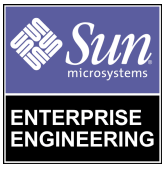
Administering Sun™ Cluster 2.2 Environments

By David Deeths - Enterprise Engineering

Sun BluePrints™ OnLine - October 2000



http://www.sun.com/blueprints

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 806-6856-10
Revision 01, October 2000

Administering Sun™ Cluster 2.2 Environments

Introduction

The Sun™ Cluster 2.2 software provides a solution for creating a cluster of nodes that provide a higher level of availability for the applications running on the cluster than could be provided by the individual nodes.

The primary goal of Sun Cluster software is to provide an increased level of availability. Once installed, it is critical that the cluster administrator maintain this level of availability. Availability permeates every aspect of cluster administration, from set-up and configuration to monitoring and maintenance. It is critical to administer for availability at every stage of the cluster lifecycle.

This article is not intended to provide all of the technical details of cluster administration; rather, it is an introduction of the basic structure of Sun Cluster 2.2 administration and informs the reader of issues that need to be considered in setting up and administering a Sun Cluster environment. In addition, this article provides tips for effective cluster administration to increase the level of availability provided by the cluster and also to minimize the risks of administrative errors. For a more exhaustive view of the subject, refer to the *Sun™ Cluster 2.2 Administration* Guide and the upcoming Sun BluePrints book titled *The Sun™ Cluster 2.2 Environment: Fundamentals and Beyond*.

Cluster Administration Fundamentals

The Sun Cluster 2.2 software operates on two to four nodes which are connected by a private network. The first node to successfully start the cluster becomes a cluster member. Subsequent nodes become members once they start the cluster software and the current cluster members determine that they are legal members. Node status is determined through heartbeats over the private interconnect and application status is determined through application specific fault monitors. An application is made cluster aware through the use of methods which allow it to start, stop, and be monitored for faults. A cluster aware application is called a highly available data service. When the cluster discovers a failure in a data service or the node it is hosted on, the cluster performs a failover, which results in the data service restarting on a new node. This provides the primary availability advantage of a cluster, since the failover is generally much faster than the time it would take to diagnose the problem and reboot a nonclustered node. Failover is accomplished through the use of logical hosts.

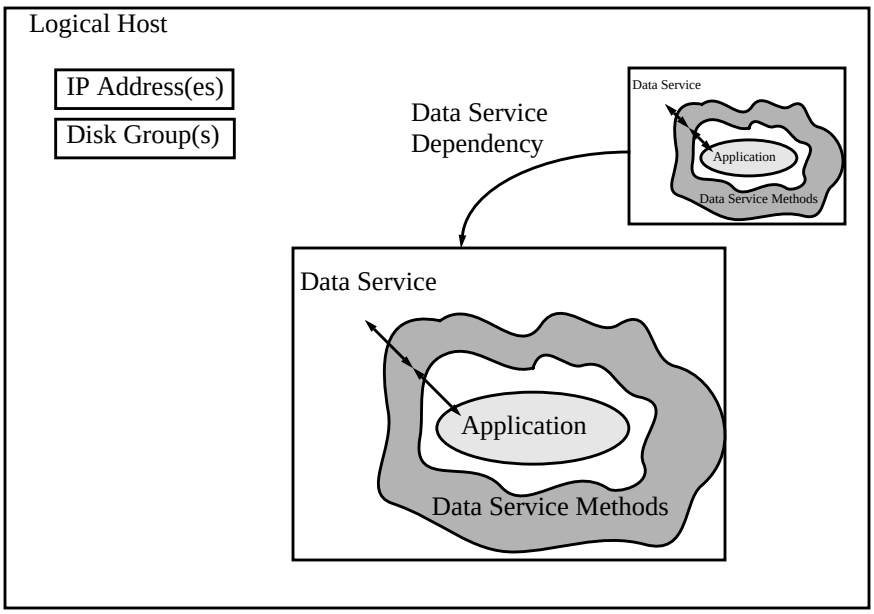
Logical hosts are the basis of highly available (HA) services, since they provide the basis for data services to fail over from one node to another. A logical host basically acts as a container for HA data services, abstracting the details of failover away from them. In other words, the logical host provides the same environment to the data services which are registered with it, regardless of which node it is hosted on. Data services access shared IP addresses and data through the logical host framework. If a failover occurs, the cluster framework will briefly shutdown the data services in a given logical host, then restart when the logical host reestablishes the shared IP addresses and data disks on the new node.

A logical host thus provides shared IP addresses and data disks to the HA services registered with it. The shared IP address is a shared interface accessible to the outside world (configured through the Public Network Management (PNM) system), while the shared data disks are a shared interface to the data (configured through the volume manager). The failover mechanisms of the logical host are part of the cluster framework.

An HA data service is simply an application which uses the proper API to interact with the logical host framework and failover via the logical host. Basically, an HA data service is an ordinary application, which can be controlled by the cluster framework through an application wrapper. This wrapper consists of written methods which allow the application to be started, stopped, and monitored for faults. The data service will run normally, unless a fault is detected in the host service or the cluster node it is hosted on. In this event, the service will failover to an alternate node via the logical host framework. Because the logical host controls both the data service's interface to the outside world (the logical host IP address), and the service's interface to the data (the shared data disks under the logical host's control); when the data service restarts on the new node, it has the same environment. In

addition, outside hosts access the data service in the same way, making the failover transparent (except for lost connections and the delay). The relationship between logical hosts, data services, and applications is shown in FIGURE 1.

FIGURE 1 Relationship Between Applications, Data Services, and Logical Hosts



Multiple data services can coexist within the same logical host. A simple dependency model can control the order in which they are started or stopped. While a single data service can be configured into multiple logical hosts, each logical host would have its own instance of the data service, meaning the data service instances can operate independently. However, when the data service is started, stopped, registered, or unregistered, all instances are affected.

In order to limit the availability cost of a node failover due to a bad network interface, the cluster software also provides for Network Adapter Failover (NAFO), which increases availability by allowing network adapters to be put in groups where one network adapter can act as a backup for another. In the case of a NAFO group failover, the redundant adapter can take over the IP address of the failing adapter. Since the data service can remain running on the same node, this is usually considerably faster than a node failover, providing an even lower downtime in the case of a failure.

The cluster configuration and state are maintained by several cluster configuration databases. Referring to the cluster databases can be a bit confusing when you use their proper names. This is because the proper names for two different cluster

configuration databases are the “Cluster Configuration Database” (CCD) and the “Cluster Database” (CDB). Because of this, for the remainder of the article, the CCD and the CDB will only be referred to by their acronyms. The CCD is divided into two pieces, the init CCD and the dynamic CCD. To match with the cluster documentation and because the word “dynamic” can be confusing in context, the term “CCD” will implicitly specify the dynamic CCD. The init CCD will always be referred to specifically. Note that for the purposes of this article, the terms “cluster configuration databases” (note the plural) and “cluster configuration database files” refer to all of the cluster configuration databases, not just **the** “Cluster Configuration Database” (CCD). This includes the init CCD, the dynamic CCD, the CDB, and the `pnmconfig` file.

The cluster configuration databases are the pieces of the cluster framework that are modified during cluster change. It is critical that all cluster nodes use the same configuration. Although the cluster has safeguards built in to prevent different cluster nodes from using inconsistent cluster databases, there are situations where the administrator could accidentally bypass the safeguards. Because of this, and the fact that it is best to prevent the problems before the cluster safeguards are needed, this article outlines a number of procedures for reducing the chances of inconsistent cluster configuration databases.

The Cluster Lifecycle

The cluster lifecycle consists of three stages: installing and configuring, steady state, and decommissioning or migrating. FIGURE 2 illustrates an overview of the cluster lifecycle from beginning to end. The first stage involves setting up the cluster environment which includes installing the hardware and operating system, followed by the necessary software (the cluster and volume manager software). Once this is accomplished, the software can be configured.

The PNM and volume manager configuration can occur before or after the cluster software configuration, but for reasons of practicality, it is usually best to configure the volume manager first. Once the cluster, volume manager, and PNM infrastructures are in place, logical hosts and data services are set up. When this setup is complete, the cluster is fully operational.

The cluster then enters the second stage of the lifecycle, the steady state. In this stage, the cluster requires little intervention unless a failure occurs or the administrator needs to change the configuration. Some of the possible failures are listed in FIGURE 2. If there are no failures or changes in configurations, the cluster runs until the administrator decides to decommission or migrate the services hosted on it to another server or cluster. This enters the cluster into the final stage of the lifecycle.

An overview of the cluster lifecycle is described below to give an understanding of the procedures required at each stage. The “Configuring for Availability” section describes the necessary decisions and procedures of cluster installation in more detail.

1st Stage: Installing and Configuring

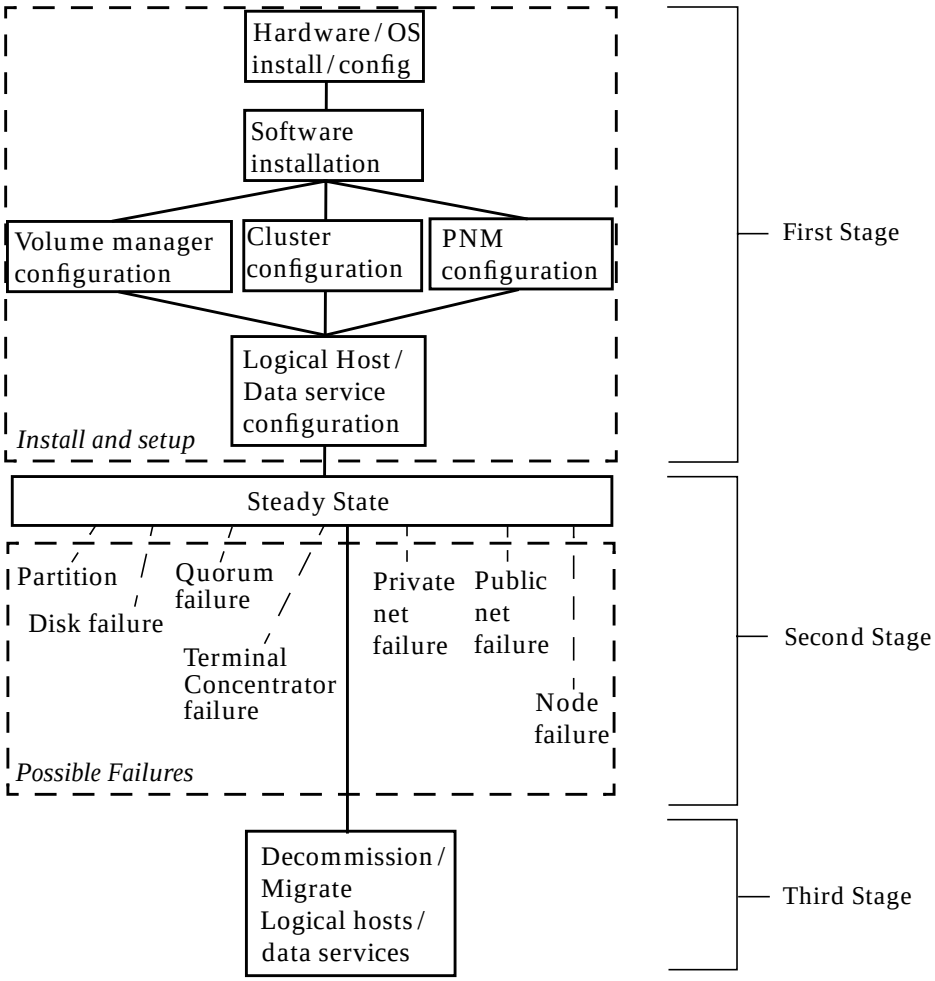
Careful planning for installing and configuring a cluster can avoid unnecessary difficulty and increase system availability. It is important to plan ahead for the installation and configuration of each component and service of the cluster. Managing and maintaining the cluster is covered in the “Maintaining the Cluster” section.

The goal of an HA cluster implementation is to create an HA service. The cluster installation builds up to this, however, several other components must be configured before an HA service is viable.

As FIGURE 2 shows, the first step in implementing a cluster is the hardware and OS installation. Afterwards, it is necessary to install and configure Public Network Management (PNM), the volume manager, and the cluster. These components are independent, so the order in which they are configured is not relevant. However, all of these components must be configured correctly in order to proceed with logical host configuration.

When installing a cluster, take special consideration in making components and services dependent on other services. In particular, data services and logical hosts should be as independent as possible in order to prevent one service or logical host from affecting others. Redundant components should be as independent as possible in order to limit single points of failure. In addition, the cluster should be as independent as possible from outside services. Creating a dependence on an outside service limits the availability of the cluster to the availability of the outside service. These issues are described in more detail in the “Configuring for Availability” section.

FIGURE 2 Cluster Lifecycle.



Once the volume manager, cluster software, and PNM components are installed, you can begin the logical host and data service setup. Note that a cluster does not necessarily require logical hosts, for instance a parallel database such as Oracle Parallel Server can use the cluster infrastructure without being part of a logical host. This section only describes setting up logical hosts, but the remainder of the article is mostly applicable to configurations using either logical hosts or parallel databases.

Logical hosts can be created by using the `scconf -L` command. This command takes the following as arguments: the disk group name, the network interfaces to use, the IP address, the logical host name, and the nodes that can master the host. In order for a logical host to function properly, it needs an administrative filesystem,

which the cluster can use to store configuration information specific to the logical host. Creating an administrative filesystem is accomplished by using the `scconf -F` command in configurations using Veritas Volume Manager (VxVM), or manually for configurations using Solstice DiskSuite™. Instructions for creating an administrative filesystem can be found in the *Sun™ Cluster 2.2 Administration Guide*. Once the logical host exists, the administrator is able to add data services to it.

There are several steps in configuring a data service. In order to be recognized by the cluster framework and to be attached to one or more logical hosts, you must register the data service with the cluster. This is accomplished by using the `hareg -r` command. The data service can be turned on using the `hareg -y` command. Splitting this procedure into a two part process allows the administrator to turn off the service without having to reconfigure it to turn it back on. If the service needs to be reconfigured, it will have to be unregistered and then reregistered. Besides the configuration that is part of the `hareg -r` command, some data services may require some additional configuration (as described in the appropriate manuals).

Multiple data services may be configured into a logical host using this process. Multiple logical hosts may be set up on each node as well.

2nd Stage: Steady State

Once the logical host and data service configuration is complete, the cluster is fully operational. At this point, the cluster configuration enters steady state, which is only interrupted by configuration changes or component failures. Even when no failures or configuration changes occur, it is important to monitor the health of cluster components. If failures aren't detected and dealt with promptly, the long-term availability of the cluster will decline. It is critical to find and fix problems quickly with both active and latent components. People, process, and product are all essential elements of a long term availability strategy. Ideally, failover drills should be run through regularly to assure that the configuration is capable of failing over and that the staff is able to respond properly. The process of cluster maintenance is covered in the "Maintaining the Cluster" section.

When failures or configuration changes occur, it is important to consider the impacts they will have on the cluster. As shown in FIGURE 2, a number of different types of failures are possible. Administrators need to be trained to deal with each type. Although general concerns relating to cluster changes are discussed in the "Managing Cluster Change" section, specific procedures for dealing with failure scenarios and cluster change are beyond the scope of this article. When the cluster is modified or maintained, you must perform steps to assure the changes are made properly. This requires proper change control, but also includes updating and verifying the cluster configuration databases to account for the new configuration. This is explored in the "Managing Cluster Change" section.

3rd Stage: Decommissioning or Migrating

The final stage of the cluster lifecycle occurs when some or all of the services hosted by the cluster are no longer needed or need to be relocated. If services need to be decommissioned or migrated to new machines, the process of adding a data service can be reversed.

Data services first need to be turned off using the `hareg -n` command. They can then be unregistered using the `hareg -u` command. Once there are no data services configured in a logical host, it can be removed by using the `scconf -L -r` command.

Migration of a logical host is a relatively painless process because the logical host has already enforced abstraction. Migrating a logical host to a new machine simply requires reconfiguring the IP addresses, shared disk groups, and the HA data service applications. Because the data services were abstracted by the logical host, they can normally keep the same configuration as before.

Configuring for Availability

During the installation portion of the cluster lifecycle, it is important to keep in mind the ultimate goal of configuring for availability. Some of the key points to consider when setting up a cluster are highlighted below.

Independence of Data Services

There are several important considerations that come into play when the data service is configured into multiple logical hosts. Removing or performing major changes (including adding disk groups or IP addresses) to a logical host requires unconfiguring all data services registered into it. There is no way to unconfigure or stop only a single instance of a data service, which can cause some unexpected dependencies between logical hosts. For instance, consider a scenario in which logical host A has data services for HA-NFS and logical host B has data services for HA-NFS and a custom service. If a major change (such as adding or removing disk groups or IP addresses) needs to be made to logical host A, HA-NFS will need to be turned off and unconfigured, which will turn off and unconfigure the HA-NFS instance in logical host B as well. The HA-NFS service on logical host B becomes unavailable, which could impact the custom service also if it is dependant on the HA-NFS instance. As this example demonstrates, it is crucial to consider the ramifications of having a single data service on multiple logical hosts, particularly if there are data service dependencies involved. In general, it is best to avoid having

the same data service run on more than one logical host. While architectural constraints may require breaking this rule, following the rule as closely as possible promotes independence between logical hosts. The end result and ultimate goal is increased availability.

Another important consideration for data service configuration is that it is usually best to limit the number of data services in a logical host. Following this rule prevents instances of "upgrade interference," where upgrading one application may be difficult or impossible due to support issues related to version numbers of the other applications on the cluster. For instance, upgrading your Web server may only be supported with an upgraded OS. However, the upgraded OS may not support the database you use. While such situations are rare with single applications, the risk of having support issues increases drastically with each increase in the number of applications. In addition, it is important to keep in mind that support for applications is very conservative in a clustered environment, owing to the fact that proper quality assurance is crucial in an HA environment. This conservativeness is a good reason to limit the number of applications on a cluster and a good reason to keep logical hosts abstracted. If the logical hosts are abstracted properly, it will eliminate dependencies which could cause difficulty when migrating logical hosts to a new server or cluster. The ability to migrate logical hosts can be important in situations where no upgrade path allows the simultaneous support of all component data services on the same cluster.

Independence of Redundant Components

It is highly recommended to make redundant components as independent from each other as possible. This applies to interfaces in the same Network Adaptor Failover (NAFO) group, the private interconnects, and the SCSI or Fibre Channel controllers. Although redundant components are, by definition, intended to reduce single points of failure; if they are on the same card or the same bus, they provide lower availability than components which are on independent cards or busses, since a failure in the common card or bus will affect both components.

It is nearly impossible to avoid single points of failure at the node level entirely, but reducing their impact as much as possible assures a high level of availability. At the very least, components of the same redundant system should reside on different interface cards. It is even better if components of a redundant system are spread across different I/O busses.

Clearly, assuring that components are on different interface cards is relatively trivial, except in cases where there are not enough I/O slots available. It is, however, a bit more difficult to determine on which bus a given controller resides. The bus for a device instance can be determined using the device's logical path. The logical path is a symbolic link to the device's physical path, which represents its actual position in the hardware hierarchy. Using the `ls -l` command on the logical path will show the symbolic link, which will be in the `/devices` directory.

Devices with a common parent in the device tree will always share a common hardware element. However, because some I/O buses or subsystems may share dependencies with others, some devices that do not share a common device tree parent may still share a common hardware element. In addition, even unrelated hardware systems can have an unexpected single point of failure. For instance, two different busses may be cooled by the same fan.

Understanding hardware dependencies requires a detailed understanding of the internal hardware and bus architecture, which is beyond the scope of this article. It is a good idea to consult with your Enpterprise Services representative on placement of components of redundant systems.

Another useful exercise is to consider the higher level redundancies which support the cluster. Consider each node as half of a redundant system, and then determine where the single points of failure lie. It may be difficult or even impossible to eliminate all single points of failure from the data center, but it is usually easier to eliminate them if they are caught early in the development cycle.

The process of determining single points of failure involves tracing all the dependencies for each subsystem. For instance, the network may have a single point of failure if all the connections end up at the same switch, or a less severe single point of failure if all the networking cables leave the building through the same route.

It is important to analyze the independence of any critical subsystems, typically the power, network, and storage. The easiest method of tracing back dependencies is to consider the paths of all the wires. While this sometimes requires literally tracing the path of individual wires, it is usually possible to refer to floor plans, blueprints, wiring diagrams or the like. For instance, check to see if the power sources for each node are independent (some sites go so far as to attach both nodes to separate municipal power grids). The same goes for the networking. Because it is not always feasible to have independent, redundant components at every point throughout a data center (including where connections leave the building and even beyond), it is nearly impossible to entirely eliminate single points of failure. However, the impact of these points of failure can be reduced by assuring that they maintain a high level of availability and can be replaced quickly and easily.

Reduce Dependency on External Services

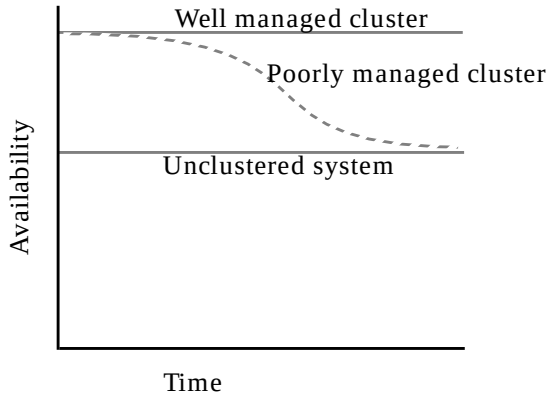
Another important aspect of configuring for availability is eliminating, or at least reducing, dependencies on external services, such as NIS or NFS mounts. The availability of the cluster is limited by the availability of any external services on which it is dependent. Ideally, there should be no dependence. However, sometimes architectural changes require dependencies outside of the cluster. If this is the case, it is important that all dependencies are well documented and understood by administrators.

Dependencies on external services can also be reduced by duplicating critical information locally. In particular, clusters should always have local `/etc/hosts` files, as well as `/etc/passwd` and `/etc/shadow` files. Many of the other NIS maps can also be duplicated in local files, though it is important to make sure that the maps are the same on both nodes. The `/etc/nsswitch.conf` file should list the files as the first source of information for as many maps as possible. In some cases, this is required by the cluster architecture.

Maintaining the Cluster

Even after the most optimal installation that follows all of the suggestions listed above, clusters need to be carefully monitored and maintained if they are to keep their high level of availability. In a properly functioning cluster, when a failure occurs the functionality of the failed component will be taken over by a redundant component. However, some effort is required to assure that the latent components will provide redundancy when needed. If latent components fail and are not replaced (either because the failure is unrecognized or unheeded), the availability of the cluster drops. For example, if a backup interface for a NAFO group fails and the system administrator fails to notice it, the availability of the formerly redundant NAFO group is reduced to the availability of the single active interface. FIGURE 3 shows the effects of unresolved latent failures on cluster availability. Note that this only shows the effects of undealt with latent failures on availability, other poor administrative practices could lower the availability of the cluster even below that of a single system.

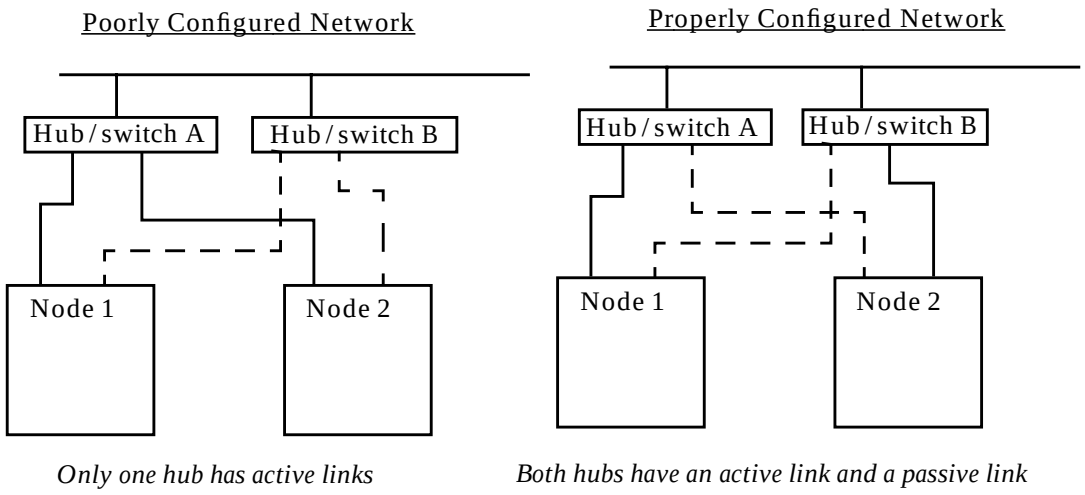
FIGURE 3 Effects of Latent Failures on a Poorly Managed Cluster



Monitoring some cluster components is relatively easy. The health of the nodes can be checked using a platform monitor such as Sun™ Management Center (SunMC), the status of the disk groups can be found using the appropriate volume manager tools, and the status of the logical hosts and data services can be checked using the cluster monitoring commands outlined in the following section. The state of NAFO groups and the private interconnects can also be checked using cluster commands. Of course, these tools need to be used regularly to be effective.

Unfortunately, some components of a cluster are not so easy to monitor. Specifically, the latent interfaces of a NAFO group can not be monitored directly. The active interface in a NAFO group is plumbed and able to send data, but the inactive interfaces are unplumbed and unused. It is important to be aware that this is the case, although it may or may not be practical to test the latent parts, depending upon your environment.

FIGURE 4 A Poorly Configured Network and a Properly Configured Network



Another problematic latent failure could occur in hubs or switches. Many hubs (and some switches) are not directly monitorable, which means a failure in a latent hub or switch could go unnoticed. The “Poorly Configured Network” diagram in FIGURE 4 shows a configuration with an active hub (which has the active interfaces from both nodes connected to it) and a latent hub (which has the latent connections from each node attached to it). It is unlikely that a latent failure would be noticed in this configuration, since no active connections go to the latent hub. The example could also apply to switches. Consider what would happen if the latent hub failed and the failure went unnoticed. If the active NAFO group interface in Node 1 fails, the latent interface will attempt to become active and fail because the hub it is connected to is not working properly. The result is that a node failover, rather than a NAFO failover, will be triggered, causing a situation that requires more downtime.

An even worse case occurs if the active hub fails after the latent hub failure goes unnoticed. In this case, all network connections from both nodes are unable to reach the network, resulting in a cluster which is essentially unusable until it is repaired. Because this is an unexpected problem, diagnosis is likely to take a fair amount of time.

A solution to this problem is simply to assure that both hubs (or switches) have active connections. This configuration assures that error messages are generated in the case of a hub (or switch) failure, which reduces the chance of such a failure going unnoticed. This configuration can be easily accomplished if one node has the active and latent connections switched, as shown in "The Properly Configured Network" diagram in FIGURE 4. This way, both hubs (or switches) have an active connection and a latent connection.

Cluster Monitoring Tools

Logs and informational commands can be used to monitor the health of a cluster. Understanding monitoring tools and what they offer is crucial to being able to identify and solve cluster problems quickly. Using the proper log files is crucial for identifying the causes of problems that have already occurred.

There are several types of commands that relate to monitoring Sun Cluster status and configuration. Some of these commands display general information about the cluster configuration and status, while others display information about only specific facets of the cluster configuration and status. There is also a GUI based tool for monitoring the cluster.

When trying to find cluster information, it is often useful to understand what each command is designed to do. In general, cluster monitoring commands are divided into two types: those that display the cluster configuration and those that display the current status and state. There are specific commands for finding information about PNM and data services. General commands provide most of the other useful cluster information. The `hastat` command also includes the latest system messages. For an overview of cluster monitoring commands, refer to TABLE 1.

TABLE 1 Cluster Monitoring Commands

Type of Information	Command to list configuration	Command to list status

General
(Logical hosts, data services, node status, private interconnects, etc.)
	`scconf -p`	`hastat`
		and
		`get_node_status`

PNM Groups	`pnmset -pntv`	`pnmstat -l`
Data Services	`hareg -q`	`hareg`

In addition, there is a graphical tool for monitoring cluster status. The Sun Cluster Manager (SCM) is a graphical interface which can be operated over the Web. It displays almost all of the cluster state information given by the commands above, and often provides a more convenient way of viewing the data. However, the SCM display can differ from reality under some circumstances (as described in the Sun Cluster 2.2 Release Notes); thus it is best used in conjunction with the other cluster monitoring tools. The SCM program provides an easy way to view a great deal of cluster configuration information, and is often easier to understand. In addition, because SCM provides clear indications when problems occur, it is often useful for identifying problems. Once problems are identified, it is often more useful to use the command line tools to track them down and fix them, since, in this case, the administrator will usually be working from the command line to fix the problem. It is also important to keep in mind that, unlike `hastat`, SCM takes its log information directly from the `/var/adm/messages` files. As a result, SCM will display old logs if messages are being redirected to another location.

Cluster Logs

Another crucial tool for monitoring the status of a cluster is the log files. Having a custom or prepackaged log monitor will make monitoring the logs much easier. Almost all cluster messages that occur outside of intentional reconfiguration indicate some sort of problem, so it is a good idea to have the log monitor flag all of them.

Most of the cluster warnings, errors, and notices go to the `/var/adm/messages` file. It is a best practice to save a copy locally to `/var/adm/messages`, but also send a copy of the logs to a remote host (for instance, the administrative workstation). This can be accomplished by modifying the `/etc/syslog.conf` file.

Detailed reconfiguration logs can be found for both the Cluster Configuration Database (CCD) and the Cluster Membership Monitor (CMM). The logs for the CCD reside at `/var/opt/SUNWcluster/ccd/ccd.log`. The logs for the CMM reside at `/var/opt/SUNWcluster/scadmin.log`. Both these logs can be used to understand problems that occur during reconfiguration. Because some of the warnings that deal with quorum devices or with terminal concentrators could be accidentally ignored when they come on the console, it is important to scan these logs occasionally to make sure the cluster membership and failure fencing infrastructure are healthy and operational.

A crucial component of managing cluster logs is verifying that the timestamps on events from the different nodes correspond. The internal clocks of servers naturally vary slightly over time due to differences in heat and other environmental conditions, electrical characteristics of the machine, or even manufacturing variance of the chips. Over time, these imperceptible differences can lead to considerable variation of the times between cluster nodes. When using timestamps to correlate events on two nodes, these differences can make the task unnecessarily difficult. This is complicated by the fact that an administrator cannot directly change the time

on a running Sun Cluster node. Administrators should never make changes directly to cluster time using `date`, `rdate`, `ntpdate`, or the like. Sudden changes to the time on a cluster node caused by running such commands could lead to errors. It is important to verify that no `crontab(1)` entries use `ntpdate(1m)` or any of the above commands.

A solution is to use Network Time Protocol (NTP) which allows consistent time to be maintained between cluster nodes without administrator intervention. NTP corrects drift by making small changes over a period of time. While cluster nodes cannot be configured as NTP servers, they can be NTP clients. In order for the cluster nodes to have time served to them, it is necessary to set up an NTP server using `xntpd` (the NTP daemon). In many large data centers, an NTP server is already configured; hence, setting up an NTP server is unnecessary in most cases. An easy option for setting up an NTP server is using the administrative workstation. While doing so may not provide the cluster nodes with an accurate representation of the actual time, it does keep the nodes consistent with each other, although not necessarily with any sort of time standard. Keep in mind that making nodes consistent with each other as well as with time standards could certainly help in situations where understanding cluster problems requires understanding the actions of clients or hardware outside the cluster.

Fixing Failures

Once a failure has been detected, it is important to fix the part as soon as practical. The details of fixing specific failures are beyond the scope of this article, but there are a few things to be aware of when replacing cluster components. In most cases, so long as the failed part is replaced with an identical part at the same physical location, the cluster will not need to be reconfigured to recognize the new part. Notable exceptions to this rule are quorum devices. If the quorum device is replaced, the cluster needs to be configured to use the new quorum device using the `scconf -q` command. This replacement is necessary because quorum device configuration is tied to the serial number of the device. Because additional administration is required when the quorum device is replaced, it is important to physically label or mark it. It must be immediately apparent when the quorum device is switched in order to prevent situations where a quorum device is replaced without updating the cluster configuration. If a quorum device is replaced without properly updating the cluster configuration, cluster membership determination could be affected in some failure cases, ultimately reducing availability.

If failed components are replaced with non-identical parts, the cluster configuration needs to be updated appropriately. This updating is required because the cluster framework will not automatically recognize the new devices.

Whenever changes are made to the cluster databases using the `scconf` command, it is important to back up a copy of the databases. A more complete discussion of cluster change and the cluster databases is in the “Managing Cluster Change” section.

Failover-drills

While running failover drills will cost down-time, they can be scheduled for periods of low usage. In fact, failover drills are important enough to justify this downtime in most cases, as they assure that the configuration and the staff know how to deal with a failover. How often failover drills should occur, or even if they should happen at all, depends entirely on the environment. In some cases, there is appropriately scheduled downtime in which failover-drills can occur. In other situations, the staff can learn the procedures involved in failover scenarios by running the drills on non-production systems. At the very least, a variety of failover drills should be run after the system is installed, but before it is put into production. This approach reveals problems before the system is in production. If expected or unexpected maintenance outage is needed, it may make sense to run a failover-drill as part of the already existing outage. Even though running failover drills on a regular basis may not make sense in all environments, the decision to not use them should be a well-informed, strategic choice.

Managing Cluster Change

Just like any other production environment, change control procedures are crucial to a clustered environment. However, special consideration must be taken to assure that cluster configuration is consistent across all nodes. While it is unlikely that any inconsistencies will be accidentally introduced into the cluster, the costs of such an inconsistency can be disastrous. In an extreme case, inconsistencies could lead to downtime for as long as it takes to track down the inconsistency. Because inconsistencies are so dangerous, it is important to take every possible effort to avoid them.

Understanding the various cluster configuration databases is an important part of administering a cluster, planning for change, and dealing with crises in a clustered environment. There are several files which maintain cluster state and configuration information, the most important of which are the CCD files and the CDB files. The `/etc/pnmconfig` file is also important for managing the cluster configuration because it determines the NAFO group configuration. These cluster configuration databases contain different subsets of the cluster configuration information, as described in TABLE 2. The exact contents of a given cluster database depend upon

architectural considerations. The dynamic CCD is kept consistent automatically on all nodes, while it is possible for an administrator to accidentally introduce inconsistencies in the other cluster databases.

In general, there are 3 processes that need to be followed to manage changes in cluster configuration: making proper changes and keeping documentation, backing up the cluster configuration databases, and verifying the consistency of the databases with backups and other nodes.

TABLE 2 Partial Contents of Cluster Configuration Database Files

Cluster Config. Database	Contents
init CCD	■ Dynamic CCD initialization Data ■ Cluster name ■ Number of possible nodes ■ Node names and id numbers
dynamic CCD	■ Logical host and Data Servicestate/configuration ■ TC/SSP information ■ PNM configuration and status ■ Cluster Membership
CDB	■ Cluster membership and failure fencing configuration (quorum devices, terminal concentrator, etc) ■ Private network addresses and interfaces ■ Node names and ids ■ Local logical host config ■ PNM tunables
`pnmconfig`	■ Initial PNM configuration

Make Proper Changes and Keep Documentation

Since the CDB and init CCD are not distributed, any commands that modify them should be run on all nodes, whether they are currently in the cluster or not. Because the init CCD holds the configuration information for the dynamic CCD, changes to the init CCD must only be accomplished when the cluster is stopped. Commands which change the dynamic CCD must only be run on one node, since it is kept consistent automatically. Although it is not a requirement, all changes to the dynamic CCD should be made when all the nodes are in the cluster. This will greatly reduce the risk of problems with incorrect CCDs.

In addition, changes to the cluster databases should always be made using the appropriate commands. Administrators should never edit the cluster configuration database configuration files by hand.

Any changes to cluster configuration should follow whatever change control procedures are in place. These changes must then be documented, including hardware and topology changes (drives added or changed network connectivity), as well as cluster configuration changes accomplished in software (using scconf, pnmset, or other Sun Cluster commands). When logging changes made through Sun Cluster commands, it is important to note which nodes the commands were run on. Logging the usage of the `scadmin` command to bring nodes up or down is not usually necessary because this information does not change the configuration files and can be easily ascertained by running get_node_status.

A good way to keep the dynamic CCD consistent in cases where changes are made with only one node present, is to use a shared CCD. A problem could occur if there are inconsistent CCDs on different nodes because one group of nodes was in the cluster during a CCD update while the other group was not. If the updated nodes all leave the cluster and the unupdated nodes restart the cluster, the new cluster would use the unupdated CCD, which results in the cluster not using the most up-to-date configuration.

In 2 node clusters, using a shared CCD eliminates the problem of inconsistent CCDs. In nearly all 2 node configurations, it is a best practice to use a shared CCD because it removes the risk of discrepancies. A shared CCD requires 2 disks. For some low-end configurations, giving up 2 disks may be too extreme a cost for the benefit of a shared CCD. The details of implementing a shared CCD are covered in the Sun Cluster 2.2 Administration Manual.

Backup the Cluster Databases

The cluster databases should be properly backed up whenever changes are made. It is not necessary to have regular backups of the cluster configuration databases, so long as backups are made after changes and regular consistency checks against the backups are performed. The consistency checks will verify that no unregistered changes have been made since the last backup. This verifies that the backups are always current, even though backups are not made regularly.

Backing up the dynamic CCD is accomplished using the "`ccdadm` *`clustername`* `-c` *`checkpointfile`*" command (where *clustername* is the name of the cluster and *checkpointfile* is the file to save to), but this should only be done with all of the logical hosts in maintenance mode. While it is not required for the logical hosts to be in maintenance mode for a checkpoint, they must be in maintenance mode for the restoration. Errors or unpredictable behavior may occur if the state information you are restoring from does not match the current state of the cluster. In order to make certain that the cluster has the same state in either instance, have the logical hosts in maintenance mode during a checkpoint operation.

Backing up the init CCD and the CDB requires manually copying the files to a new location. Once the logs are archived locally, they should be copied from each node to a central location (for instance the admin workstation). Scripts could be used to automate this process. The names of each backup file should reflect the name of the node the file is from, as well as the date of the backup. Keeping old backups here for historical records allows for quick recovery from catastrophic failures.

Verify the Consistency of the Databases with Backups and the other Nodes

Another crucial aspect of managing the cluster configuration databases is assuring that all of the databases are consistent between nodes. Using the backup scheme described above can make this much easier. The `pnmconfig` file is not clusterwide, and thus the `pnmconfig` file on one node does not need to be compared with the `pnmconfig` file on other nodes. For the CDB and init CCD, the easiest way to check consistency is to use the `diff` command on the archived versions to make certain that all nodes are using the same databases. Inconsistencies in the timestamp line are acceptable, though it may be important to understand the cause of the discrepancy.

If you have local copies of the historical database files, it may be easier to check only the consistency of the database files between nodes after changes are made. In the absence of changes, it is easier to simply verify the current database against the archived copy to assure that no unregistered changes have been made.

For the dynamic CCD, the consistency of the database can be checked using the `ccdadm` *`clustername`* `-v` command, where *clustername* is the name of the cluster. This command checks that the local CCD copies on all running cluster nodes are consistent. It is a best practice to have all nodes in the cluster during changes that involve the CCD. However, sometimes this is not possible because of emergencies or commands which require nodes to be out of the cluster when they are run. If there are nodes out of the cluster during CCD changes, it is important to document them. It is also important to know which nodes were in the cluster during the last change so that the nodes with the most current CCD can be brought up first. As with the CDB and init CCD, check the CCD against a backup using `diff(1)` to verify that no unregistered changes have been made since the last archive.

Resolving Cluster Configuration Database Problems

While the suggestions above will help prevent inconsistencies during cluster change, they do not cover what to do in a situation where cluster configuration database problems occur. The following provides an overview of some possible cluster configuration database problems and how they can be resolved.

Because proper cluster databases are so crucial to proper cluster operation, problems with the cluster databases are noted through error and warning messages during cluster startup. The effects of having problems with each type of database differ.

The CDB is the first cluster database read during cluster startup. If the CDB does not exist, the cluster will not be able to verify its name and will not start. If a node tries to start up with a CDB that has a syntax error, the `startnode` command exits with an error.

If the CDB on a newly joining node is syntactically correct, but differs from that of the nodes already in the cluster, the reconfiguration will fail with an error.

In either case, the CDB can be restored from backups; they can also be `ftp`'d from another node onto the node that is having problems. This will assure that the database is syntactically correct and consistent with the other nodes. Note that in order for two CDBs to be considered consistent, they must be exactly the same. Adding an extra blank space, or even an extra comment to one file will result in an inconsistent CDB message.

If the init CCD has an invalid checksum or the init CCD on a newly joining node differs from the init CCD of the other nodes in the cluster, the joining node will panic. This prevents a node with an incorrect init CCD (and thus a potentially incorrect understanding of the cluster setup) from affecting the cluster.

If a node with a missing, inconsistent, or incorrect dynamic CCD joins an already running cluster with a correct dynamic CCD, the dynamic CCD of the new node is replaced by the dynamic CCD being used by the cluster, so there is no consistency issue. This is automatic in Sun Cluster 2.2.

If a node starts a new cluster and its dynamic CCD has an invalid checksum, the node will be allowed to join, but a warning will be displayed. In this case, CCD changes can not be made until a CCD quorum exists. A CCD quorum is achieved when valid CCDs from over half the possible nodes are present in the cluster. As a result, in a two node cluster, quorum is impossible if the first node to join the cluster has an invalid CCD. In this case the administrator has to restore the CCD on the first node. Using a shared CCD, which enables a mirrored, persistent, and shared copy of the CCD, will prevent this problem. Using a shared CCD is almost always a best practice in a 2 node cluster and is described in more detail in the *Sun™ Cluster 2.2 Administration* Guide.

Summary

The primary goal of an HA cluster is to provide an increased level of availability, so the primary goal of a cluster administrator is to maintain the cluster so this is possible.

During cluster setup and configuration, it is important to maximize independence. Cluster components should be configured for maximum independence, as should the cluster itself.

Once the cluster is operational, it is important to monitor the health of cluster components. Failures need to be fixed promptly. When the cluster is modified, steps need to be taken to assure that the changes are made properly. This requires proper change control, but also includes updating and verifying the cluster databases to account for the new configuration.

This article covered the life-cycle of a cluster from implementation to decommission, and included best practices for each stage of the cluster's implementation. In addition, this article provided an overview of the administrative and monitoring tools available as part of the Sun Cluster distribution. It also provided an overview of cluster management, along with methodologies and administrative best practices.

References

This article was developed in tandem with a soon-to-be-released Sun BluePrints™ book titled *The Sun Cluster 2.2 Environment: Fundamentals and Beyond* by Enrique Vargas, Joe Bianco, and David Deeths. This article was meant to provide a short introduction to cluster administration, while the book covers the subject in greater detail.

The standard reference for Sun™ Cluster administration is the *Sun™ Cluster 2.2 Administration* Manual.

Author's Bio: David Deeths

David Deeths is a member of the technical staff of Sun Microsystems' Enterprise Engineering group. He has been working at Sun for 3 years. His current focus is clusters, including doing some of the research, development, and writing for an upcoming Sun BluePrints titled "The Sun Cluster 2.2 Environment: Fundamentals and Beyond." David has degrees in Electrical Engineering and Cognitive Science from the University of California, San Diego.