Open Single System Image (openSSI) Linux Cluster Project

Bruce J. Walker, Hewlett-Packard
Bruce.walker@hp.com

Abstract

The openSSI Cluster project is an ongoing open source project which was started two years ago to bring together some of the best Linux and Unix clustering technologies into a single integrated and yet modular project.

Linux is rich in cluster technology but is segmented into 6 different cluster areas - high performance, load-leveling, web-service, storage, database and high availability. The openSSI project address all cluster environments by simultaneously addressing the three key cluster goals - availability, scalability and manageability.

To accomplish this ambitious goal, the project was started with a Linux adaptation of the NonStop Clusters for Unixware code, contributed by Compaq/HP. That code included membership, internode communication, clusterwide process management, clusterwide devices, a cluster filesystem, clusterwide IPC (pipes, fifos, msgqueues, semaphores, etc.) and clusterwide tcp/ip networking. Other open source clustering code has been integrated into the modular architecture, including openGFS, openDLM, LVS, Lustre and a small component of Mosix. The architecture of the project allows for subsetting and substitution of components. A full function initial release is available in both source and RPM form. Many enhancement opportunities still exist both in integrating with other technologies and by improving scalability and availability.

Introduction

In this paper we explain Single System Image (SSI) clustering, we rationalize SSI clustering with other clustering paradigms and we explain how the open source Linux project (openSSI) can be the foundation for a wide range of cluster solutions. Technical detail on the openSSI architecture and components is provided, to the extent that space permits.

The term clustering, with no adjective specifying the type of clustering, means different things to different groups. Below is a short discussion of *high performance* (HP) clusters, *load-leveling* clusters, *web-service* clusters, *storage* clusters, *database* clusters and *high-availability* (HA) clusters. All of these have a few things in common that distinguish clusters from other computing platforms.

Clusters are typically constructed from standard computers, or nodes, without any shared physical memory. That, combined with the characteristic of running an OS kernel on each node, distinguishes a cluster from a NUMA or SMP platform (of course a cluster could be constructed from NUMA or SMP nodes). The "purpose" of a cluster is to work together to better provide HA, HP, parallel processing or parallel web servicing. How is a cluster different from a client-server distributed computing environment? Each of the cluster platform solutions is more peer-to-peer, typically with a sense of membership that each node has access to.

Solution Clusters

High performance (HP) clusters, typified by Beowulf clusters [Beowulf], are constructed to run parallel programs (weather simulations, data mining, etc.). While a master node typically drives them, the nodes work together and communicate to accomplish their task.

Load-leveling clusters, typified by Mosix [Mosix], are constructed to allow a user on one node to spread his workload transparently to other nodes in the cluster. This can be very useful for compute intensive, long running jobs that aren't massively parallel.

Web-service clusters, typified by the Linux Virtual Server (LVS) project [LVS] and Piranha do a different form of load leveling. Incoming web service requests are load-leveled between a set of standard servers. Arguably this is more of a farm than a cluster since the server nodes don't typically work together. Web-service clusters are include in this list because the service nodes will eventually work more closely together.

Storage clusters, typified by Sistina's GFS [GFS], the OpenGFS project [OpenGFS], and the Lustre [Lustre] project, consist of nodes which supply parallel, coherent and highly available access to filesystem data.

Database clusters, typified by Oracle Parallel Server (OPS; now Oracle 9I RAC)[Oracle], consist of nodes which supply parallel, coherent and HA access to a database. The database cluster is the most visible form of application specific clusters.

High Availability clusters, typified by ServiceGuard [ServiceGuard], Lifekeeper [Lifekeeper], FailSafe [FailSafe] and Heartbeat [HALinux], are also often known as failover clusters. Resources, most importantly applications and nodes, are monitored and scripts are run when a failure is detected. Scripts are used to fail over IP addresses, disks and filesystems as well as restarting applications.

It should be obvious that many of the above solution clusters could benefit by combining/leveraging other cluster technology. In fact, there is ongoing work in most of the cluster areas to add other cluster capabilities. After examining SSI clustering in general, we will look at how the Open SSI Cluster Project [OpenSSIC] is striving to bring all the best technology in each of these solution areas together on a SSI platform.

What is a SSI Cluster?

Presenting the collection of machines that make up the cluster as a single machine is what SSI clustering is all about. SSI Clustering is not new. The Locus Distributed System [Popek85] provided a SSI environment back in the 1980's. Sun Microsystems had an SSI project [Khalidi96] and Gregory Pfister dedicated a chapter of his book "In Search of Clusters" to Single System Image, claiming that general cluster adoption has been slow because of "lack of single system image software"[Pfister98]. The SSI presentation comes in several forms and to varying degrees. Below we look at 6 current "SSI" cluster approaches.

The database cluster is SSI with respect to the database. Instances of the database run on each of the nodes, accessing the same data and coherently presenting the same view of the data from each instance.

The storage cluster is SSI with respect to a set of devices and more importantly a set of filesystems. Even more SSI is accomplished if the root filesystem is shared between the nodes.

Web-service clustering (Linux Virtual Server (LVS) and other examples) presents the cluster to the outside world as a single name and single IP address, with different techniques to load level incoming connections and optionally failover the routing/redirection service. This is one form of SSI clustering.

Scyld is providing "SSI" for Beowulf clusters [Scyld] using the bproc process movement technology [bproc]. All processes are initiated on/from the master node and programmatically distributed to all the compute nodes via rexec, rfork or move. From the master node all these processes are still visible and signalable, just as if they were all running on a single machine. The cluster, defined as the parallel program running on it, is presented to the user as a single system. The filesystem is not SSI, the kernel IPC objects are not SSI, the device space is not SSI, independent processes on compute nodes would not see a SSI process space, etc. Nonetheless, given the specialized manner in which the cluster is used (typically one large parallel application is all that is running, with one process per compute node), this "master node" SSI is enough SSI to provide a significant improvement over pure Beowulf clusters.

Mosix provides "home node" SSI. Home node SSI is similar to Scyld's "master node" SSI except there can be lots of home nodes. A process is created on a node (called its home node). It might be transparently migrated to another node in the cluster but:

a) it still sees the environment of it's home node (processes, devices, filesystems, ipc, etc.)
b) other processes started on that home node see the process as if it was still running on the home node (for the most part, the kernel portion of the process is still running on the home node; system calls made by the process are shipped back to the home node for execution).

While the entire cluster is not presented to each process like one big machine, a given process does see the same single machine (the home node) view no matter where it is executing. Additionally, the user logged into the given home node sees all the processes created via that login session, no matter where they are running (and that user has full control of those processes).

The ideal SSI cluster is more complete in scope than any of the above SSI clusters for Linux - in terms of process management, in terms of SSI for other kernel resources and in terms of providing a high availability platform. The ideal SSI cluster unifies and extends the clustering described above to result in:

- a unified clusterwide process model with load balancing
- unified clusterwide device, storage and filesystem models with parallel and served access (thereby being the ideal platform for parallel databases)
- a unified clusterwide networking model, providing both a highly available single access point to a parallel server, and an SSI view from within the cluster;
- unified clusterwide inter-process communication (IPC) objects like pipes, fifos, semaphores, message queues, shared memory, sockets and signals, so programs, processes and users can work together no matter which node in the cluster they are executing on
- unified clusterwide system management, where single system tools can see and manage all resources of all nodes from any node
- simple, yet powerful high availability (HA) capability is provided for any resource from networking to inter-node interconnect to storage to filesystems to applications to processes to IPC objects

The ideal cluster, as described above, is the topic of the Open SSI Cluster project described in the next section.

Open SSI Cluster Project

This section has four parts - goals of the project, component technologies, cluster architecture and status.

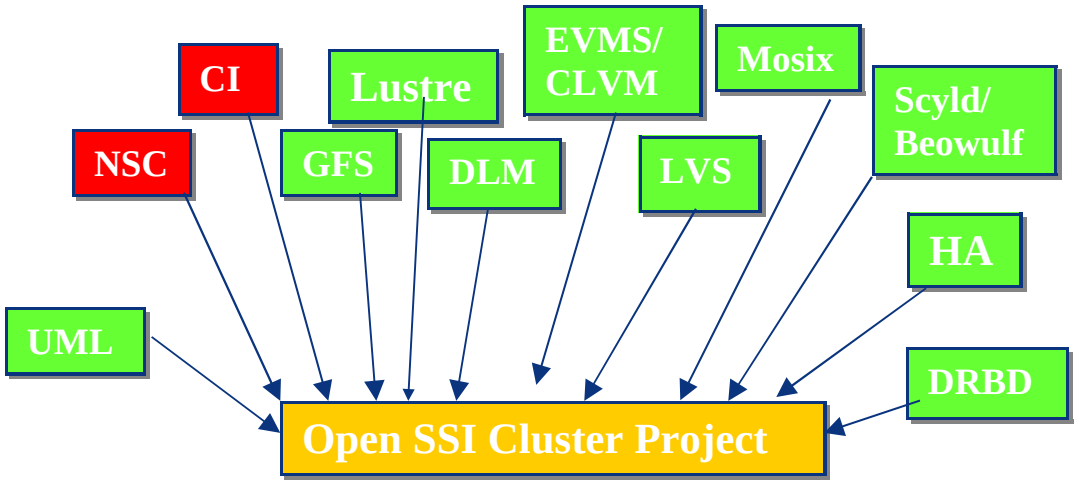
Project Goals

The ambitious goals of the project include:

- building a full SSI cluster using open source cluster components;
- extending the SSI cluster in areas of scalability and availability, again using open source cluster components;
- maintaining a modular architecture so components can be substituted as newer versions are produced.

Project Components

The figure below shows the many open source components currently envisioned as contributing to the Open SSI Cluster project. Other technologies will be added as needed.



Before presenting an overview of the project architecture and a review of the project status, a brief description of the contributed technologies is given.

NSC (NonStop Clusters for Unixware):
This was a product from Compaq and SCO/Caldera that provided a high degree of Single System Image on a Unixware base [NSC]. It is based on earlier work [Popek85,Walker98,Walker99]. Much of the code has been adapted to Linux as part of the Open SSI Cluster project, including cluster membership, inter-process communication, process management, process migration, cluster filesystem, clusterwide device access, and many different clusterwide IPC subsystems (including pipes, fifos, message queues, semaphores, shared memory and sockets). Application and node monitoring capability, along with application restart capability, have also been adapted to the Open SSI Cluster project.

CI (Cluster Infrastructure):
The cluster membership and inter-node communication subsystem pieces of NSC were made available as the CI project [CI], on which the SSI project is layered. Other cluster technologies may also be layered on CI.

GFS (Global File System):
GFS, developed by Sistina Software [GFS], is available from Sistina or through the OpenGFS project [OpenGFS]. It is being used in the Open SSI Cluster project as one option to provide a shared root or other filesystem for the cluster. It is not yet integrated with the DLM.

DLM (Distributed Lock Manager):
IBM open sourced a DLM code base. It is available under the GPL license [OpenDLM]. It has been integrated with the SSI membership subsystem of CI to allow dynamic addition and deletion of participating nodes.

LVS (Linux Virtual Server):
This open source technology [LVS] presents a single IP address and then load levels incoming connections to a set of backend Web servers. Several methods of forwarding the traffic are available and there are several different load-leveling algorithms to pick from. This technology works with the other parts of the SSI project.

Lustre:
The Lustre [Lustre] open source filesystem project is designed to scale to thousands of client nodes. Unlike GFS, Lustre defines a new protocol between the clients and the storage.

Mosix:
The Mosix project [Mosix] does transparent load leveling of unmodified processes. The Open SSI Cluster project has a different process management and process migration technology but is using the load leveling decision algorithms from the Mosix project.

Scyld/Beowulf:
The Beowulf parallel program [Beowulf] and communications libraries should be straightforward to port to the SSI cluster. The bproc library is also easily adapted to the APIs available in the SSI cluster. The concept of network boot and network install that Scyld utilizes has been developed for the SSI project as well. Thus only one node must do a full install and nodes can optionally netboot into the cluster.

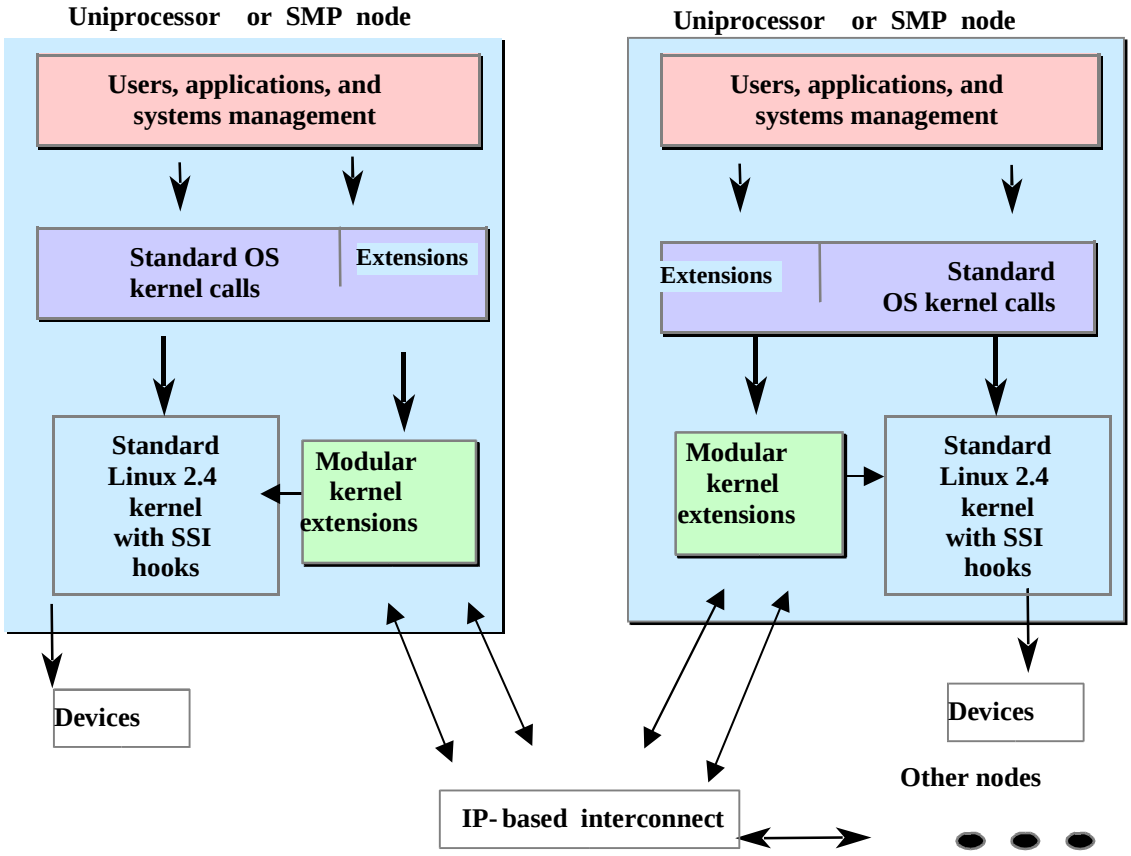
HA (High Availability):
There are many HA products and projects on Linux, from "heartbeat"[HALinux] to ServiceGuard [ServiceGuard], FailSafe [FailSafe] and Lifekeeper [LifeKeeper]. To date, all of these work on very loosely coupled clusters, without even a shared filesystem. Integrating one or more of these solutions on the Cluster Infrastructure (CI) platform or the Open SSI cluster platform will provide a simpler yet more powerful capability.

UML (User Mode Linux):
UML has been integrated with SSI so that you can set up a multi-node SSI cluster on a single machine. This may not have significant commercial value but makes a powerful development environment for cluster subsystems.

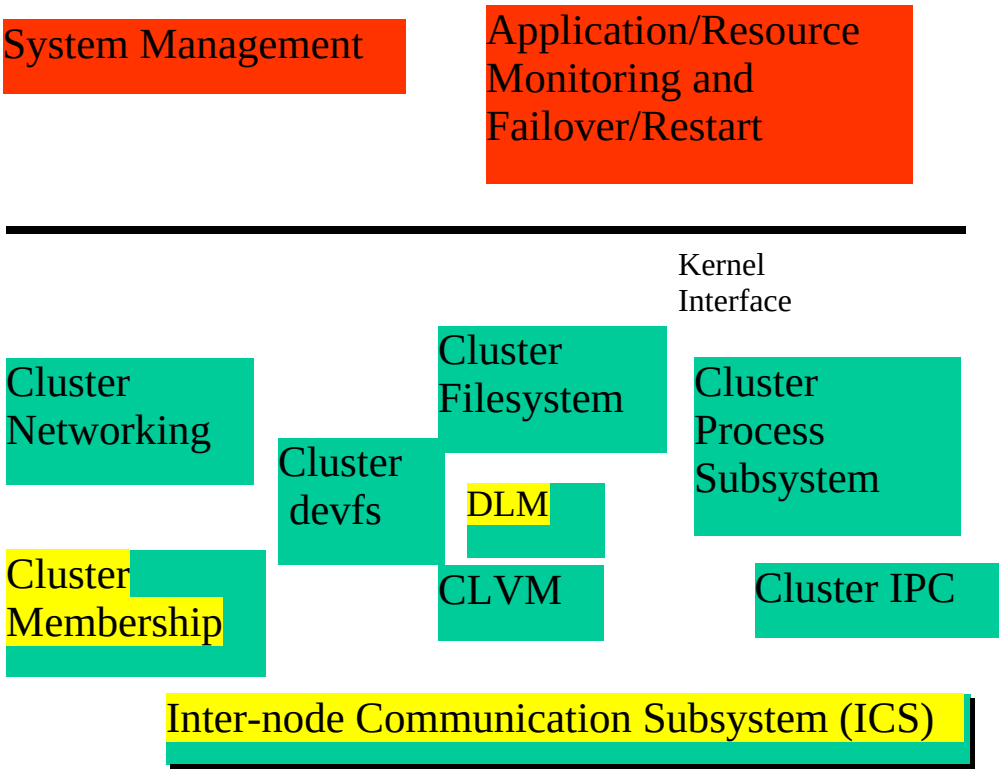
Project Architecture

The SSI cluster has a few characteristics that influence the architecture and distinguish it from other cluster solutions. Running on a single shared root filesystem, the SSI cluster is made up of an OS kernel on each node communicating and co-ordinating with each other. The OS kernel per node is for availability and performance. The shared root is for ease of management and it should contain few or no node-specific files. The shared root can either be managed by a parallel physical filesystem like GFS or can be served by one node at a time and shared via a layered cluster filesystem like CFS. In either case, nodes need to join the cluster early in the boot cycle and there should be a strong sense of membership - all nodes should know the membership and agree on it. Using one or more kernel-to-kernel communication paths, the kernels should present a highly available single machine at the system call level.

The co-operating kernels present a single namespace for nameable kernel objects and provide coherent distributed access to all objects. The diagram below portrays how unmodified applications interact with a standard kernel that has SSI hooks and SSI extensions. The extensions allow the kernels to co-ordinate access to objects like processes, files, message queues and devices. New interfaces are made available to allow cluster-aware applications to more fully take advantage of the cluster for added scalability, availability or manageability.



The SSI cluster enhancements generally fall into one of three categories - extensions outside the kernel for system management or high availability, kernel infrastructure pieces used to layer on added functions, and extensions to existing kernel subsystems to provide the SSI view of kernel objects. The figure below breaks the architecture components into those 3 categories. System management and application/resource monitoring and failover/restart are two subsystem areas outside the kernel. The infrastructure pieces in the kernel are cluster membership, inter-node communication and the distributed lock manager (DLM). Extended base subsystems include filesystems, process management, cluster devfs, a cluster volume manager, cluster tcp/ip networking, and clusterwide IPC objects.



Each of the subsystem areas is described below.

Cluster Membership:
The Cluster Membership code is adapted from the NSC code base and comes to the open SSI cluster project from the CI project. It has the algorithms to form the cluster, to monitor membership, to invoke registered nodeup and nodedown routines, to notify processes of membership events and support a set of membership APIs.

The cluster membership service (CLMS) is code that exists on all nodes. Each node is running in one of two modes - master or slave. At any time there is one master for the cluster. Takeover is done if the master fails. Via the master-run algorithms, nodes transition through a set of states which can include NeverUP, ComingUP, UP, Shutdown, Goingdown and Down. Each state transition is atomic and clusterwide so all nodes always have identical information. The CLMS master makes membership decisions, enforces any majority voting algorithm and does STONITH (Shoot The Other Node In The Head, used to ensure a node excluded from the cluster won't do any additional disk I/O) enforcement.

A largely independent node-monitoring subsystem informs the CLMS if a node is unresponsive. Nodes can also be managed through APIs. Most, if not all of the other kernel subsystems included in the presentation of SSI register nodedown routines. Each node runs these routines when a node fails, in order to cleanup data structures associated with the lost node. After all kernels have done their cleanup, the "init" process may run a script to do additional cleanup or failover. Only after that is the node allowed to rejoin the cluster.

CLMS works with and leverages the underlying Inter-node Communication Subsystem (ICS).

Inter-node Communication Subsystem (ICS):
ICS is the kernel-to-kernel communication path used by almost all the kernel SSI components. Like CLMS, it came from NSC and is part of the CI project. ICS is architected into two pieces, the transport independent piece (upper layer) and the transport dependent piece (lower layer).

The upper layer (transport independent) of ICS interfaces to the other kernel subsystems. It presents three programming paradigms - RPC, request/response or async RPC, and message passing. Message size is variable and there are data types to designate out-of-band data to be sent DMA if the underlying transport supports it (either push or pull). The upper layer piece also has a dynamic pool of threads to service incoming requests, which are prioritized and possibly queued to avoid resource deadlock and server process thrashing.

The transport dependent piece (lower layer) interfaces to a reliable message service over some hardware. The current implementation is running over tcp connections but other transports have been used and will be supported. The lower layer pieces also define how out-of-line data is transferred from node to node.

ICS works with CLMS to set up communication when a node joins the cluster and flush incoming and outgoing traffic on nodedown.

Distributed Lock Manager (DLM):
The DLM is a kernel component that comes from an IBM open source project. It runs on each node and is integrated with the Cluster Membership subsystem. It may in the future be integrated with the ICS subsystem. The DLM is used primarily to maintain various forms of cache coherency across the nodes in the cluster. One form of cache is the kernel filesystem cache. Both the OpenGFS and the Sistina GFS projects intend to use the DLM when it is integrated. The other key use of the DLM is for coherency between the caches of a cluster application. The Oracle Parallel Server (OPS) database was the first major application to use a DLM.

Clusterwide Filesystem:
The goal of the filesystem part of the project is to automatically, transparently and coherently present the same filesystem hierarchy on all nodes of the cluster at all times, in a highly available manner. This would include the root filesystem as well as access to filesystems of all types. There are three components to the architecture. First is support for various forms of cluster filesystem. Second is a mechanism to ensure that each node has the same mount hierarchy. Third is ensuring that open files are retained on process movement.

There are three approaches to cluster filesystems and the project supports all three. One approach can be characterized as a parallel physical filesystem. An instance of the filesystem is run on each node. Each instance goes directly to the storage and has it's own log area. The instances co-ordinate their caches through a lock manager subsystem (ideally the DLM). Sistina GFS and OpenGFS are examples of this type of cluster filesystem. Another approach is that a client instance is run on every node and every client learns the data layout from a metadata server and goes directly to each of many dedicated storage nodes, over an interconnect. Lustre [Lustre] uses this parallel network approach. The final approach also has a client instance on each node. The clients communicate over the interconnect to a server node for each filesystem (much less parallelism than the parallel network approach). Each server node is part of the cluster and supplies one or more filesystems to the cluster. This approach is attractive for small clusters or clusters without shared media or as a way to clusterize different physical filesystems like JFS, XFS, cdfs, ext3, etc. A key component of all solutions is the ability to transparently continue operation in the face of failures.

Clusterwide Process Subsystem:
Two goals of the process management system are to make all processes on all nodes transparently visible and accessible from all processes and to allow process families and process groups to be transparently distributable. Other goals include single site semantics in the face of arbitrary node failures, a clusterwide /proc, the ability to migrate a process from one node to another while it is running and a flexible load leveling subsystem. To accomplish these ambitious goals:

- process are assigned clusterwide unique process ids (pids), which they retain if they migrate to other nodes;

- a "vproc" structure (somewhat like a vnode structure) is added and process relationship fields are moved from the task struct to the vproc struct. A given process has a single task struct and it is on the node where the process is executing. There may be several vproc structs for the process around the cluster as needed to track and represent various process relationships.
- System calls are executed on the node where the process is currently running; if remote resources are needed to complete the system call, object specific requests are sent over the cluster interconnect via ICS.
- Enough information is kept redundantly in the cluster so that relationships that survive arbitrary node failures are successfully reconstructed
- Rexec(), rfork(), migrate() and kill3() calls are added so cluster-aware applications can manage process placement. A sigmigrate signal is added to facilitate the movement of unmodified processes and process groups from node to node.
- An optional process load balancing subsystem is configurable to transparently move processes around the cluster to better utilize the processing power of each node.

Clusterwide Devices Subsystem:
There are several goals with respect to device management in the SSI cluster. First, all devices on all nodes should be nameable, preferably in a node-independent manner, and transparent access to all devices from all nodes should be available. Devices with shared physical access should have a single name. Device major/minor numbers seen via the "stat" call must be clusterwide unique and both names and major/minors must be persistent across boots and independent of the order nodes boot in.

Accomplishing these goals is done via extensions to the kernel devfs and without changes to device drivers. First, each device is transparently assigned a clusterwide-unique major/minor in addition to its node-specific major/minor that the driver may want. The clusterwide value is exposed in the "stat" interface, where clusterwide uniqueness is more important than matching the value the driver is used to. Next, the devfs code on each node has the complete device information for the cluster so lookups and opens happen directly from the process execution node to the node where the device is attached. Next, there is a set of file operations that allow an open on one node to access a device on another node.

The /dev directory is changed to be a context dependent symlink where the context is either default or a compatibility context. The default is a single dev directory with the union of all the devices in the cluster and subdirectories for the devices on each node. The compatibility context would just give the application the devices of the node it is executing on

Cluster Logical Volume Management (CLVM):
The logical volume manager [LVM] abstracts the physical boundaries of disks to present logical entities. Co-ordination is needed if multiple nodes are going to access and possibly change the structure of logical volumes at the same time. The architectural plan is to leverage the existing base LVM technology and support any CLVM project (e.g. EVMS).

Clusterwide Networking Subsystem:
From a performance perspective, it is important that each node do as much network processing as is feasible without interacting with other cluster nodes. Some interacting is needed to attain an SSI view. An important goal of the open SSI cluster project is that the cluster looks, from a networking standpoint, to the outside world, like a single, scalable and highly available machine. The Linux Virtual Server (LVS) project, with failover, provides much of the needed capability. A single address is advertised but connections are load leveled to designated service nodes through a variety of algorithms. The SSI project may augment LVS by making configurations more automatic and by adding additional availability features. What LVS does not address is the view of the cluster from within the cluster and the view of the external world from within the cluster. To be strictly SSI, all network devices and thus IP addresses in the cluster should look local to all nodes (LVS only make the Cluster IP look local to all nodes) so a wildcard listen on any node would effectively be listening on all the devices on all nodes. Also, strict SSI would allow two processes to communicate over the loopback devices, even if they were executing on different cluster nodes. Finally, the routing tables on each node should be co-ordinated so that if one can reach a foreign host from one node, you can automatically do the same from any node in the cluster.

For the open SSI cluster project, the first step is to have at least basic LVS capability with failover. Providing more strict SSI capability will be addressed as needed.

Clusterwide Inter-Process Communication (IPC):
There are many forms of IPC in Linux, ranging from signals, pipes, fifos, semaphores, message queues, shared memory, internet sockets to unix-domain sockets. Signals are discussed under process management. Some of the other IPC objects are nameable and some are not. The architecture of the open SSI cluster project is straightforward. There is a clusterwide namespace for all nameable objects. Objects are always created locally, for performance. Process movement does not loose access to named or unnamed objects. Objects should be able to move from node to node. The namespace must adhere to standard practice for naming each object in Linux, it must be efficient on lookup and create, and it must be resilient to arbitrary node failures. To accomplish all these goals, the IPC nameservers are:

- centralized so a lookup or create is no more than an single RPC (no polling)
- rebuildable from remaining cluster nodes.

Nodes in the cluster may cache information about where a particular IPC object is currently managed, with the understanding that the nameserver can be consulted if the cache turns out to be stale.

Clusterwide System Management:
The goals with respect to system management are to use slightly enhanced single machine management programs rather than a whole new set of cluster management tools. Such an approach should greatly reduce the management burden clusters often impose. This approach was the primary approach for NonStop Clusters for Unixware (NSC). Enhancements to the single machine tools include dealing with nodes that are currently unavailable and displaying which node a resource is affiliated with.

System management also includes installation and booting. The installation architecture is to do a fairly standard install on a single node, including putting the root filesystem on a shared media, and then trivially adding nodes both at initial install time and anytime thereafter. Booting should be as SSI as feasible, taking into account the availability requirements that nodes can join an existing cluster at any time. The architecture involves having a single "init" process that, on cold boot, takes the initial set of nodes to the specified run level in parallel. Nodes that join later are not generally visible until they reach the run level of the rest of the cluster. To manage cluster booting in this manner, some rc scripts must be modified, as well as init itself. A graceful shutdown of the entire cluster as well as individual nodes is also architected through init.

Application/Resource Monitoring and Failover/Restart:
This component is key to satisfying our high availability goal. Part of the functionality standard HA products and projects provide is satisfied by the infrastructure and core OS components and part is layered over the SSI cluster platform. Nodeup and nodedown detection is accomplished through the cluster membership subsystem with the sigcluster signal and membership APIs available to applications and monitoring software. Simple filesystem availability is accomplished by the filesystem (GFS or CFS). Simple network IP address availability is accomplished via cluster network failover. Advanced monitoring capability should be integrated with the appropriate subsystem. Redundant interconnects is part of the ICS subsystem. Split brain detection and avoidance and STONITH capability are part of the cluster membership subsystem.

The application monitoring and restart architecture has two conflicting goals to deal with - compatibility with a solution on a non-SSI loosely coupled cluster and simplicity of configuration and management possible due to the underlying SSI platform. It should be the case that one can adapt any of the existing launching/monitoring/restarting capabilities to the SSI platform. Reliance on independent namespaces (files, IPC objects) can complicate the adaptation.

Project Status

There have been several source releases from the Open SSI Cluster project since it debuted in June of 2001. Until recently, the releases were development releases. However, starting with the 0.7.5 release (Oct 2002 for source, November for binary), the system is complete enough for use in a variety of environments. Version 0.7.5 is based on the 2.4.18 kernel. It can work in any one of several distributes and supports IA-32, Itanium-2 (IA-64) and Alpha processor types. The binary release is based on RH7.3.

On a subsystem by subsystem basis, the status is:

Cluster Membership (CLMS):
CLMS is quite stable and quite functional at this point. It should work up to 50 nodes but some work on initial booting is needed to scale to large numbers of nodes. Membership policy isn't very sophisticated yet and integration with STONITH algorithms is yet to be done. Nodedown detection frequency can be tuned so sub-second detection is feasible.

Inter-node Communication Subsystem (ICS):
ICS is quite functional over tcp. Outstanding enhancements would be to put it over native Myrinet or Infiniband. Some forms of redundant NICs are transparently supported under ICS. Other forms require some work in ICS to route around failed NICs or failed subnets.

Distributed Lock Manager (DLM):
The open DLM project has much of the DLM working on Linux and integration with CLMS has been done.

Clusterwide Filesystems:
The project currently supports all three types of cluster filesystem – parallel physical via openGFS, stacked neetwork via CFS and parallel network via Lustre. Both CFS and openGFS have been used for the root filesystem. Failover capability for CFS is being actively worked on and is expected in January 2003. An early version of the Lustre client was tried on the system. Enforcing and automating a clusterwide mount model is largely complete. Inheriting open files as processes move is provided.

Clusterwide Process Subsystem:
Most of the process management code is up and running. Process relationships can be distributed; access to any process from any process clusterwide is supported; processes can rfork, rexec, migrate or be signaled to migrate with kill3 and SIGMIGRATE; nodedown handling cleans up data structures and maintains remaining relationships. Currently clones must be co-located (no cross-node shared address space capability). Clusterwide /proc for processes is in, as is transparent clusterwide ptrace. The load-leveling algorithm adapted from the Mosix project is included in recent releases.

Clusterwide Devices Subsystem:
Complete naming and access to remote devices is supported. Each node uses devfs to name its devices and CFS provides the clusterwide naming. Rather than a single unified /dev, /dev is currently a context symlink to node specific device directories.

Cluster Logical Volume Management (CLVM):
Work is ongoing at Sistina on a CLVM but no work to integrate it with SSI has started. The EVMS subsystem is now released for single machines and work to clusterize it should be starting soon.

Clusterwide Networking Subsystem:
The LVS code has been integrated with SSI. Some enhancements to provide more SSI have been coded but the exact nature of the complete offering is still under discussion. Failover capability using keepalived is provided.

Clusterwide Inter-Process Communication (IPC):
Signals are clusterwide. Pipes and fifos are clusterwide, as are message queues. Internet sockets are clusterwide. Semaphores, shared memory and unix-domain sockets will be supported by the end of 2002.

Clusterwide System Management:
Installation of the first node is fairly easy is you use the CFS option to distribute the root. Installation of the first node is still very cumbersome if you choose GFS because GFS is not in the base, so getting a GFS root filesystem is at least a 2 or 3 step process. Installation of other nodes is trivial due to netboot and PXE boot. After installation, nodes can either have a ramdisk and kernel on their local disks or can netboot into the cluster.

Init has been enhanced to support a cluster of nodes and to be statefully restartable. There is no run level support yet and thus only a crude method of getting all nodes to a reasonable run level exists.

Little effort has been spent yet to enhance the other sysadmin tools to make them cluster-aware and present a SSI environment (other than selected filesystem commands).

Application/Resource Monitoring and Failover/Restart:
To date, there has been little integration with the Linux HA projects. As explained in the architecture section, some areas will be dealt with via the subsystem (ICS, GFS and LVS+ will do interconnect, filesystem and IP takeover so the HA subsystem doesn't have to). An application monitoring and restart capability was ported from NSC and is part of the current release.

UML integration :
User-Mode-Linux (UML) is a supported architecture for SSI kernels so one can create a multi-node SSI cluster on a single physical machine in order to do development and testing.

Project Enhancement Opportunities

While the current code base for the openSSI project has a rich set of functionality, there are always many opportunities to enhance it either by integrating other functional components or by extending the current functions. Components which would be valuable to integrate include DRBD (Distributed Replicated Block Device [DRBD]), EVMS (Enterprise Volume Management System [EVMS]), OCFS (Oracle Cluster File System [OCFS]), and Failsafe [Failsafe]. Also, integrating MPICH and the Bproc libraries would enhance the High Performance capability.

Possible enhancements for availability would include dual interconnects, process checkpointing, fault tolerant processes and IPC object movement. Scalability enhancements include support for other interconnects (Quadrics, Myrinet, Infiniband) and some algorithm changes to scale from hundreds to thousands of nodes. Manageability enhancements include streamlining installation and node monitoring capabilities.

Summary

To date, clustering on Linux has been quite fragmented, as solutions to specific problems have been developed and deployed, often concentrating on only one of the three key cluster goals - availability, scalability and manageability. Now, each solution is looking for the added functionality that the other solutions have. This could eventually result in some consolidation. The SSI platform, as pursued by the Open SSI Cluster project, is attempting to not only bring the best of each cluster solution area together in a single offering, but also is doing so in a modular way that should allow component subsetting and substitution and expansion over time.

The term SSI cluster is not an absolute term. There are variants like "master node" SSI and "home node" SSI that provide a single machine view to a set of processes spread around the cluster. There are also degrees on SSI - one can do SSI system management without SSI process management, or SSI file management without a SSI device space. The Open SSI Cluster project is the most ambitious attempt to provide the greatest degree of SSI clustering.

Having just SSI is not enough, however, to make a cluster solution valuable to a broad segment of the computer world. Added capability in the areas of scalability and availability are critical to a general cluster strategy. Even though SSI provides a sharable environment where processes on different nodes can work together and scale, load balancing both processes and incoming work requests enhances the SSI cluster with respect to scaling. Integrating parallel programming libraries is also important to scaling. A cluster that acted like a single machine and thus completely failed if any node failed would look like an SMP or NUMA system from an HA perspective and would miss the opportunity presented by having a copy of the OS kernel on each node. Instead, each node should be an independent fault zone and all the

monitoring, failover and restart capability of HA/failover clusters should be put on the SSI cluster offering. The SSI cluster presents a much simpler environment for providing HA because the system can manage networking, device and filesystem monitoring and failover. Application HA is much simpler since there is a common root filesystem to hold a single copy of the process availability configuration and because any program can run on any node at any time.

While the Open SSI Cluster project is quite young and ambitious, it is well along the way to providing a full SSI environment. Part of the reason is that it was seeded with a set of SSI code from Compaq's NonStop Cluster for Unixware product. The other reason is that completed code from several other open source projects, including GFS, Mosix, LVS, Lustre, DLM and Beowulf is already integrated or is ongoing.

References

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

[bproc] http://bproc.sourceforge.net

[CI] http://sourceforge.net/projects/ci-linux

[DRBD] http://drbd.cubit.at

[EVMS] http://sourceforge.net/projects/evms

[FailSafe] http://oss.sgi.com/projects/failsafe

[GFS] http://www.sistina.com/products_gfs.htm

[HALinux] http://linux-ha.org

[Khalidi96] J. Kahlidi, J. Bernabeu, V. Matena, K. Sherriff, M. Thadani, "Solaris MC: A Multi Computer OS", USENIX 1996 Annual Technical Conference, pp191-203

[LifeKeeper] http://www.steeleye.com

[Lustre] http://www.lustre.org

[LVS] http://www.LinuxVirtualServer.org

[LVM] http://www.sistina.com/products_lvm.htm

[Mosix] http://www.openmosix.org

[NSC] http://docsrv.caldera.com/TopicViews/nsc_intro.html

[OCFS] http://otn.oracle.com/tech/linux/content.html

[OpenDLM] http://sourceforge.net/projects/opendlm

[OpenGFS] http://www.opengfs.org

[OpenSSIC] http://sourceforge.net/projects/ssic-linux

[Pfister98] G. Pfister, "In Search of Clusters", 2nd edition, Prentice Hall PTR, 1998

[Popek85] G. Popek, B. Walker, "The LOCUS Distributed System Architecture", MIT Press, 1985

[ServiceGuard] http://www.hp.com/products1/unix/highavailability/ar/mcserviceguard/

[Scyld] http://www.scyld.com

[UML] http://user-mode-linux.sourceforge.net/

[Walker98] B. Walker, "Clusters: Beyond Failover", Performance Computing, August 1998

[Walker99] B. Walker, D Steel, "Implementing a Full Single System Image UnixWare Cluster: Middleware vs Underware", PDPTA99