the primary storage device to a secondary storage device. In the event of a failure on the primary storage device, the system automatically makes the secondary storage device the primary, usually without disrupting running applications.
- GDPS/PPRC: It is a synchronous data mirroring technology (PPRC) that can be used on mainframes 200 kilometres (120 mi) apart. In a two-system model, both sites can be administered as if they were one system. In the event of a failure of a system or storage device, recovery can occur with limited or no data loss automatically.
- GDPS/XRC: It is an asynchronous Extended Remote Copy (XRC) technology with no restrictions on distance. XRC copies data on storage devices between two sites such that only a few seconds of data may be lost in the event of a failure. If a failure does occur, a user must initiate the recovery process. Once initiated, the process is automatic in recovering from secondary storage devices and reconfiguring systems.
- GDPS/GM: It is an asynchronous IBM Global Mirror technology with no restrictions on distance. It is designed to recovery from a total failure at one site. It will activate secondary storage devices and backup systems.
- GDPS/MGM & GDPS/MzGM: These are configurations for systems with more than two systems/sites for purposes of disaster recovery. GDPS/MGM and GDPS/MzGM are based on GDPS/PPRC and GDPS/XRC, respectively.

Chapter-8

Xgrid

Xgrid

Developer(s)	Apple Inc.
Initial release	January 6, 2004
Development status	Active
Operating system	Mac OS X
Platform	Independent
Type	Distributed computing
License	Proprietary EULA

Xgrid is a proprietary software program and distributed computing protocol developed by the Advanced Computation Group subdivision of Apple Inc that allows networked computers to contribute to a single task.

It provides network administrators a method of creating a computing cluster, which allows them to exploit previously unused computational power for calculations that can be divided easily into smaller operations, such as Mandelbrot maps. The setup of an Xgrid cluster can be achieved at next to no cost, as Xgrid client is pre-installed on all computers running Mac OS X 10.4 or later. The Xgrid controller, the job scheduler of the Xgrid operation, is also included within Mac OS X Server and as a free download from Apple. Apple has kept the command-line job control mechanism minimalist while providing an API to develop more sophisticated tools built around it.

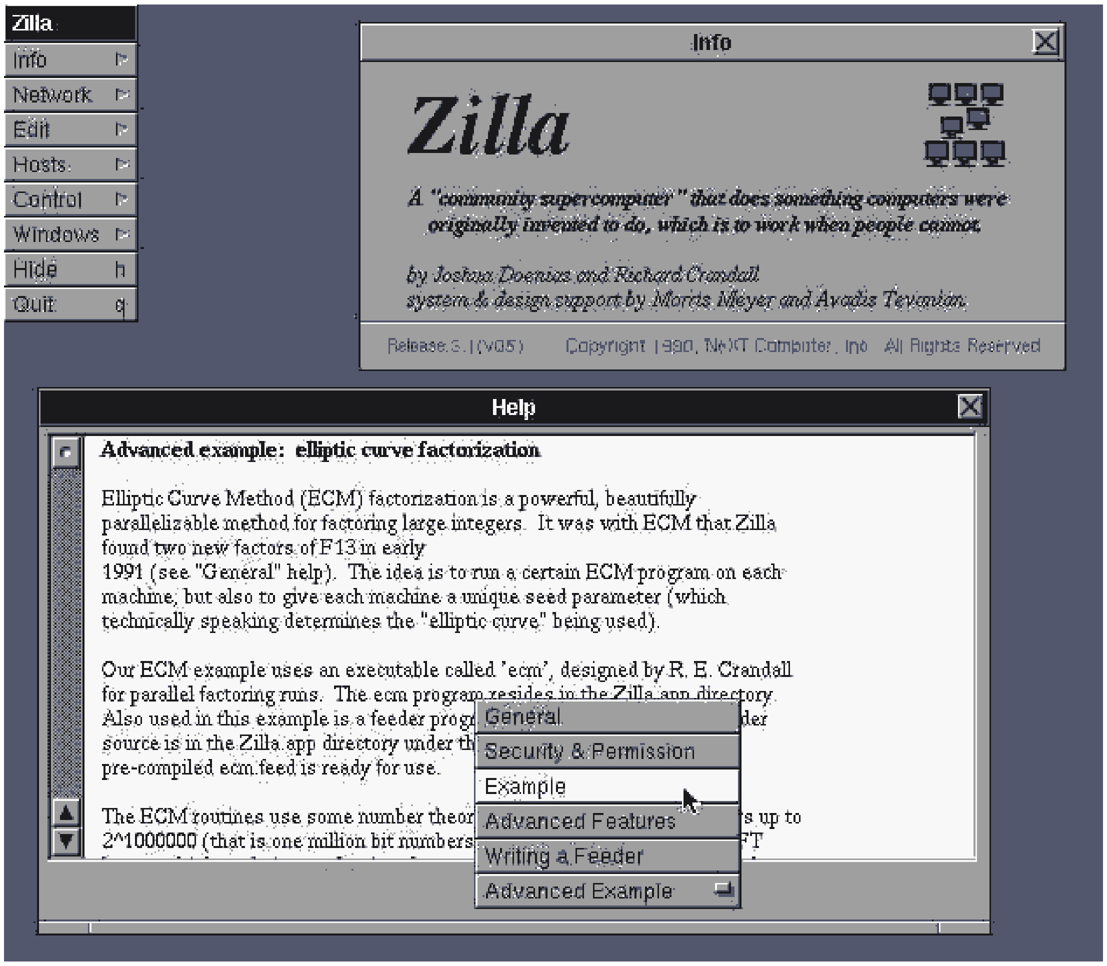
The program employs its own communication protocol layered on top of a schema to communicate to other nodes. This communication protocol interfaces with the BEEP infrastructure, a network application protocol framework. Computers discovered by the

Xgrid system, that is computers with Mac OS X's Xgrid service enabled, are automatically added to the list of available computers to use for processing tasks.

When the initiating computer sends the complete instructions, or job, for processing to the controller, the controller splits the task up into these small instruction packets, known as tasks. The design of the Xgrid system consists of these small packets being transferred to all the Xgrid-enabled computers on the network. These computers, or nodes, execute the instructions provided by the controller and then return the results. The controller assembles the individual task results into the whole job results and returns them to the initiating computer.

Apple modeled the design of Xgrid on the Zilla program, distributed with NeXT's OPENSTEP operating system application programming interface (API), which Apple owned the rights to. The company also opted to provide the client version of Mac OS X with only command-line functions and little flexibility, while giving the Mac OS X Server version of Xgrid a GUI control panel and a full set of features.

History



Zilla

A "community supercomputer" that does something computers were originally invented to do, which is to work when people cannot.

by Joshua Doenias and Richard Crandall
system & design support by Morris Meyer and Avadis Tevanian

Release 3.1 (v05) Copyright 1990, NeXT Computer, Inc. All Rights Reserved

Advanced example: elliptic curve factorization

Elliptic Curve Method (ECM) factorization is a powerful, beautifully parallelizable method for factoring large integers. It was with ECM that Zilla found two new factors of F13 in early
1991 (see "General" help). The idea is to run a certain ECM program on each machine, but also to give each machine a unique seed parameter (which technically speaking determines the "elliptic curve" being used).

Our ECM example uses an executable called 'ecm', designed by R. E. Crandall for parallel factoring runs. The ecm program resides in the Zilla.app directory. Also used in this example is a feeder prog der source is in the Zilla.app directory under th pre-compiled ecm.feed is ready for use.

The ECM routines use some number theor s up to 2^1000000 (that is one million bit numbers FT

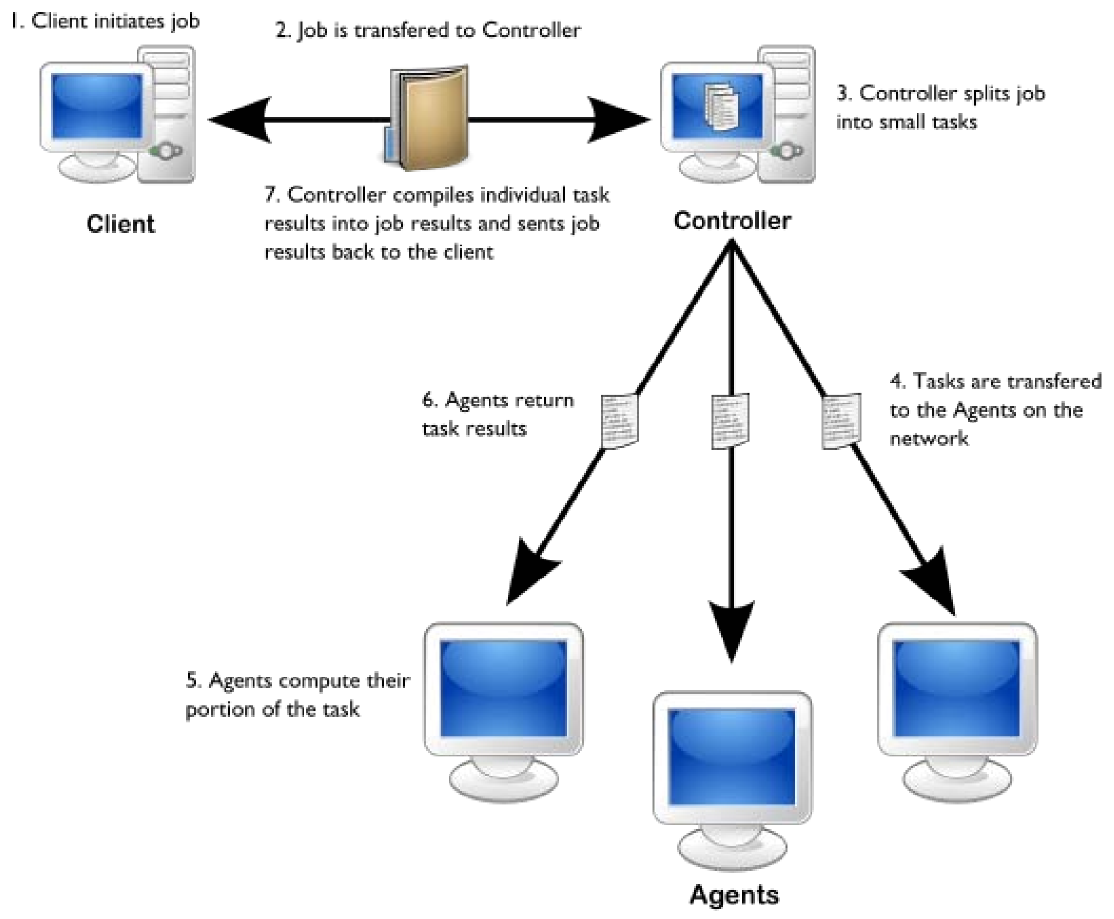
Zilla

Xgrid's original concept can be traced back to Zilla.app, found in the OPENSTEP operating system API, created by NeXT in the late 1980s. Zilla was the first distributed computing program released on an end-user operating system and which used the idle screen-saver motif, a design feature found in widely used projects such as Seti@Home and Distributed.net. Zilla won the national ComputerWorld Smithsonian Award (Science Category) in 1991 for ease of use and good design. Apple acquired Zilla, along with the rest of NeXT, in 1997 and later used Zilla as inspiration for Xgrid. The first beta version of Xgrid was released in January 2004.

Several organizations have adopted Xgrid in large international computing networks. One example of an Xgrid cluster is MacResearch's OpenMacGrid, where scientists can request access to large amounts of processing power to run tasks related to their research. Another was the now defunct Xgrid@Stanford project, which used a range of computers on the Stanford University campus and around the world to perform biochemical research.

In a pre-release promotional piece, *MacWorld* cited Xgrid among the Unix features in "10 Things to Know about TIGER", calling it "handy if you work with huge amounts of experimental data or render complex animations". After Xgrid's introduction in 2004, *InfoWorld* noted that it was a "'preview' grade technology" which would directly benefit from the Xserve G5's launch later that year. *InfoWorld* commentator Ephraim Schwartz also predicted that Xgrid was an opening move in Apple's entry into the enterprise computing market.

Protocol


1. Client initiates job
2. Job is transfered to Controller
3. Controller splits job into small tasks
Client
7. Controller compiles individual task results into job results and sents job results back to the client
Controller
6. Agents return task results
4. Tasks are transfered to the Agents on the network
5. Agents compute their portion of the task
Agents

Xgrid Protocol

The Xgrid protocol uses the BEEP network framework to communicate with nodes on the network. The system's infrastructure includes three types of computers which communicate over the protocol. One is the client, which communicates the calculation. Next is the controller, which starts and segregates the calculation. Finally, the agents process their own allocated part of the calculation.

A computer can act as one or all three of these components at the same time. The Xgrid protocol provides the basic infrastructure for computers to communicate, but is not involved in the processing of the specified calculation. Xgrid is targeted towards time consuming computations that can be easily segregated into smaller tasks, sometimes called *embarrassingly parallel* tasks. This includes Monte Carlo calculations, 3D rendering and Mandelbrot maps.

Within the Xgrid protocol, three types of messages can be passed to other computers on the same cluster: requests, notifications and replies. Requests must be responded to by the recipient with a reply, notifications do not require a reply, and replies are responses to sent messages. They are identified by their name, type (request/notification/reply) and contents. Each message is encapsulated in a BEEP message (BEEP MSG) and is acknowledged on receipt by an empty reply (RPY). Xgrid does not leverage BEEPs message/reply infrastructure. Any received message which requires a response merely generates an independent BEEP message containing the reply. The Xgrid messages are encoded as dictionaries of key/value pairs which are converted to XML before being sent across the BEEP network.

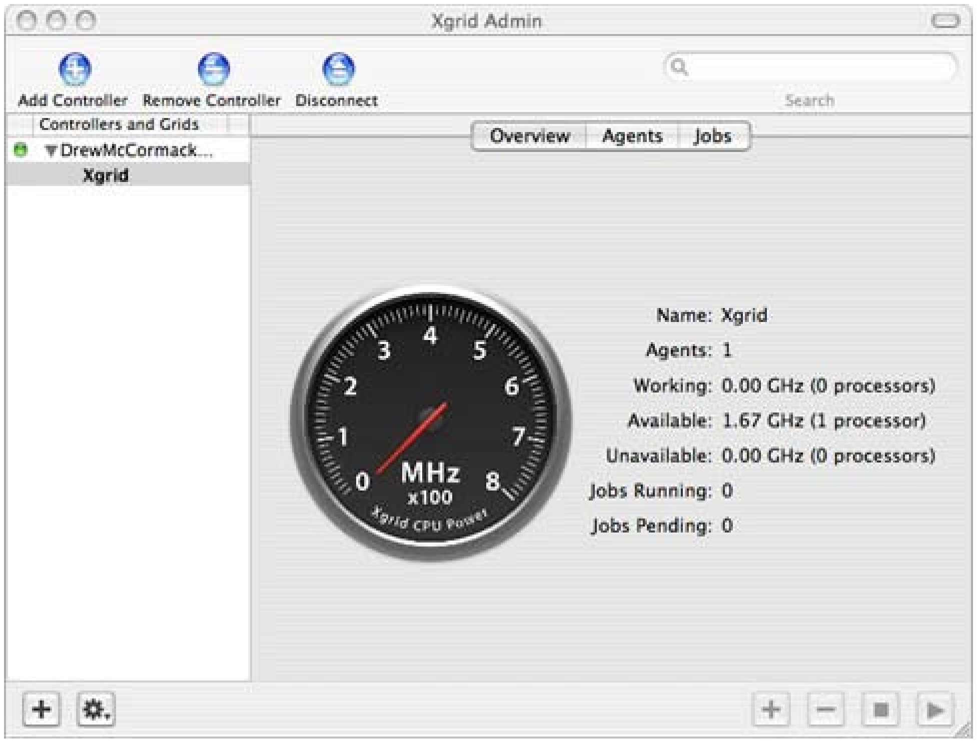
Architecture

The architecture of the Xgrid system is designed around a job based system; the controller sends agents jobs, and the agents return the responses. The actual computation that the controller executes in an Xgrid system is known as a job. The job contains all the files required to complete the task successfully, such as the input parameters, data files, directories, executables and/or shell scripts, the files included in an Xgrid job must be able to be executed either simultaneously or asynchronously, or any benefits of running such a job on an Xgrid is lost. Once the job completes, the controller can be set to notify the client of the task's completion or failure, for example by email. The client can leave the network while the tasks are running. It can also monitor the job status on demand by querying the controller, although it cannot track the ongoing progress of individual tasks.

The controller is central to the correct function of an Xgrid, as this node is responsible for the distribution, supervision and coordination of tasks on agents. The program running on the controller can assign and reassign tasks to handle individual agent failures on demand. The number of tasks assigned to an agent depend on two factors: the number of agents on an Xgrid and the number of processors in each node. The number of agents on an Xgrid determines how the controller will assign tasks. The tasks may be assigned simultaneously for a large number of agents, or queued for a small number of agents. When a node with more than one processor is detected on an Xgrid, the controller may assign one task per processor; this only occurs if the number of agents on the network is lower than the number of tasks the controller has to complete.

Xgrid is layered upon the Blocks Extensible Exchange Protocol (BEEP), an IETF standard comparable to HTTP, but with a focus on two-way multiplexed communication, such as that found in peer-to-peer networks. BEEP, in turn, uses XML to define profiles for communicating between multiple agents over a single network or internet connection.

Interface



Name: Xgrid
Agents: 1
Working: 0.00 GHz (0 processors)
Available: 1.67 GHz (1 processor)
Unavailable: 0.00 GHz (0 processors)
Jobs Running: 0
Jobs Pending: 0

Xgrid administration tool

While it is possible to access Xgrid from the command line, the Xgrid graphical user interface, a program bundled with Mac OS X Server and, as of March 2009, available online, is a much more efficient way of administering an Xgrid system. Originally, the Xgrid agent was included in all Mac OS X version 10.4 installations but the GUI was reserved for users of Mac OS X Server. This decision limited the efforts of the computer community to embrace the platform. Eventually, Apple released the Mac OS X Server Administration Tools to the public, which included the Xgrid administration application bundled with Mac OS X Server.

Despite the lack of a graphical controller interface in the standard (non-server) Mac OS X distribution, it is possible to set up an Xgrid controller via the command line tools `xgridctl` and `xgrid`. Once the Xgrid controller daemon is running, administration of the grid with Apple's Xgrid Admin tool is possible. Some applications, such as VisualHub, provided Xgrid controller capability through their user interfaces.

Chapter-9

Single System Image and Red Hat Cluster Suite

Single system image

In distributed computing, a **single system image** (**SSI**) cluster is a cluster of machines that appears to be one single system. The concept is often considered synonymous with that of a **distributed operating system**, but a single image may be presented for more limited purposes, just job scheduling for instance, which may be achieved by means of an additional layer of software over conventional operating system images running on each node. The interest in SSI clusters is based on the perception that they may be simpler to use and administer than more specialized clusters.

Different SSI systems may provide a more or less complete illusion of a single system.

Features of SSI clustering systems

Different SSI systems may, depending on their intended usage, provide some subset of these features.

Process migration

Many SSI systems provide process migration. Processes may start on one node and be moved to another node, possibly for resource balancing or administrative reasons. As processes are moved from one node to another other associated resources, for example IPC resources may be moved with them.

Process checkpointing

Some SSI systems allow checkpointing of running processes, allowing their current state to be saved and reloaded at a later date. Checkpointing can be seen as related to migration, as migrating a process from one node to another can be implemented by first checkpointing the process, then restarting it on another node. Alternatively checkpointing can be considered as *migration to disk*.

Single process space

Some SSI systems provide the illusion that all processes are running on the same machine - the process management tools (e.g. "ps", "kill" on Unix like systems) operate on all processes in the cluster.

Single root

Most SSI systems provide a single view of the file system. This may be achieved by a simple NFS server, shared disk devices or even file replication.

The advantage of a single root view is that processes may be run on any available node and access needed files with no special precautions. If the cluster implements process migration a single root view enables direct accesses to the files from the node where the process is currently running.

Some SSI systems provide a way of "breaking the illusion", having some node-specific files even in a single root, e.g. HP TruCluster provides a "context dependent symbolic link" (CDSL) which points to different files depending on the node that accesses it. This may be necessary to deal with *heterogeneous* clusters, where not all nodes have the same configuration.

Single I/O space

Some SSI systems allow all nodes access the I/O devices (e.g. tapes, disks, serial lines and so on) of other nodes. There may be some restrictions on the kinds of accesses allowed (For example OpenSSI can't mount disk devices from one node on another node).

Single IPC space

Some SSI systems allow processes on different nodes to communicate using inter-process communications mechanisms as if they were running on the same machine. On some SSI systems this can even include shared memory (can be emulated with Software Distributed shared memory).

In most cases inter-node IPC will be slower than IPC on the same machine, possibly drastically slower for shared memory. Some SSI clusters include special hardware to reduce this slowdown.

Cluster IP address

Some SSI systems provide a "cluster address", a single address visible from outside the cluster that can be used to contact the cluster as if it were one machine. This can be used for load balancing inbound calls to the cluster, directing them to lightly loaded nodes, or for redundancy, moving the cluster address from one machine to another as nodes join or leave the cluster.

Some example SSI clustering systems

SSI Properties of different clustering systems

Name	Process migration	Process checkpoint	Single process space	Single root	Single I/O space	Single IPC space	Cluster IP address[t 1]
Amoeba	Yes	Yes	Yes	Yes	Unknown	Yes	Unknown
AIX TCF	Unknown	Unknown	Unknown	Yes	Unknown	Unknown	Unknown
DragonFly BSD	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown
Genesis							
Inferno							
Kerrighed	Yes	Yes	Yes	Yes	Unknown	Yes	Unknown
LinuxPMI	Yes	Yes	No	Yes	No	No	Unknown
LOCUS	Unknown	Unknown	Yes	Yes	Yes	Yes	Unknown
MOSIX	Yes	Yes	No	Yes	No	No	Unknown
openMosix	Yes	Yes	No	Yes	No	No	Unknown
Open-Sharedroot	No	No	No	Yes	No	No	Yes
OpenSSI	Yes	No	Yes	Yes	Yes	Yes	Yes
VMScluster	No	No	Yes	Yes	Yes	Yes	Yes
Plan 9							
Sprite	Yes	Unknown	No	Yes	Yes	No	Unknown
TruCluster	No	Unknown	No	Yes	No	No	Yes

Red Hat Cluster Suite

The **Red Hat Cluster Suite** includes software to create a high availability and load balancing cluster, it currently does not contain functionality for distributed computing.

Cluster types

The cluster suite itself contains two products. Both can be used on the same system although this use case is unlikely. Both products have been initiated in the community and repackaged by Red Hat. Computational clustering is not part of Cluster Suite, but instead provided by Red Hat MRG.

High Availability Cluster

A **Red Hat cluster suite**, when configured for high availability, attempts to ensure service availability by monitoring other nodes of the cluster. All nodes of the cluster must agree on their configuration and shared services state before the cluster is considered Quorate and services are able to be started.

The primary form of communicating node status is via a network device (commonly Ethernet), although in the case of possible network failure, quorum can be decided through secondary methods such as shared storage or multicast.

Software services, filesystems and network status can be monitored and controlled by the cluster suite, services and resources can be failed over to other network nodes in case of failure.

The Cluster suite forcibly terminates a cluster node's access to services or resources to ensure the node and data is in a known state. The node is terminated by removing power or access to the shared storage.

Service locking and control is guaranteed through fencing and STONITH; more recent versions of Red Hat use a distributed lock manager (DLM), to allow fine grained locking and no single point of failure. Earlier versions of the cluster suite relied on GULM (Grand Unified Lock Manager) which could be clustered, but still presented a point of failure if the nodes acting as GULM servers were to fail. GULM is available, but deprecated, in Red Hat Cluster Suite 5.

Technical Details

- Support for up to 128 nodes (16 on Red Hat Enterprise Linux 3 and 4)
- NFS (Unix) /CIFS (Windows)/GFS (Multiple Operating systems) File system failover support
- Service failover support
- Fully shared storage subsystem
- Comprehensive Data Integrity
- SCSI and Fibre Channel support

Load balancing Cluster

Red Hat adapted the Piranha load balancing software to allow for transparent load balancing and failover between servers. The application being balanced does not require special configuration to be balanced, instead a Red Hat Enterprise Linux server with the load balancer configured, intercepts and routes traffic based on metrics/rules set on the load balancer.

Support and product life cycle

Red Hat Cluster suite support is tied to a matching version of Red Hat Enterprise Linux and follows the same maintenance policy. The product has no activation, time limit or remote kill switch, it will remain working after the support life cycle has ended. It is partially supported running under VMware Virtual Machine .

History

The cluster suite is available in Red Hat Enterprise Linux 2.1, 3, 4 and 5. Supported Global File System as a filesystem in 3 and above. The load balancing software was a fork of the open source Piranha load balancing software.

Chapter-10

CHAOS (Operating System) and FhGFS

CHAOS (operating system)

CHAOS

```
CHAOS     CD version 1.6     http://purehacking.com/chaos/

                         Welcome to CHAOS
                 "The supercomputer for your wallet"

                   Supported ISO boot options are:
                      j1-4, x1-5, l1, l3, s1

                  For boot menu help and information,
                          please press F4.

Pure Hacking Pty Ltd
Internet Security through Penetration Testing Risk Management Services

F2: About     F3: HW Req'd    F4: Boot Opts  F5: IP Opts    F6: Copyrights
boot: _
```

CHAOS-1.6 Boot Welcome Screen

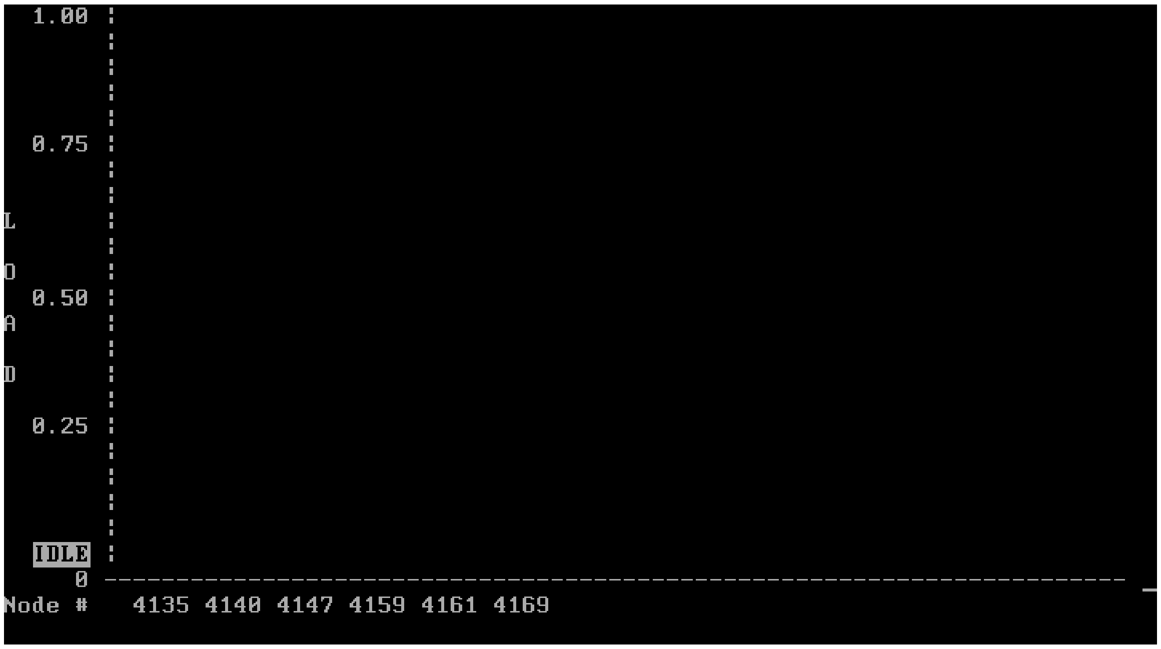
Company / developer	Midnight Code / Ian Latter
OS family	Unix-like
Working state	Current
Source model	Open source
Latest stable release	1.6 / April 2005
Kernel type	Monolithic kernel
Default user interface	text (bash)
License	Various

CHAOS is a small (6Mbyte) Linux distribution designed for creating ad hoc computer clusters. CHAOS is a Live CD which fits on a single business card sized CD-ROM. This tiny disc will boot any i586 class PC (that supports CD booting), into a working openMosix node, without disturbing (or even touching) the contents of any local hard disk.

Designed for large-scale ad hoc clusters, once booted, CHAOS runs from memory allowing the CD to be used on the next node (and allowing for automated rebooting into the host operating system). CHAOS aims to be the most compact, secure and straight-forward openMosix cluster platform available.

About

What it is



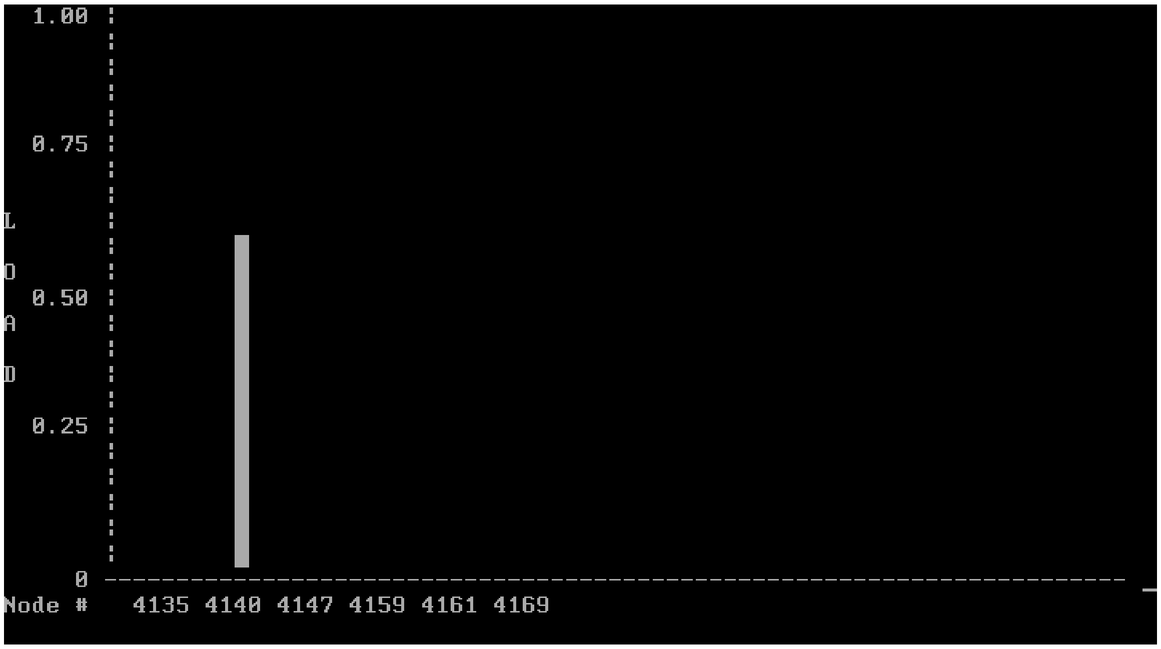
A six node CHAOS/openMosix cluster: The mosmon view with no load

CHAOS is built around the open source project openMosix created by Moshe Bar. openMosix, itself, is a piece of software that is added to the Linux kernel, to allow many Linux computers to work together as a Single System Image (SSI) type cluster.

CHAOS creates a basic node in an openMosix cluster, and is typically not deployed on its own; cluster builders will use feature rich Linux distributions (such as Quantian or ClusterKnoppix) as a "head node" in a cluster to provide their application software, while the CHAOS distribution runs on "drone nodes" to provide "dumb power" to the cluster.

While this deployment model suits the typical cluster builder, openMosix is a peer-based cluster, consisting of only one type of node. All openMosix nodes are inherently equal and each can be, simultaneously, parent and child.

How it works

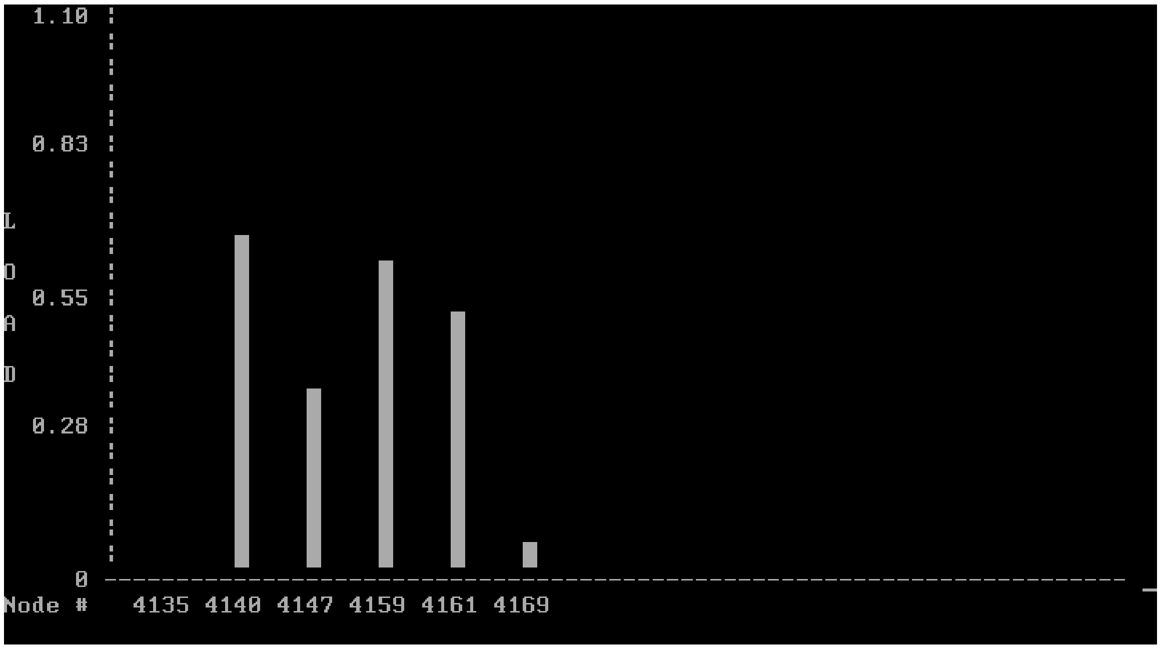


A six node CHAOS/openMosix cluster: The mosmon view with one process' load, launched from node two

As each new node is booted it will locate one cluster node, then negotiate its entry into the cluster. If the IP address of a node is not supplied to the booting node, it will multicast for one. The first responding node will be used as the point of negotiation. The local CHAOS node initiates an IPSEC tunnel to the elected negotiation node using a pre-shared key. If the tunnel fails to establish, the new node is unable to join the cluster. With the tunnel established the new node requests a copy of the openMosix cluster map from the negotiating cluster node. The new node then repeats this process with every node in the cluster map; establishing an IPSEC tunnel, validating the cluster map, then moving on. In this way, every node is interconnected with every other node by "n-1" IPSEC tunnel connections. All openMosix cluster communications are then said to be authenticated and encrypted via the CHAOS platform.

Once an openMosix cluster is established on the CHAOS platform, openMosix can operate as if it were on any Linux platform. Any node can launch a process and have that process migrate to the node with the best performance characteristics for executing that particular process. The openMosix environment has the "mosmon" utility to display the performance of the entire cluster, from any node. The image series on the right shows a six node openMosix cluster running on the CHAOS platform.

Why it was built

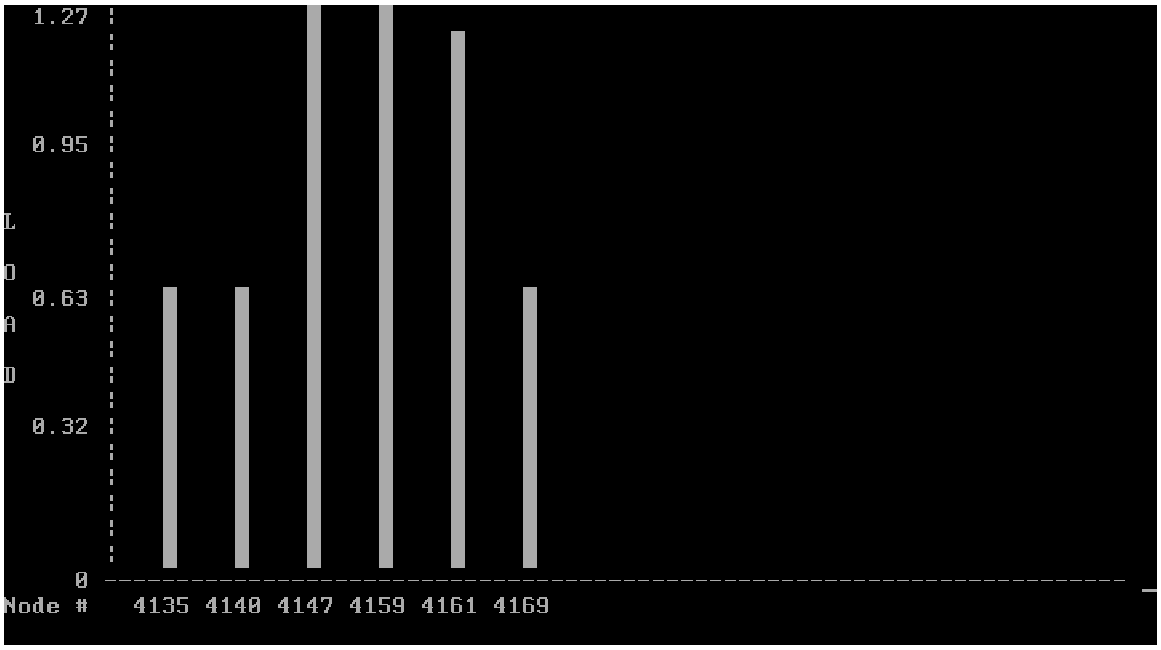


A six node CHAOS/openMosix cluster: The mosmon view with four process' load, launched from node two

CHAOS was developed to utilise idle desktop computer resources to perform pro-active brute-force cryptanalysis against given password hashes. A brute-force attack, as its name suggests, requires an adversary to employ a mammoth work effort into the resolution of a cryptographic problem. Typically, this is an exhaustive search of a particular key-space. For example, resolving the password for three upper-case alpha characters would require exploring the key-space for: AAA, AAB, AAC ... ZZX, ZZY, ZZZ.

In order to reduce the time required to search the key-space, portions of the work effort can be farmed out idle resources. As opposed to rainbow tables this technique allows CHAOS to perform brute-force attacks against irregular or salted algorithms.

Security assessed by



A six node CHAOS/openMosix cluster: The mosmon view with nine process' load, launched from node two

The tool used to provide the cryptographic tests was John the Ripper (JtR). JtR was scaled by using named pipes to funnel a controlled dictionary (a set of keys to try) into an arbitrary number of JtR clients. Each client would take one key, encrypt it, and test it against a local copy of the hash(es). John the Ripper on CHAOS differed from Cisillia as it facilitated dictionary based brute-force attacks across a large number of algorithms, rather than an entire key-space driven brute-force attack across one or two algorithms.

Security provided by

CHAOS was the first openMosix distribution to provide IPSEC and IP packet filtering to the cluster node, enabling authentication and encryption for inter-node communications, and enabling packet filtering to prevent non-cluster devices from accessing the vulnerable openMosix communications ports. These security controls allowed the cluster builder to utilise desktop computers in semi-trusted networks with minimal risk to cluster integrity, thus increasing the number of resources available for inclusion within the cluster.

History

2003: The creation of CHAOS

The project started as tool development work for the IT Security group at Macquarie University in 2003, with an initial team that included Rob Dartnell, Ian Latter and Ty

Miller. There was a need to demonstrate the weakness in one particular application's security via its one hashed, network transmitted, password. The openMosix cluster software, at that time, was available via a number of Linux distributions, but these were neither secure nor dynamic enough to support the campus PC environment that the cluster software was to be deployed into.

The CHAOS distribution was created to fill this need, and was developed under the GPL to allow the openMosix community members to benefit from the security enhancements employed around the openMosix software (the clustering technology that is added to the Linux kernel). Security improvements made by the team included IPSEC tunnels for all cluster communications, state aware packet filtering for each node, a tiny operating system image which allowed for PXE booting to remote PC memory, zero-touch cluster creation, etc.

2004: CHAOS, CoSMoS and team departure

A presentation was made to the Australian Unix Users Group (AUUG) Security Symposium in February 2004 at about two thirds of the way through CHAOS' initial two year development cycle.

In mid to late 2004 CHAOS was adapted to the Cooperative Linux (coLinux) framework, allowing openMosix to run as a node on a Microsoft Windows PC for the first time. This was significant as there was now the ability to run ad-hoc clusters 24x7, and not just out of business hours. The version of CHAOS created for coLinux was dubbed CosMos (Chaos-OS on Microsoft-OS) and was also released under the GPL, complete with Windows installer software.

Later that year work stalled on CHAOS and CosMos when the IT Security team broke up to work for various organisations. Development halted for most of the six months beginning Q4 2004.

2005: Relocation and public dissemination

There was renewed interest in CHAOS development when both Ian and Ty began work at Pure Hacking in Q2 2005. Pure Hacking could identify a need with the resource that CHAOS provided and offered to sponsor further CHAOS development so that it could remain under the GPL. A package updated version of CHAOS was released at that stage, but Pure Hacking provided no additional development time, leaving the project to grind to a halt again. CHAOS was "Slashdotted" during this time, due to the press that came from Pure Hacking's sponsorship announcement. Unfortunately, Pure Hacking were unable to provide the time needed to develop or maintain CHAOS. Version 1.6 of CHAOS, the only version released in Q1-3 of 2005, was released from development work performed in private time.

In Q4 2005 Ian added CHAOS to the midnightcode.org web site (at the location advertised when leaving the University in 2004) - in the hope of better maintaining the project. Improvements desperately needed include code and protocol clean ups, better enterprise management support, operational documentation, and simpler integration with the supporting openMosix distributions (Quantian and ClusterKnoppix).

2006-2007: Redevelopment

Many of the code clean-up issues (focused on Init and Tyd, particularly) will be resolved with the integration of the Midnight Code libraries. While currently being developed these libraries already provide better program execution, configuration control, network interface manipulation and status management than those currently in CHAOS.

FhGFS

	FhGFS
Developer	Fraunhofer ITWM
Full name	FraunhoferFS
Introduced	2007 (Linux)

The **Fraunhofer Parallel File System**, also known as **FraunhoferFS** and abbreviated **FhGFS**, is a parallel file system, optimized for usage in the field of High Performance Computing. The most important aspect is data throughput. It is developed at the Fraunhofer Institute for Industrial Mathematics (ITWM) in Germany. FhGFS can be downloaded and used free of charge.

Design

Access to user data is parallelized by dividing the data into so-called chunks. The chunks are stored distributed and independent of each other on multiple servers. The management of metadata and the mapping of a file to the appropriate chunks is done by a special metadata server. FhGFS supports distributed metadata over multiple servers. Therefore file access should scale very well. The internal connection of the servers is done by using either Infiniband (RDMA) or TCP/IP connections (Ethernet, IPoIB).

The developers put their priorities on scalability, flexibility (e.g. support of multiple network interfaces and dynamic failover) and ease of use (e.g. by integrating their own graphical tools).

Deployment

FhGFS is currently deployed at several supercomputing sites, including some of the fastest computer clusters in the world.

Chapter-11

FinisTerrae and VMScluster

FinisTerrae

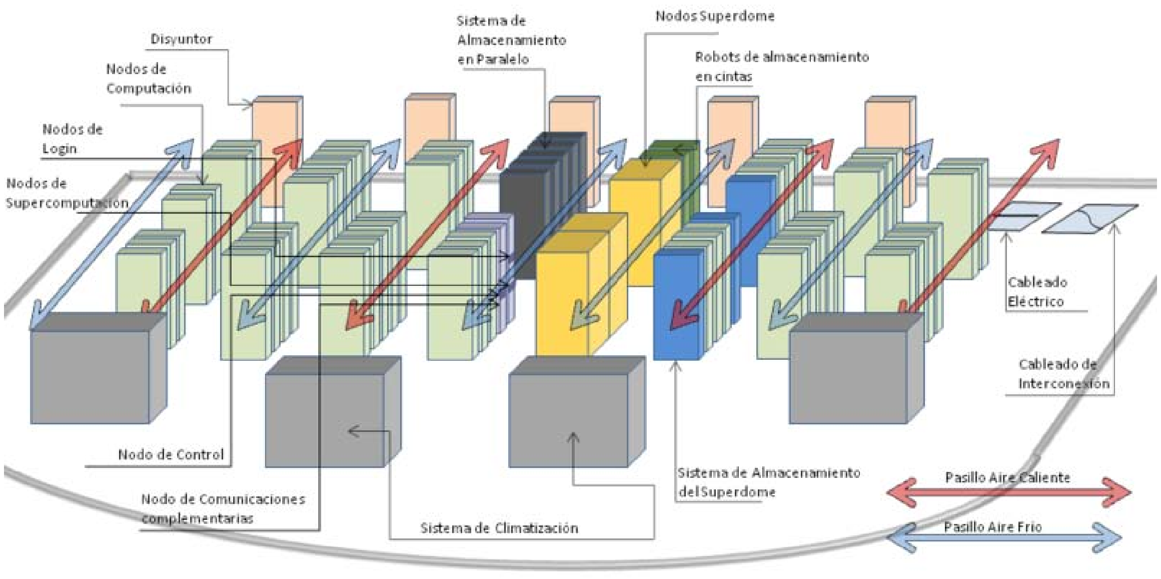
Finisterrae

Location	CESGA Santiago de Compostela (La Coruña) Spain
Coordinates	42°52′33″N 8°33′12″W / 42.875833°N 8.553333°W
Staff	16

Finisterrae was the 100th supercomputer in Top500 ranking in November 2007. Running at 12.97 Teraflops, it would rank at position 258 on the list as of June 2008. It is also the third most powerful supercomputer in Spain (after MareNostrum and Magerit). It is located in Galicia.

This project is promoted by the Xunta de Galicia (regional government of Galicia) and the Spanish National Research Council (CSIC). It was founded in the year 1993 to serve as a platform to foster scientific innovation and invest in Research and Development.

It is estimated that the base project will be completed in 2010. It is expected to reach the TOP10 of the most powerful supercomputers in the world when it reaches full capacity. The supercomputer is physically hosted at CESGA.



Disyuntor
Nodos de Computación
Nodos de Login
Nodos de Supercomputación
Sistema de Almacenamiento en Paralelo
Nodos Superdome
Robots de almacenamiento en cintas
Cableado Eléctrico
Cableado de Interconexión
Nodo de Control
Nodo de Comunicaciones complementarias
Sistema de Climatización
Sistema de Almacenamiento del Superdome
Pasillo Aire Caliente
Pasillo Aire Frio

FinisTerrae Distribution Diagram

Overview

The main Finisterrae characteristics are depicted on the following table:

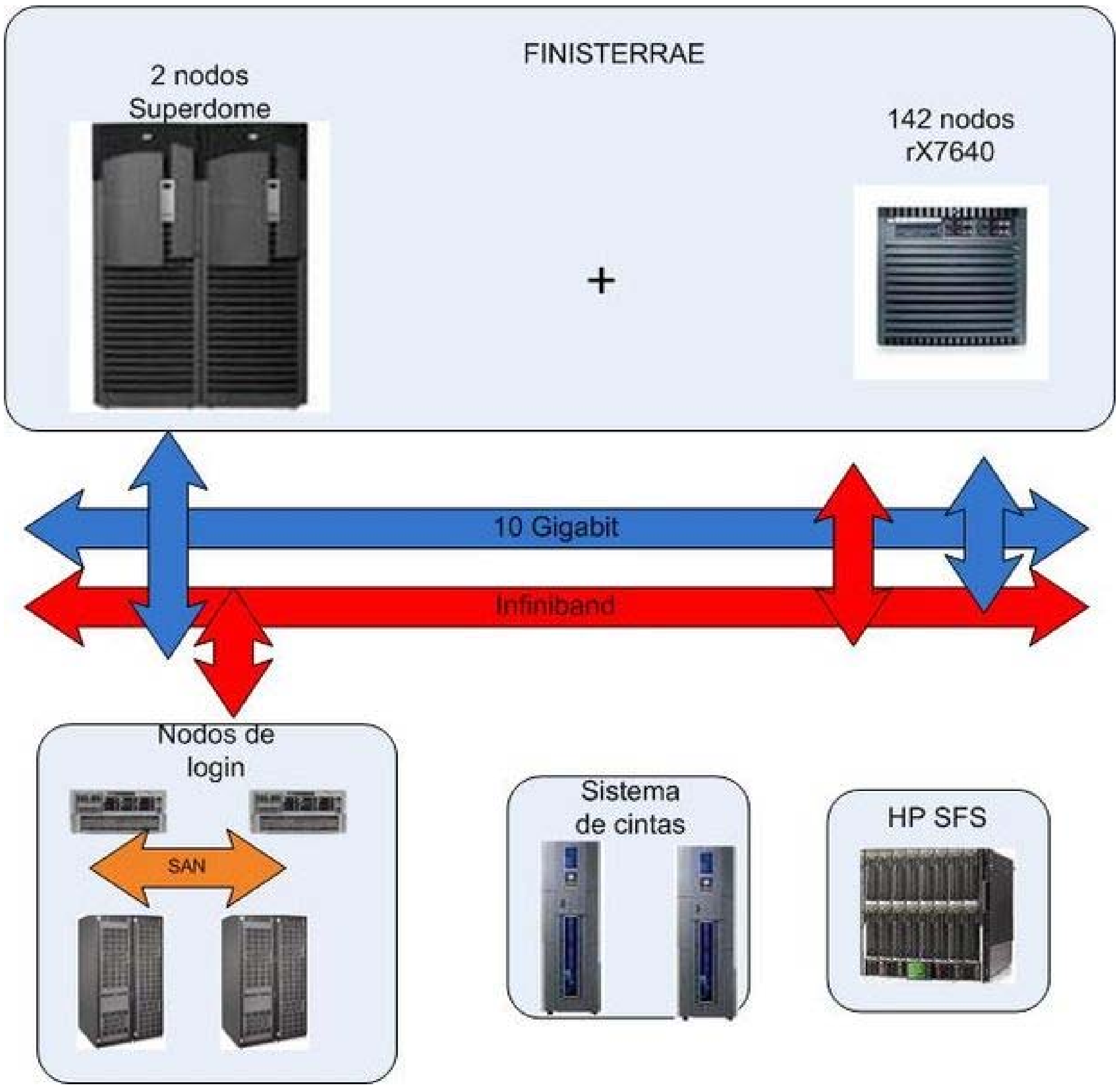
Name	Finisterrae
Place	CESGA
Country	Spain
Year	2007
OS	Linux
Architecture	Itanium 2 64 (Intel)
Processors	2400
Memory	19584 GB
Power	12970 GFLOPS
Top500 Ranking	100

One of the special characteristic about this supercomputer is the ratio between cores and RAM memory. This was one reason why it received the denomination of "singular technical and scientific installation" from the Spanish government, a denomination given to some installations which have some value that makes them singular in some way.

Some of those installations include the Canary Island grand telescope, or the Alba synchrotron.

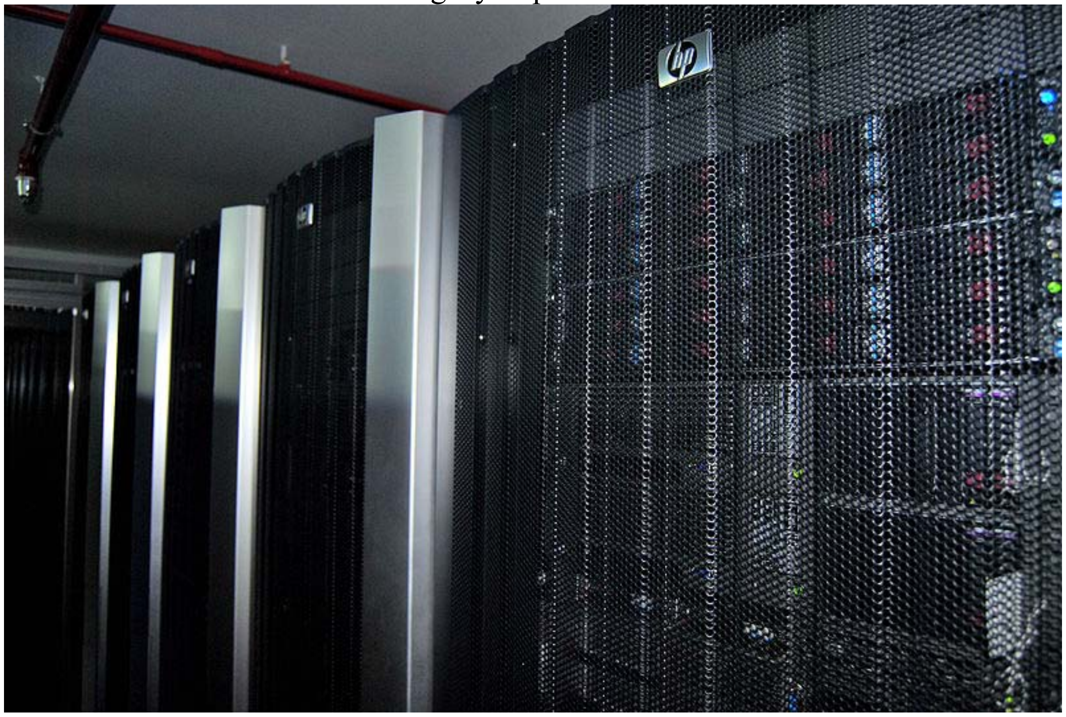
Even if this is the third fastest supercomputer in Spain, some projects that require special amounts of memory cannot be held by the first or second supercomputer, and therefore must be executed on the Finisterrae.

Architecture



FinisTerrae Architecture

HP Integrity Superdome Nodes



HP-SFS storage system Nodes

HP Integrity RX7640 Nodes

HP ESL 712e Tape Library

FinisTerrae supercomputer, located in CESGA is an integrated system by shared-memory nodes with and SMP NUMA architecture. FinisTerrae is composed of

144 computational nodes:

- 142 hp (106 kW) HP Integrity rx7640 nodes with 16 Itanium Montvale cores and 128 GB memory each one.
- 1 hp (0.75 kW) Integrity Superdome node, with 128 Itanium Montvale cores and 1024 GB memory.
- 1 hp (0.75 kW) Integrity Superdome node, with 128 Itanium 2 cores and 384 GB memory.

An hierarchic storing system with:

- 22 nodes for storing management with 96 processing cores.
- The storage is connected to 72 cabins connected to 864 SATA disks, managed by Lustre. Backups are stored on tapes which provide 2.200.000 GB.

- An interconnecting network between Infiniband 4x DDR at 20 Gbit/s nodes, providing 16 Gbit/s of effective bandwidth.

- An external management Gigabit Ethernet network.

- The nodes user SUSE Linux enterprise server as the Operating System. This operating system helped saving an important amount of money when comparing to other non-free options.

- FinisTerrae includes open software and standard like: Linux, Lustre, Grid Engine and Globus.

- System also has next compilators, libraries and development tools: Intel C/C y Fortran, Intel MKL, Vtune, HP-MPI y HP UPC.

Projects

As main purpose, the aim of the supercomputer is research. The supercomputer is mainly used by the three universities located in Galicia, (Universidade da Coruña, Universidade de Santiago de Compostela and Universidade de Vigo), as well as other research organizations like CSIC. The main projects hold by the supercomputer are divided into four fields:

- Life sciences.
- Nanotechnology.
- Sea sciences.
- Renewable energies.

Curiosities

- The machines need 140 square meters of space.
- It weights around 33,5 tons.
- The amount of cables used to interconnect the nodes is around 85 km, the distance between the supercomputer location (Santiago) and Finisterre.

VMScluster

A **VMScluster** is a computer cluster involving a group of computers running the OpenVMS operating system. Whereas tightly-coupled multiprocessor systems run a single copy of the operating system, a VMScluster is loosely-coupled: each machine runs its own copy of OpenVMS, but the disk storage, lock manager, and security domain are all cluster-wide. Machines can join or leave a VMScluster without affecting the rest of the cluster. For enhanced availability, VMSclusters support the use of dual-ported disks connected to two machines or storage controllers simultaneously. With OpenVMS now ported to Alpha and IA-64 machines, the facility originally named VAXclustering was renamed to VMSclustering.

Initial release

Digital Equipment Corporation first announced VAXclusters in May 1983. At this stage, clustering required specialised communications hardware, as well as some major changes to low-level subsystems in VMS. The software and hardware were designed jointly.

At the center of each cluster was a star coupler, to which every *node* (computer) and data storage device in the cluster was connected by one or two pairs of *CI cables*. ("CI" stands for Computer Interconnect.) Each pair of cables had a transmission rate of 70 megabits per second, a high speed for that era. Using two pairs gave an aggregate transmission rate of 140 megabits per second, with redundancy in case one cable failed; the star couplers also had redundant wiring for better availability.

Each CI cable connected to its computer via a *CI Port*, which could send and receive packets without any CPU involvement. To send a packet, a CPU had only to create a small data structure in memory and append it to a "send" queue; similarly, the CI Port would append each incoming message to a "receive" queue. Tests showed that a VAX-11/780 could send and receive 3000 messages per second, even though it was nominally a 1-MIPS machine. The closely-related Mass Storage Control Protocol (MSCP) allowed similarly-high performance from the mass storage subsystem. In addition, MSCP packets were very easily transported over the CI allowing remote access to storage devices.

VAXclustering was the first clustering system to achieve commercial success, and was a major selling point for VAX systems.

Later developments

In 1986, DEC added VAXclustering support to their MicroVAX minicomputers, running over Ethernet instead of special-purpose hardware. While not giving the high-availability advantages of the CI hardware, these *Local Area VAXclusters* provided an attractive expansion path for buyers of low-end minicomputers.

Later versions of OpenVMS (V5.0 and later) supported "mixed interconnect" VAXclusters (using both CI and Ethernet), and VAXclustering over DSSI (Digital Systems and Storage Interconnect) and FDDI, among other transports. Eventually, as high-bandwidth wide area networking became available, clustering was extended to allow satellite data links and long-distance terrestrial links. This allowed the creation of *disaster-tolerant clusters*; by locating the single VAXcluster in several diverse geographical areas, the cluster could survive infrastructure failures and natural disasters.

VAXclustering was greatly aided by the introduction of terminal servers using the LAT protocol. By allowing ordinary serial terminals to access the host nodes via Ethernet, it became possible for any terminal to rapidly and easily connect to any host node. This made it much simpler to accomplish fail over of the user terminals from one node of the cluster to another.

Eventually, VAXclusters reached the point where the cluster as a whole essentially never went down. *Rolling upgrades* even allowed the system operators to upgrade the OpenVMS system software, shutting down, upgrading, and rebooting individual nodes while the cluster as a whole continued processing. Cluster uptimes are frequently measured in years with the current longest uptime being at least twelve years.

As mentioned above, OpenVMS now also runs on Alpha and IA-64 systems, so the term *VAXcluster* has been replaced by *VMScluster*. With Gigabit Ethernet now common and 10-gigabit Ethernet being introduced, standard networking cables and cards are quite sufficient to support VMSclustering.

Chapter-12

Solaris Cluster, Veritas Cluster Server and GPU Cluster

Solaris Cluster

Solaris Cluster (sometimes **Sun Cluster** or **SunCluster**) is a high-availability cluster software product for the Solaris Operating System, created by Sun Microsystems, a subsidiary of Oracle Corporation. It is used to improve the availability of software services such as databases, file sharing on a network, electronic commerce websites, or other applications. Sun Cluster operates by having redundant computers or **nodes** where one or more computers continue to provide service if another fails. Nodes may be located in the same data center or on different continents.

Sun Microsystems Solaris Cluster

Background

Solaris Cluster provides services that remain available even when individual nodes or components of the cluster fail. Solaris Cluster provides two types of HA services: failover services and scalable services.

To eliminate single points of failure, a Solaris Cluster configuration has redundant components, including multiple network connections and data storage which is multiply connected via a storage area network. Clustering software such as Solaris Cluster is a key component in a Business Continuity solution, and the Solaris Cluster Geographic Edition was created specifically to address that requirement.

Solaris Cluster is an example of kernel-level clustering software. Some of the processes it runs are normal system processes on the systems it operates on, but it does have some special access to operating system or kernel functions in the host systems.

In June 2007, Sun released the source code to Solaris Cluster via the OpenSolaris HA Clusters community.

Solaris Cluster Geographic Edition

SCGE is a management framework that was introduced in August 2005. It enables two Solaris Cluster installations to be managed as a unit, in conjunction with one or more Data replication products, to provide Disaster Recovery for a computer installation. By ensuring that data updates are continuously replicated to a remote site in near-real time, that site can rapidly take over the provision of a service in the event that the entire primary site is lost as a result of a disaster, either natural or man-made. This is a key to minimizing the Recovery point objective (RPO) and Recovery time objective (RTO) for the service.

SCGE today supports replication with Sun StorageTek AVS, EMC SRDF and Hitachi TrueCopy. Support for application-based replication with both Oracle Data Guard and MySQL is planned for early 2009.

Proxy file system

PxFS (Proxy file system) is a distributed, high availability, POSIX compliant filesystem internal to Solaris Cluster nodes. Global devices in Sun Cluster are made possible by PxFS.

Supported applications

Solaris Cluster uses software components called *agents* which monitor an application to detect whether it is operating correctly, and take action if a problem is detected. Agents for common applications are included such as Siebel Systems, SAP Livecache, WebLogic Server, Sun Java Application Server, MySQL, Oracle RAC, Oracle E-

Business Suite and Samba among others; there is also a wizard which allows the cluster implementer to create agents for other applications.

Veritas Cluster Server

Veritas Cluster Server (also known as **VCS** and also sold bundled in the **SFHA** product) is a High-availability cluster software, for Unix, Linux and Microsoft Windows computer systems, created by Veritas Software (now part of Symantec). It provides application cluster capabilities to systems running databases, file sharing on a network, electronic commerce websites or other applications.

Description

High availability clusters (HAC) improve *availability* of applications by *failing* them over or *switching* them over in a group of systems as opposed to High Performance Clusters which improve *performance* of applications by allowing them to run on multiple systems simultaneously.

Most Veritas Cluster Server implementations attempt to build availability into a cluster, eliminating single points of failure by making use of redundant components like multiple network cards, storage area networks in addition to the use of VCS.

Similar products include Fujitsu PRIMECLUSTER, IBM HACMP, HP ServiceGuard, IBM Tivoli System Automation for Multiplatforms (SA MP), Linux-HA, Microsoft Cluster Server (MSCS), NEC ExpressCluster, Red Hat Cluster Suite, SteelEye LifeKeeper and Sun Cluster. VCS is one of the few products in the industry that provides both high availability and disaster recovery across all major operating systems while supporting 40+ major application / replication technologies out of the box.

VCS is mostly a user-level clustering software; most VCS processes are normal system processes on the systems it operates on, and have no special access to the Operating System or kernel functions in the host systems. However, the interconnect (heartbeat) technology used with VCS is a proprietary Layer 2 ethernet-based protocol that is run in the kernel space using kernel modules.. The group membership protocol that runs on top of the interconnect heartbeat protocol is also implemented in the kernel.

Veritas Cluster Server for Windows is available as a standalone product. It is also sold bundled with Storage Foundation as Storage Foundation HA for Windows; Veritas Cluster Server for AIX, HP-UX, Linux, and Solaris is supplied as a standalone product.

The Veritas Cluster Server product includes VCS Management Console, a multi-cluster management software that automates disaster recovery across data centers.

Release history

- From its second version, it has supported the AIX operating system.
- Veritas Cluster Server 5.0
- Veritas Cluster Server 5.1

GPU cluster

A **GPU cluster** is a computer cluster in which each node is equipped with a Graphics Processing Unit (GPU). By harnessing the computational power of modern GPUs via General-Purpose Computing on Graphics Processing Units (GPGPU), very fast calculations can be performed with a GPU cluster.

Hardware (GPU)

The hardware classification of GPU clusters fall into two categories: Heterogeneous and Homogeneous.

Heterogeneous

Hardware from both of the major IHV's can be used (ATi and nVidia). Even if different models of the same GPU are used (i.e. 8800GT mixed with 8800GTX) the gpu cluster is considered heterogeneous.

Homogeneous

Every single GPU is of the same hardware class, make, and model. (i.e. a homogeneous cluster comprising 100 8800GTs, all with the same amount of VRAM)

Classifying a GPU cluster according to the above semantics largely directs software development on the cluster, as different GPUs have different capabilities that can be utilized.

Hardware (Other)

Interconnect

In addition to the computer nodes and their respective GPUs, a fast enough interconnect is needed in order to shuttle data amongst the nodes. The type of interconnect largely depends on the number of nodes present. Some examples of interconnects include Gigabit Ethernet and InfiniBand.

Software

The software components that are required to make many GPU-equipped machines act as one include:

1. Operating System
2. GPU driver for the each type of GPU present in each cluster node.
3. Clustering API (such as the Message Passing Interface, MPI).

Algorithm mapping

Mapping an algorithm to run a GPU cluster is somewhat similar to mapping an algorithm to run on a traditional computer cluster. Example: rather than distributing pieces of an array from RAM, a texture is divided up amongst the nodes of the GPU cluster.

Chapter-13

High-availability Cluster

High-availability clusters (also known as **HA Clusters** or **Failover Clusters**) are computer clusters that are implemented primarily for the purpose of providing high availability of services which the cluster provides. They operate by having redundant computers or **nodes** which are then used to provide service when system components fail. Normally, if a server with a particular application crashes, the application will be unavailable until someone fixes the crashed server. HA clustering remedies this situation by detecting hardware/software faults, and immediately restarting the application on another system without requiring administrative intervention, a process known as Failover. As part of this process, clustering software may configure the node before starting the application on it. For example, appropriate filesystems may need to be imported and mounted, network hardware may have to be configured, and some supporting applications may need to be running as well.

HA clusters are often used for critical databases, file sharing on a network, business applications, and customer services such as electronic commerce websites.

HA cluster implementations attempt to build redundancy into a cluster to eliminate single points of failure, including multiple network connections and data storage which is multiply connected via Storage area networks.

HA clusters usually use a **heartbeat** private network connection which is used to monitor the health and status of each node in the cluster. One subtle, but serious condition all clustering software must be able to handle is split-brain. Split-brain occurs when all of the private links go down simultaneously, but the cluster nodes are still running. If that happens, each node in the cluster may mistakenly decide that every other node has gone down and attempt to start services that other nodes are still running. Having duplicate instances of services may cause data corruption on the shared storage.

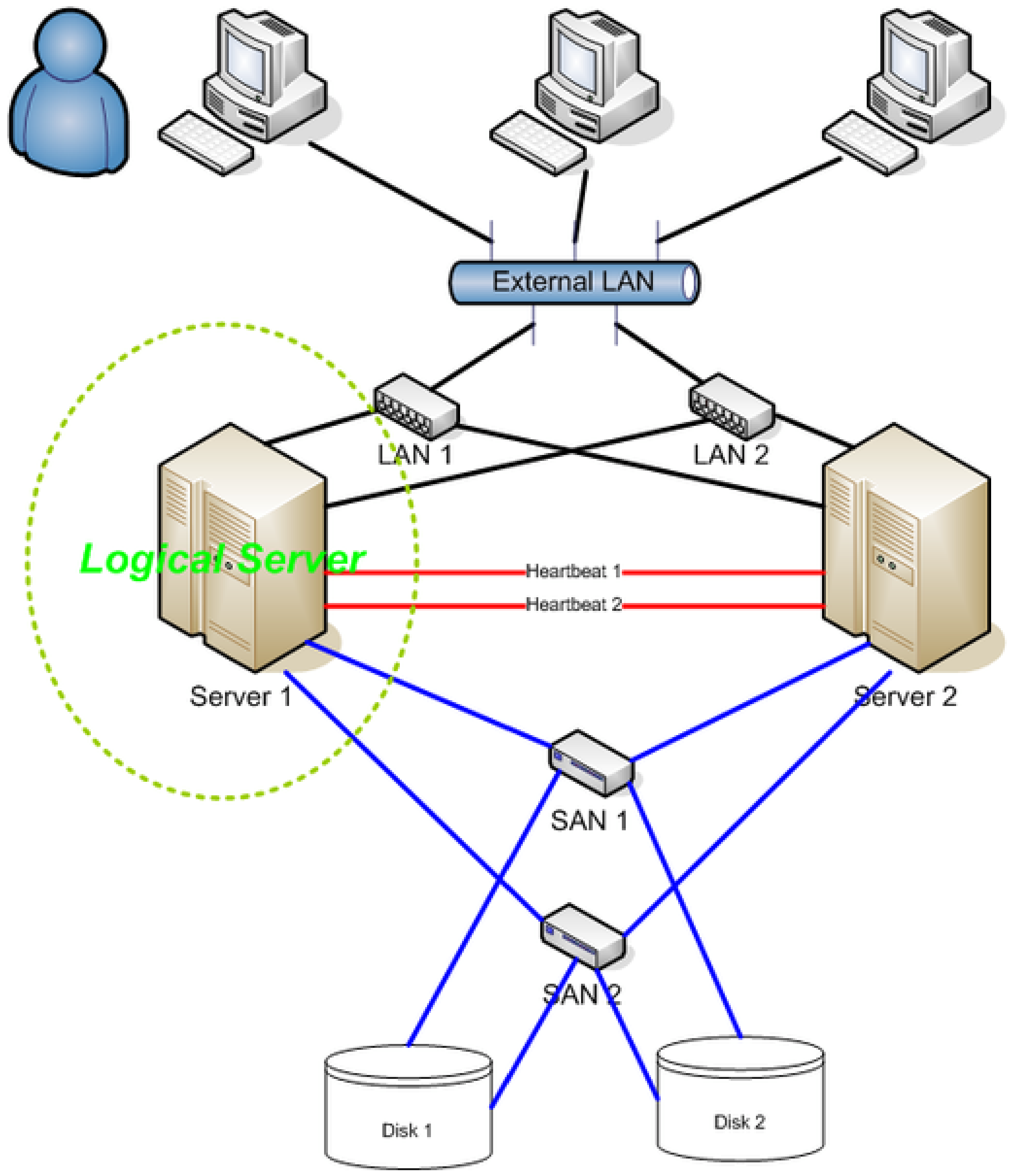
Application Design Requirements

Not every application can run in a high-availability cluster environment, and the necessary design decisions need to be made early in the software design phase. In order to run in a high-availability cluster environment, an application must satisfy at least the following technical requirements:

- There must be a relatively easy way to start, stop, force-stop, and check the status of the application. In practical terms, this means the application must have a command line interface or scripts to control the application, including support for multiple instances of the application.
- The application must be able to use shared storage (NAS/SAN).
- Most importantly, the application must store as much of its state on non-volatile shared storage as possible. Equally important is the ability to restart on another node at the last state before failure using the saved state from the shared storage.
- The application must not corrupt data if it crashes, or restarts from the saved state.

The last two criteria are critical to reliable functionality in a cluster, and are the most difficult to satisfy fully. Finally, licensing compliance must be observed.

Node configurations



2 node High Availability Cluster network diagram

The most common size for an HA cluster is a two-node cluster, since that's the minimum required to provide redundancy, but many clusters consist of many more, sometimes dozens of nodes. Such configurations can sometimes be categorized into one of the following models:

- Active/Active — Traffic intended for the failed node is either passed onto an existing node or load balanced across the remaining nodes. This is usually only possible when the nodes utilize a homogeneous software configuration.

- Active/Passive — Provides a fully redundant instance of each node, which is only brought online when its associated primary node fails. This configuration typically requires the most extra hardware.
- N+1 — Provides a single extra node that is brought online to take over the role of the node that has failed. In the case of heterogeneous software configuration on each primary node, the extra node must be universally capable of assuming any of the roles of the primary nodes it is responsible for. This normally refers to clusters which have multiple services running simultaneously; in the single service case, this degenerates to Active/Passive.
- N+M — In cases where a single cluster is managing many services, having only one dedicated failover node may not offer sufficient redundancy. In such cases, more than one (M) standby servers are included and available. The number of standby servers is a tradeoff between cost and reliability requirements.
- N-to-1 — Allows the failover standby node to become the active one temporarily, until the original node can be restored or brought back online, at which point the services or instances must be failed-back to it in order to restore High Availability.
- N-to-N — A combination of Active/Active and N+M clusters, N to N clusters redistribute the services or instances from the failed node among the remaining active nodes, thus eliminating (as with Active/Active) the need for a 'standby' node, but introducing a need for extra capacity on all active nodes.

The term **Logical host** or **Cluster logical host** is used to describe the network address which is used to access services provided by the cluster. This logical host identity is not tied to a single cluster node. It is actually a network address/hostname that is linked with the service(s) provided by the cluster. If a cluster node with a running database goes down, the database will be restarted on another cluster node, and the network address that the users use to access the database will be brought up on the new node as well so that users can access the database again.

Node reliability

HA clusters usually utilize all available techniques to make the individual systems and shared infrastructure as reliable as possible. These include:

- Disk mirroring so that failure of internal disks does not result in system crashes.
- Redundant network connections so that single cable, switch, or network interface failures do not result in network outages.
- Redundant Storage area network or SAN data connections so that single cable, switch, or interface failures do not lead to loss of connectivity to the storage (this would violate the share-nothing architecture).
- Redundant electrical power inputs on different circuits, usually both or all protected by Uninterruptible power supply units, and redundant power supply units, so that single power feed, cable, UPS, or power supply failures do not lead to loss of power to the system.

These features help minimize the chances that the clustering failover between systems will be required. In such a failover, the service provided is unavailable for at least a little while, so measures to avoid failover are preferred.

Failover strategies

Systems that handle failures in distributed computing do have different strategies to get rid of a failure. For instance, the Apache Cassandra API Hector (API) defines three ways to configure a failover:

- FAIL_FAST: The try fails, if the first node cannot be reached.
- ON_FAIL_TRY_ONE_NEXT_AVAILABLE: Tries one more host before giving up
- ON_FAIL_TRY_ALL_AVAILABLE: Tries all existing nodes before giving up

Chapter-14

Common Address Redundancy Protocol and Linux Virtual Server

Common Address Redundancy Protocol

The **Common Address Redundancy Protocol** or **CARP** is a protocol which allows multiple hosts on the same local network to share a set of IP addresses. Its primary purpose is to provide failover redundancy, especially when used with firewalls and routers. In some configurations CARP can also provide load balancing functionality. It is a free, non patent-encumbered alternative to Cisco's HSRP, implemented mostly in BSD operating systems.

Example

If there is a single computer running a packet filter, and it goes down, the networks on either side of the packet filter can no longer communicate with each other, or they communicate without any packet filtering. If, however, there are two computers running a packet filter, running CARP, then if one fails, the other will take over, and computers on either side of the packet filter will not be aware of the failure, so operation will continue as normal. In order to make sure the new master operates the same as the old one, pfsyncd is used to synchronize packet filter states.

Principle of redundancy

A group of hosts using CARP is called a "group of redundancy". The group of redundancy allocates itself an IP address which is shared or divided among the members of the group. Within this group, a host is designated as "Master". The other members are called "slaves". The main host is that which "takes" the IP address. It answers any traffic or ARP request brought to the attention of this address. Each host can belong to several groups of redundancy. Each host must have a second unique IP address.

A common use of CARP is the creation of a group of redundant firewalls. The virtual IP address allotted to the group of redundancy is indicated as the address of the default router on the computers behind this group of firewalls. If the main firewall breaks down or is disconnected from the network, the virtual IP address will be taken by one of the firewall slaves and the service availability will not be interrupted.

History

In the late 1990s IETF began working on a solution to the problem of shared IPs. In 1997, Cisco informed them that this was already covered by Cisco patents. In 1998, Cisco told them it was covered by their patent of HSRP (Hot Standby Router Protocol). Nonetheless, IETF continued work on VRRP (Virtual Router Redundancy Protocol). After some debate, people decided it was alright to allow patented material in a standard, as long as it was available under RAND (Reasonable and Non-Discriminatory) Licensing terms. Because VRRP fixed problems with the HSRP protocol, Cisco began using VRRP instead, while still claiming it as its own.

Cisco informed the OpenBSD developers they would enforce their patent of HSRP. This may have been related to their lawsuit with Alcatel. Thus, a free implementation of VRRP could not be made. OpenBSD developers started CARP as an alternative to the patented VRRP, as the "reasonable and non-discriminatory" licensing terms necessarily excluded open-source implementations. To avoid infringing the HSRP patent, they ensured their idea for CARP was fundamentally different. Because of OpenBSD's focus on security, CARP was designed with security in mind, and is designed to use cryptography. It became available, completely for free, in October 2003. It was integrated into FreeBSD and released initially with FreeBSD 5.4 in May 2005. It has since been integrated into NetBSD.

No official internet protocol number

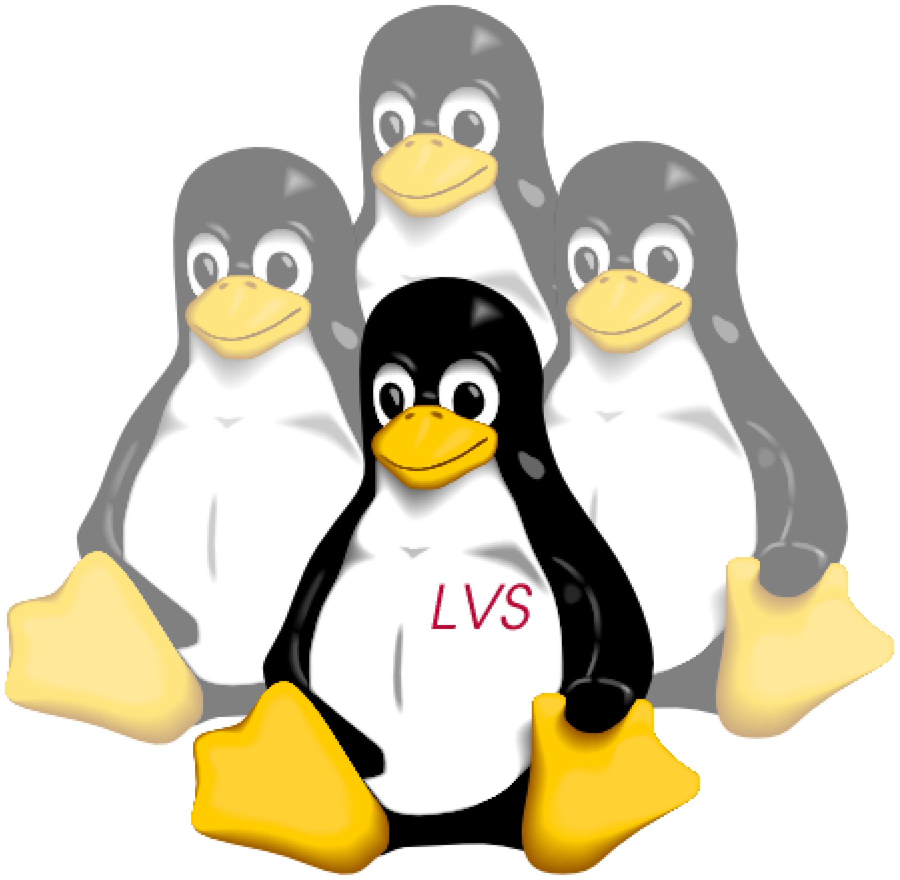
From OpenBSD.org:

As a final note of course, when we petitioned IANA, the IETF body regulating "official" internet protocol numbers, to give us numbers for CARP and pfsync our request was denied. Apparently we had failed to go through an official standards organization. Consequently we were forced to choose a protocol number which would not conflict with anything else of value, and decided to place CARP at IP protocol 112. We also placed pfsync at an open and unused number. We informed IANA of these decisions, but they declined to reply.

IP protocol numbers at the time when the above request was made were allocated by IANA according to the rules in RFC 2780, i.e., under the "IESG Approval" or "Standards Action" process (with "Expert Review" being a third option that was not applicable to this request). Both of these processes require a textual specification describing the protocol for which a protocol number is requested, which did not exist for CARP. The OpenBSD implementation is the closest thing to a formal specification of the protocol, but source code - especially source code licensed under specific terms - is not the same as a textual technical specification. No technical specification was submitted for CARP, and IANA declined the request.

The incompatible Cisco/IETF VRRP also uses IP protocol 112, having been assigned it by IANA.

Linux Virtual Server



Linux Virtual Server (**LVS**) is an advanced load balancing solution for Linux systems. It is an open source project started by Wensong Zhang in May 1998. The mission of the project is to build a high-performance and highly available server for Linux using clustering technology, which provides good scalability, reliability and serviceability.

The major work of the LVS project is now to develop advanced IP load balancing software (IPVS), application-level load balancing software (KTCPVS), and cluster management components.

- IPVS: is an advanced IP load balancing software implemented inside the Linux kernel. The IP Virtual Server code was already included into the standard Linux kernel 2.4 and 2.6.
- KTCPVS: implements application-level load balancing inside the Linux kernel, currently under development.

Users can use the LVS solutions to build highly scalable and highly available network services, such as web, email, media services and VoIP services, and integrate scalable network services into large-scale reliable e-commerce or e-government applications.

The LVS component depends upon the Linux Netfilter framework and its source code is available in the `net/netfilter/ipvs/` subdirectory within the kernel source. It implements several balancing schedulers, listed below with the relevant source file:

- Round-Robin (`ip_vs_rr.c`)
- Weighted Round-Robin (`ip_vs_wrr.c`)
- Least-Connection (`ip_vs_lc.c`)
- Weighted Least-Connection (`ip_vs_wlc.c`)
- Locality-Based Least-Connection (`ip_vs_lblc.c`)
- Locality-Based Least-Connection with Replication (`ip_vs_lblcr.c`)
- Destination Hashing (`ip_vs_dh.c`)
- Source Hashing (`ip_vs_sh.c`)
- Shortest Expected Delay (`ip_vs_sed.c`)
- Never Queue (`ip_vs_nq.c`)

The module is able to handle UDP, TCP layer-4 protocols as well as FTP passive connection by inspecting layer-7 packets. It provides a hierarchy of counters in the `/proc/` directory.

The userland tool is `ipvsadm`.

Chapter-15

Computer Cluster in Virtual Machines and Two-node Cluster

Computer cluster in virtual machines

Computer clusters run usually on physical computers. With the virtualization approach there are new possibilities of setting up different kinds of clusters.

There are different categorizations of such clusters, depending

- on the type of underlying virtualization, and
- of the level, where some Cluster Manager is running.

Note that the implementation sections give only some typical examples of the described cases - this is not a list of all available configurations.

Host System

The *Host System* is the hardware and the virtualization software, like a hypervisor.

Guest System

The *Guest System* is the operating system that is running atop a *Host System*. Sometimes this is called *Virtual Machine*, but mostly every vendor of virtualization technology has its own name for this.

Cluster Manager

The *Cluster Manager* as used here, is software handling all the things that are needed in a computer cluster environment, e. g. supervision of resources (processes, IP Addresses, ...), restarting of resources, handling of failover.

Cluster Service

A *Cluster Service* is a service (application) running under control of a Cluster Manager. This is sometimes also called *Process Resource*.

Virtual vs. Physical

Virtual Cluster Nodes on one Host System

All the cluster nodes [note: a definition for node is needed above as that term is not defined or linked - one can only assume that "node" is synonymous to "guest" as defined above when used in a cluster] are all running on the same Host System. This does not increase any availability of the applications in case of a hardware outage, therefore it is mostly only used during development and testing of a Cluster Service.

Example: LinuxHA on Xen

Different Guest Systems run on one Host System. The Host System uses Xen as virtualization technology. The Guest Systems use Linux-HA for the high availability. [note: this example does not help in clarifying the concepts being defined and appears a gratuitous mention of a specific vendor]

Virtual Cluster Nodes on different Host Systems

All nodes of the cluster are running on different Host Systems. This gives the same availability of the underlying cluster technology in case of a hardware outage and also gives the possibility to live migrate one virtual cluster node to another Host System for maintenance of a Host System.

Virtual - Physical Cluster

This scenario only makes sense for high availability failover clusters. One node is running on the physical machine the other on a Guest System. The normal way of operation is, that everything runs on the physical machine, only when there is a failure, the node on the Guest System gets control. The advantage of this scenario is, that it is possible to run some failover nodes in some Guest Systems of the same Host System. So it is possible to save mostly all of the additional second nodes. The disadvantage is, that the virtualized computer may not handle the load, when all physical nodes stop working at the same time.

Cluster Manager Level

Cluster Manager only in Guest Systems

The Cluster Manager runs only on a Guest System level. Two (or more) Guest Systems run as a cluster.

Example: Solaris Cluster in Guest Systems

There is an experiment from SUN running Solaris Cluster atop VMware.

Example: openMosix for Xen

It is possible to run different Guest Systems atop Xen as an openMosix cluster.

Cluster Manager only on Host Systems

The Cluster Manager is running on the Host System level. It supervises the Guest Systems themselves and can react on some event: e. g. to restart the Guest System on another machine.

Example: VMware HA

The VMware HA can restart Guest Systems when they fail.

Independent Cluster Manager on Guest and Host Systems

This is adds the features of the Guest Only and the Host Only Clustering.

Integrated Cluster Manager on Guest and Host System

The Cluster Manager is running in the Host and Guest System level and knows about resources (e.g. processes) inside a Guest System and also about whole Guest Systems. It can therefor react on problems inside a Guest System or even of a problem with a whole Guest System, when e.g. the Guest System hangs.

Benefits

For HA clusters, the benefit for using virtual nodes is that even during hardware maintenance, all cluster nodes stay available. This can be achieved live by migrating one Virtual Cluster Node from the Host System that must be maintained to some other Host System.

Two-node cluster

A **two-node cluster** is the minimal high-availability cluster that can be built. Should one *node* fail (for a hardware or software problem), the other must acquire the resources being previously managed by the failed node, in order to re-enable access to these resources. This process is known as *failover*.

Introduction and some definitions

There are various kinds of resources, notably:

- *Storage space* (containing data, binaries, or everything else that needs to be accessed)
- *Network address(es)* (the users can reach the resources via a network connection)
- *Application software* (that acts as an interface between users and other resources)

Typical *services* provided by a computer cluster are built from a combination of each of the previously defined resources.

As an example, an Oracle database service might be composed of:

- some storage space, to hold the database files (and, ultimately, the data);
- an Oracle installation, configured to be remotely (or locally) accessed; and,
- an IP address to listen on; the users must connect to this address in order to use Oracle to access the data.

Hardware components

Required

- Two hosts, each with its own local storage device(s)
- Shared storage, that can be accessed by each host (or *node*), such as a file server
- Some method of interconnection that enables one node to see if the other is dead, and to help coordinate resource access

Interconnection topologies

- A *serial crossover cable* is the simpler (and more reliable) way to ensure proper intracluster communication
- An *Ethernet crossover cable* needs each host's TCP/IP stack to be functional to ensure proper intracluster communication
- A shared disk (in advanced setups), usually used for heartbeat only

Optional but strongly recommended

Gear to eliminate other single points of failure:

- Three Uninterruptible Power Supplies, one for each node and one for the shared storage
- Redundant network connections (using dual NICs and dual switches with bonding or trunking software on the server)

Some method of exclusive access to shared resources. This can be:

- Physical, in order to forcefully eject the other machine:
    - Two power switches, allowing each node to remotely cut the power to the other when it becomes stuck or inoperative
- Logical, using one of (as appropriate for each resource):
    - SCSI-3 persistent reservation , for denying access to shared storage
    - Controlling access to Fibre Channel or Ethernet network through manageable switches, and disabling the other node's port

Classification by role symmetry

There are two kinds of two-node clusters, from this perspective:

Active/Passive

One node owns the services, the other one remains inoperative.

Should the primary node fail, the secondary or backup node takes the resources and reactivates the services, while the ex-primary remains in turn inoperative.

This is a configuration where only one node is operative at any point of time.

Active/Active

There is no concept of a primary or backup node: both nodes provide some service, should one of these nodes fail, the other must also assume the failed node's services.

Classification by service aggregation level

Two kinds of computer cluster, again:

Service based

Every service is independent from each other, provided by the cluster: for example, a web server and a mail server are run on the cluster, and each one can be independently managed, switched from one node to the other, without affecting the functionality of other services.

One open source example of this kind of cluster is Kymberlite.

Logical-host based

In more complex configurations, there can be *dependencies* among services.

For example, a mail server receives e-mail for local users, thus storing the mail on a local storage resource, but requires the ability to read this email from a remote site. This then requires a *mail retrieving server*, such as an IMAP server.

Both of these services need access to the same storage resource, the first for writing the e-mail messages that arrived from the Internet, the second to read, move or delete them.

This means that the mail server cannot simply be failed over from one node to the other, as the mail retrieving server will still need access to the same data. These two services must be grouped together, forming a so-called *logical host*. To be more precise, this logical host will consist of three resources:

- the *storage resource*, needed by both server applications;
- the *mail transfer service* that receives e-mail from the Internet; and,
- the *mail retrieval service'* that acts as an interface, permitting users to view their email.

If it becomes necessary to fail over this logical host, the following steps need to be automatically or manually performed:

- stop the mail retrieving service on the failed node (if possible)
- stop the mail transfer service on the failed node (if possible)
- release the storage resource on the failed node (if possible)
- acquire the storage resource on the failover node
- start the mail transfer service on the failover node
- start the mail retrieving service on the failover node

One open source example of this kind of cluster is Linux-HA; one commercial example for systems running the Solaris Operating System is called Sun Cluster.